TU WIEN Informatics

Autonomous Orchestration of Computing Continuum Systems through Active Inference

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

eingereicht von

Dipl.-Ing. Boris Sedlak, B.Sc.
Matrikelnummer 01529846

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dr. Schahram Dustdar

Diese Dissertation haben begutachtet:

Massimo Mecella	Juan M. Murillo Rodríguez

Wien, 25. April 2025

Boris Sedlak

TU WIEN Informatics

Autonomous Orchestration of Computing Continuum Systems through Active Inference

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Dipl.-Ing. Boris Sedlak, B.Sc.
Registration Number 01529846

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dr. Schahram Dustdar

The dissertation has been reviewed by:

Massimo Mecella	Juan M. Murillo Rodríguez

Vienna, April 25, 2025

Boris Sedlak

Erklärung zur Verfassung der Arbeit

Dipl.-Ing. Boris Sedlak, B.Sc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 25. April 2025

Boris Sedlak

Acknowledgements

To begin with, I had a wonderful time during my PhD, thanks to all the colleagues and friends that I made during this intense but incredibly productive period of my life. Although I had not originally planned to do a PhD, it was my former master thesis advisor, Ilir Murturi, who noticed that it might suit me, and my supervisor, Schahram Dustdar, who gave me the opportunity to pursue a PhD in this incredible group. Over the course of my PhD, I felt Schahram's confidence in my work grow, and how this inspired me to dig deeper, but also to gain a broader understanding, which eventually led to this thesis. At this point, I would also like to thank Uwe Zdun and Horst Eidenberger for their encouraging words during my Proficiency Evaluation, as well as Massimo Mecella and Juan Manuel Murillo for providing their expertise in agreeing to review this thesis.

Personally, I think I have been particularly fortunate with the network of postdocs I have had around me: Víctor Casamajor Pujol was my closest colleague on a day-to-day basis, who would spare no efforts to discuss whatever ideas came to my head; Praveen Kumar Donta introduced me to the world of academia, including all the little rules you need to play this game; Andrea Morichetta was always around as a coffee companion, but it took a joint research visit to China until he also took a major role in my growth as a researcher; and Pantelis Frangoudis answered all my questions when none of the others knew an answer. However, the DSG would not be the same without the combined force of Renate Weiss, Christine Kamper, and Margret Steinbuch, who would patiently support us on a daily basis. Naturally, this also counts for Alexander Knoll—a most iconic and inspiring young man—who would brighten up your day when it got dark.

While I have enjoyed the creative and supportive working environment at DSG, a healthy work-life balance requires friends and family who equally support you throughout this journey. First and foremost, I would like to thank Laura Dusl, who has always been there for me when I was on the verge of losing myself in my work. From time to time, a creative mind also needs a break—after working up a sweat, there was no better place to recover than among the Sauna Herrenrunde group. Thank you for being there!

The research presented in this thesis was funded by the EU's Horizon Research and Innovation Programme under grant agreement No. 101070186. Views and opinions expressed are however those of the author only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them. EU website for Teadal: https://teadal.eu/

Kurzfassung

Als Gesellschaft sind wir mittlerweile in hohem Maße von einer Vielzahl an Internet of Things (IoT) Geräten umgeben, die all jene intelligenten Systeme ermöglichen, mit denen wir tagtäglich interagieren. Die meisten dieser Anwendungen erfordern die Verarbeitung von IoT-Daten mit geringer Latenz, was den Umstieg zu Edge-Computing, also der Verarbeitung in nächster Nähe zur IoT Quelle, einläutete. Dennoch hat sich gezeigt, dass Edge-Computing das traditionelle Cloud-basierte Computing nicht ersetzt. Vielmehr hat sich in der Praxis eine Kombination mehrerer Rechenebenen als vorteilhaft erwiesen, die zu einer zusammenhängenden Plattform – dem sogenannten Computing Continuum (CC) – führt. Das CC ermöglicht die Verteilung und Ausführung von Microservices (dt. Mikrodienste) entsprechend ihrer individuellen Anforderungen. Um diese Anforderungen sicherzustellen – spezifiziert als Service Level Objectives (SLOs) – erfordert es schnelle Reaktionen auf das Laufzeitverhalten; daher muss diese Entscheidungslogik auch von den Services dezentralisiert ausgeführt werden. Allerdings ist dies problematisch, da die Services keine globale Übersicht darüber haben, wie sich ihre Aktionen auf abhängige Dienste auswirken, was die Erfüllung ihrer SLOs gefährdet. Darüber hinaus werden gängige Skalierungsmaßnahmen, wie die Bereitstellung zusätzlicher Ressourcen, selten von heterogenen Edge-Geräten unterstützt; die Definition eines benutzerdefinierten Skalierungsverhaltens pro Gerätetyp erscheint nicht zielführend. Um Microservices vor Umgebungsdynamiken zu schützen, ist es daher erforderlich, flexible Skalierungslösungen zu entwickeln, die den genauen Kontext jedes Services berücksichtigen.

Zur Schließung dieser Forschungslücke präsentiert diese Thesis eine Lösung für die autonome Orchestrierung von CC-Systemen, welche sich im Laufe der Zeit weiterentwickelt, um den Anforderungen aller eingebetteten Komponenten gerecht zu werden. Unterstützend wirkt dabei das Konzept der Active Inference (AIF), welches aus den Neurowissenschaften stammt und darauf abzielt, die Interaktionen zwischen einer Komponente und ihrer Umgebung zu modellieren. Durch gezielte Erkundung der Umgebung identifiziert AIF externe Faktoren, die sich auf die lokale SLO-Erfüllung auswirken, was erklärt, warum ein SLO zu einem bestimmten Zeitpunkt verletzt wurde. Umgekehrt quantifizieren wir, wie sich lokale Aktionen unter bestimmten Bedingungen auf die Umgebung auswirken, was bei der Ermittlung der optimalen Elastizitätsstrategie hilft. Gemeinsam verwenden wir dies, um ein kohärentes Modell der Verarbeitungsumgebung zu erstellen, mit dem sich abschätzen lässt, wie sich die Aktionen eines einzelnen Dienstes (z.B. seine Elastizitätsstrategien) auf die gesamte CC-Architektur auswirken. In der Folge lässt sich auch der erwartete

Ressourcenverbrauch ableiten, was die Möglichkeit, Dienste auf dem selben physischen Geräten auszuführen, erheblich verbessert. Das kohäsive CC-Modell bildet den Kern dieser Arbeit; die dem CC innewohnende Dynamik führt jedoch dazu, dass solche Modelle mit der Zeit immer ungenauer werden, was sich wiederum auf die Qualität der abgeleiteten Orchestrierungsmechanismen auswirkt. In diesem Sinne stellt diese Arbeit auch ein funktionierendes Ökosystem zur kontinuierlichen Aktualisierung und Verfeinerung dieser Service-Interpretationsmodelle während der Laufzeit vor. Hervorzuheben ist, dass alle in dieser Arbeit vorgestellten Konzepte in physischen Testumgebungen implementiert und evaluiert wurden, was den Weg für zukünftige Evaluierungen in groß angelegten Umgebungen ebnet.

Abstract

Today, we are surrounded by an immense number of Internet of Things (IoT) devices that power the smart environments we interact with on a daily basis. Most of these applications require low-latency processing of IoT data, which heralded the shift of processing resources to the Edge. Nevertheless, Edge computing has not replaced traditional Cloud-based computing; rather, the combination of multiple computing tiers into a cohesive platform – called the Computing Continuum (CC) – has shown synergies. As such, the CC allows microservices to be distributed according to their individual requirements. Ensuring requirements – formulated as Service Level Objectives (SLOs) – requires rapid response to runtime behavior, so this logic must also be executed decentralized by the services. However, this is problematic because services lack a global view of how their actions affect dependent services, jeopardizing their ability to meet SLOs. Furthermore, common scaling actions, such as provisioning additional resources, are rarely supported by heterogeneous Edge devices; defining a custom scaling behavior per device type seems exhaustive. Thus, for protecting microservices from environmental dynamics, the CC requires flexible scaling solutions that take into account the precise context of each service.

To address this research gap, this thesis presents a framework for autonomous orchestration of CC system, which evolves over time to meet the requirements of all its embedded components. This behavior is driven by Active Inference (AIF), a concept from neuroscience that seeks to model the interactions between a component and its environment. Through exploration, AIF identifies external factors that impact local SLO fulfillment and explain why an SLO was violated at a specific time. Conversely, we quantify how local actions affect the environment under certain conditions, which helps identify the optimal elasticity strategy. Together, we use this to build a cohesive model of the processing environment, which can estimate how an individual service's actions (e.g., its elasticity strategies) affect the entire CC architecture. As such, it can also infer the expected resource consumption, which greatly improves the possibility to co-locate services in multi-tenant environments. While this cohesive CC model forms the core of this thesis, the inherent dynamism in the CC causes such models to become increasingly inaccurate over time, which in turn affects the quality of the inferred orchestration mechanisms. To that extent, this thesis also presents a working ecosystem for continuously updating and refining these service interpretation models at runtime. Notably, all concepts presented in this thesis were implemented and evaluated in physical testbeds, paving the way for embedding them in large-scale environments for subsequent evaluations.

Contents

Publications

The research presented in this thesis is partly based on the following peer-reviewed publications. A full list of publications can be found in my Google Scholar Profile[1]

- Boris Sedlak, Víctor Casamayor Pujol, Praveen Kumar Donta, Schahram Dustdar. "Controlling Data Gravity and Data Friction: From Metrics to Multidimensional Elasticity Strategies", in *2023 IEEE International Conference on Software Services Engineering (SSE)*, pages 43-49, 2023.

- Boris Sedlak, Víctor Casamayor Pujol, Praveen Kumar Donta, Schahram Dustdar. "Designing Reconfigurable Intelligent Systems with Markov Blankets", in *Service-Oriented Computing*, pages 42-50, 2023.

- Boris Sedlak, Víctor Casamayor Pujol, Praveen Kumar Donta, Schahram Dustdar. "Equilibrium in the Computing Continuum through Active Inference", in *Future Generation Computer System*, volume 160, pages 92-108, 2024.

- Boris Sedlak, Víctor Casamayor Pujol, Praveen Kumar Donta, Schahram Dustdar. "Active Inference on the Edge: A Design Study", in *2024 IEEE PerCom Workshops*, pages 550-555, 2024.

- Boris Sedlak, Víctor Casamayor Pujol, Praveen Kumar Donta, Schahram Dustdar. "Markov Blanket Composition of SLOs", in *2024 IEEE International Conference on Edge Computing and Communications (EDGE)*, pages 128-138, 2024.

- Boris Sedlak, Víctor Casamayor Pujol, Praveen Kumar Donta, Schahram Dustdar. "Diffusing High-level SLO in Microservice Pipelines", in *2024 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pages 11-19, 2024.

- Boris Sedlak, Andrea Morichetta, Yuhao Wang, Yang Fei, Liang Wang, Schahram Dustdar, Xiaobo Qu. "SLO-Aware Task Offloading Within Collaborative Vehicle Platoons", in *Service-Oriented Computing*, pages 72-86, 2024.

[1] `https://scholar.google.com/citations?hl=en&user=m0yHSA4AAAAJ`

- Boris Sedlak, Andrea Morichetta, Philipp Raith, Víctor Casamayor Pujol, Schahram Dustdar. "Towards Multi-dimensional Elasticity for Pervasive Stream Processing Services", in *2025 IEEE PerCom Work In Progress (WIP)*, 2025 (accepted).
- Praveen Kumar Donta, Ilir Murturi, Víctor Casamayor Pujol, Boris Sedlak, Schahram Dustdar. "Exploring the Potential of Distributed Computing Continuum Systems", in *Computers*, volume: 12, issue: 10, article: 198, 2023.
- Víctor Casamayor Pujol, Boris Sedlak, Yanwei Xu, Praveen Kumar Donta, Schahram Dustdar. "DeepSLOs for the Computing Continuum", in *Proceedings of the 2024 Workshop on Advanced Tools, Programming Languages, and PLatforms for Implementing and Evaluating algorithms for Distributed systems*, pages 1-10, 2024.

Furthermore, several topics were investigated together with students through their Master's and Bachelor's thesis under my supervision. A full list of topics investigated with students can be found on the Distributed Systems Group website[2]

- Marco Brandstätter, "Optimizing Health Indicators by Applying Causal Reasoning to Complex Health and Environmental Data", Bachelor Thesis (2024)
- Yana Peycheva, "Dynamic Processing Routes for Mobile Video Streaming Inference", Master Thesis (2025 – in process)
- Elias Huhsovitz, "SLO-Aware Parallelization of Video Stream Processing in Heterogeneous Edge Environments", Bachelor Thesis (2025 – in process)

[2] `https://dsg.tuwien.ac.at/team/bsedlak/`

CHAPTER 1

Introduction

Over the last decade, the Internet of Things (IoT) has found its way into everyday life, ranging from wearables that track vital parameters for smart health, up to smart cities that detect traffic accidents according to camera images, or highly frequented areas according to smartphone locations. These are just three examples of how the ubiquity of IoT devices has revolutionized our daily lives. To process IoT data with low latency and without exposing it to the internet, Edge computing [SD16] heralded a transition of processing resources into the vicinity of IoT devices. Nevertheless, Edge computing has not shown to replace traditional Cloud-based computing by any means; rather, combining these two into a cohesive platform has shown synergies [KMH+21]: use the Edge for low-latency operations, and use the Cloud for resource intensive tasks that require high availability. Research has named this platform the Edge-Cloud continuum, or more frequently, the Computing Continuum (CC), which is not limited to specific tiers; hence, it can also include Fog nodes, such as telecommunication stations.

When processing IoT data – commonly over an extended time period – one central question is always whether processing actually fulfills its stakeholders' requirements, e.g., in terms of end-to-end latency or data quality. These requirements are commonly formalized as Service Level Objectives (SLOs), which are continuously evaluated during runtime. In case an SLO is violated, e.g., the desired response time of a traffic application is exceeded during rush hours, the system can provision additional resources to recover the SLO fulfillment. This proved fruitful for Cloud systems, which can access a vast amount of virtualized resources. However, Edge devices are naturally limited by their local resources, which opens up a series of challenges: (1) how to protect SLO fulfillment against external influences, (2) how to co-locate processing services on one device without compromising their SLO fulfillment, and (3) where to deploy each of the services so that global SLO fulfillment is optimized. Furthermore, if the SLOs for CC components would be evaluated centrally, this would create a serious communication overhead for collecting

metrics in the Cloud; hence, another problem is (4) how to decentralize the requirements assurance to local inference, executed directly on Edge devices.

These and other challenges have recently been documented in the context of the CC [CPDM+23b, NRRC24, Pet21]; most notably, how the orchestration of CC systems poses new problems due to its highly-distributed nature and the volatility in the architecture: devices may join at any time, or new service instances might suddenly be needed. However, research on CC has not solved these problems yet, with recent works still marking out the capabilities of CC systems [JWTI23], or developing simulation environments [ATG+24] that coat some of these problems. Still, to date, researchers start to present scaling mechanisms for the CC [CDM+25, ZFFP24], or orchestration mechanisms for Edge devices, which use Machine Learning (ML) techniques to ensure SLOs [ZZL23] in multi-tenant devices [ZMC+22]. While this works on a local scale, these techniques do not evaluate dependencies between services – they lack a holistic representation of how CC components affect each other. To simplify the orchestration of CC systems, this calls for a flexible approach that answers (1) how to ensure SLOs on a local level, (2) how changes perpetuate in microservices architectures [VF23], so that (3) the global SLO fulfillment can be optimized both at design and runtime.

To address this research gap, this thesis offers a framework for autonomous orchestration of CC system, which evolves over time to meet the requirements of all its embedded components. This behavior is fueled by Active Inference (AIF) [PPF22], a concept from neuroscience that seeks to model the interactions between a component and its environment [KPP+18]. Through exploration, AIF identifies external factors that impact local SLO fulfillment; conversely, it quantifies how local actions affect the environment. Together, we use this to build a cohesive model of the processing environment, which can estimate how an individual service's actions (e.g., its elasticity strategies) affect the entire CC system. While this concept lies at the core of this thesis, the inherent dynamism in the CC causes such models to become increasingly inaccurate over time, which in turn affects the quality of the inferred orchestration mechanisms. To that extent, this thesis also presents a working ecosystem for continuously updating and refining the service models, so that the overall SLO fulfillment can be optimized throughout the CC.

1.1 Problem Statement

Computing architectures are quickly growing in size, which allows tackling new categories of problems. However, this also comes with a number of challenges due to their inherent complexity, in particular: (1) how can the implications and interactions of services be modeled so that defective components can quickly be pinpointed, (2) how to improve the adaptability of computing systems in case resources are limited, (3) how to ensure model accuracy throughout variable drifts, and (4) how to raise the confidence in AI mechanisms. In the following, these four aspects are elaborated in more detail to give a clear understanding of the open problems that this thesis does address.

Complex Service Interactions

Large-scale computing architectures, such as CC systems, can span multiple computing tiers with applications that provide ultra low latency on the Edge, up to high availability and vast resources in the Cloud. While such architectures promise to be the backbone of many distributed processing use cases [NRRC24], e.g., smart cities or smart manufacturing, this also comes with a price: complexity. In particular, connecting wider networks and higher numbers of applications (i.e., microservices) makes it difficult to build a global state, or take decisions that require one. Building a global state requires sophisticated messaging protocols to decrease the overhead, while sometimes a local state – comprising all the neighboring services – might suffice [KLM+23]. While it is simple to answer which services interact, e.g., according to networking information, the larger question is what are their impacts on each other [CQH19]. Namely, how does the quality of a service x affect subsequent service y, and how does x need to operate to fulfill y's SLOs? Not only does this require analyzing pairwise relations, but this also calls for a coherent system model that can express the dependencies through the entire service network.

Rigid Elasticity Models

One of the great benefits that gave rise to Cloud computing is the virtualization of computing resources [DGST11], which allows clients to rent a specified amount of them. By making this resource share dynamic, clients are able to cope for periodically changing workload patterns, e.g., mobility applications would usually require more resources during rush hours. However, providing additional resources has become an universal remedy in case of SLO violation [GMP+21], which is problematic for multiple reasons: (1) resources are not always the bottleneck, it may be, for example, the software architecture or communication overhead, (2) not all workloads can be efficiently parallelized, and most importantly, (3) it cannot always be assumed that additional resources can be provisioned [FFACP18], as is the case for Edge devices. This calls for more flexible ways to model the behavior of computing systems [LMF+25] – *if an application observes a, it does b, but only if c is given.* However, such models would be very dependent on the context of applications, which requires to build a (global) state, as discussed above.

Inaccuracy of One-Shot Training

To better understand the interactions between microservice applications, one option is to learn a model of their state transition probabilities or policies that optimize SLO fulfillment [DSCPD23a, THB12]. However, these models inevitably become outdated due to variable drifts in the processing environment. The fact that thousands of variables can be tracked also makes such models more error prone to inaccuracy because any variable could be the source of error [CPMM+23], or there might even be cases where not all aspects of a system were identified, leading to confounding variables. Hence, training a model for the CC cannot be a one-shot operation, but must involve sophisticated mechanisms for lifelong learning [SBIB+24]. However, this in turn poses multiple challenges to the

system architecture: (1) new training data must be collected continuously over the entire application lifecycle, (2) training a model requires dedicated resources, i.e., a share of the resources must be reserved for this task, or when no local resources are available, the training must be offloaded to remote nodes, and (3) if models for individual services are amalgamated into larger cohesive structures, every time a model is retrained, the new models must be distributed in the network so that it can then be reintegrated in the larger-scale model. Overall, this learning overhead must be minimized [DFP+24].

Limited Confidence in Black-Box Models

Deep learning has made rapid advances due to its astonishing abilities to capture intricate dependencies between variables, e.g., to classify images or spam emails [MPN+23b]. To the present day, this capabilities are not supported to the same extend by any other technology, but deep learning also comes with a price: limited interpretability. While the outcome of a neural network may be verified, e.g., it correctly identified the cat, the individual weights in a network cannot be interpreted that simply [GFB+23]. Conventionally, neural network do not provide any legal guarantee, which is why results must be verified by human agents to clarify the accountability in case of misclassifications. However, this combination of human resources with the lack of interpretability for neural networks becomes problematic in cases where the behavior of an AI must be debugged to find an error [RPN+22], or when authorities are audited to answer on data provenance and how specific samples did impact the decision-making.

1.2 Research Questions

The CC is designed to operate vast numbers of processing services on heterogeneous devices [DPD23]; however, there is a gap for orchestration mechanisms that can evaluate and enforce SLOs throughout multiple CC tiers [PD23]. This thesis aims to fill this gap. In the following, the outlined problems are summarized in three concise research questions (RQs) that will be addressed throughout the chapters of this thesis:

RQ.1 How to continuously assure the accuracy of service orchestration models so that reactive elasticity strategies provide maximum utility?

To fulfill processing requirements (i.e., SLOs) in the CC throughout external perturbations, it is necessary to dynamically adjust a system – commonly by using elasticity strategies. Considering that a system can be scaled in multiple dimensions [DGST11, FFACP18], e.g., quality or resources, ML techniques can be used to infer the most efficient elasticity strategy. However, in real-world systems, variable distributions can change at any point [LWL+23]; hence, any interpretation model, ideally causal, used to find SLO-fulfilling elasticity strategies for individual services (**RQ.2**) or their compositions (**RQ.3**) must be continuously adjusted according to new observations. Thus, services could avoid sub-optimal strategies when the QoS is compromised due to concept drifts, temporary

perturbations, or confounding variables. In the following, we elaborate further under which scenarios these three undesirable phenomena can occur:

1. *Concept drifts* are a commonly reported problem in ML [SCQC23] that can occur due to non-iid distribution of sampled data or natural processes, like machine erosion. This promotes *lifelong learning* to capture changes in the distribution [BXA+22, SBIB+24]; feedback loops, such as in [DGH21], could be a valuable extension that allows agents in the CC to continuously ensure accuracy.

2. *Temporary perturbations* are commonly observed when monitoring services that are exposed to fluctuating client behavior [NKFW19], e.g., increased taxi demand due to rainfall. If the system has a way to observe the rainfall it can learn to adjust to these dynamics in the environment.

3. *Confounding variables* are factors that have an impact on a process, but for which the system cannot (yet) account for. Continuing the above example, if a system has no understanding of the weather as an external factor, it cannot learn the cause of the increased client demand. Hence, systems extend their list of considered variables, as done for feature evolvable streams [CL24].

These examples describe three fundamental problems that occur during real-world processes; **RQ.1** thus aims to find answers to them. Any answer to this would allow to make decisions based on recent and accurate assumptions, while evolving over time.

RQ.2 How to efficiently choose between elasticity strategies by quantify their impact on both the SLO fulfillment and underlying processing hardware?

When aiming to ensure requirements for the CC, the bottom-up approach is to start from the smallest unit that should be supervised: an individual processing service [CPSX+24]. Suppose this service gets constrained by a set of SLOs (e.g., response time), the question is whether the service will be able to fulfill these SLOs given its available resources. Hence, the type of hosting device has clear implications on the potential SLO fulfillment [PNM+22], e.g., a CV task might largely benefit from an available GPU. Given the amount of heterogeneous resources in the CC, estimating the potential SLO fulfillment of a device at a certain device type would allow: (1) optimizing service deployment at design time by scheduling workloads to devices that likely fulfill their requirements [MPN+23a], or (2) adjust services elastically during runtime according to dynamic changes [CLPNR22].

While scheduling and runtime adaptation are two well-studied areas, they lack a holistic representation that answers *why* a specific solution (e.g., service configuration or load distribution) fulfills SLOs. Thus, it is difficult to empirically debug the decisions made by agents or explain produced results to non-technical stakeholders [MCM19]. Further, deep learning models are prone to concept drifts that occur when slightly adjusting the problem [SCQC23], e.g., by changing SLO thresholds. This impedes model transfer between different problem domains; to that extent, **RQ.2** aims to provide interpretable

service models that allow inferring the expected SLO fulfillment in heterogeneous and volatile processing environments, such as CC systems.

RQ.3 How to model the interactions and dependencies between microservices to estimate the impact they have on each other's SLO fulfillment?

Microservice architectures commonly form sequential pipelines, where the output of one service becomes the input of another service [VF23]. While service compositions allow building more advanced solutions, complex service interactions obfuscate the expected SLO fulfillment of individual microservices [SPDD24d]. Whenever dependent services can take a multitude of states, these can change the expected SLO fulfillment entirely. Hence, modeling implications and dependencies between services improves the system's elasticity; as a result, SLOs can be achieved through collaboration between devices.

However, existing solutions [ATG+24, JWTI23, NRF+22] for service deployment in heterogeneous CC environments do not consider the precise resource consumption of individual services; hence, co-locating multiple services at one device has unpredictable implications for their SLO fulfillment. Further, in microservice chains, services can cause SLO violations at successive services [PNM+22]: for instance, a service A might cause the latency SLO of a subsequent service B to fail, or the quality provided by A already makes it impossible for B to fulfill its quality SLO. Under these circumstances, modeling the dependencies between services (e.g., in terms of latency and quality) allows inferring how actions or states of one service impact another. Hence, **RQ.3** aims at providing an overarching service representation that can optimize global SLO fulfillment.

1.3 Scientific Contributions

This thesis has three main contributions, each consisting of multiple sub-contributions, which address the presented research challenges. In the following, we summarize the three contributions and indicate the respective chapter(s) where they are developed. Together, the three contributions form a cohesive framework, as illustrated in Figure 1.1.

C.1 Continuous Model Training for Service Interpretation

The goal to sustain – or fulfill a set of SLOs – is inherent in human cognition for thousands of years; hence, we use AIF [PPF22], a concept from neuroscience, to train self-evidenced agents [SPDD24a, SPDD24b], i.e., they autonomously ensure SLOs by resolving uncertainty in their understanding of the processing environment.

To that extent, we built a service interpretation model from sensory information (i.e., service metrics), which expresses internal dependencies between service variables (**C.1.1**). To ensure model accuracy throughout variable drifts, we developed a continuous exploration mechanism (**C.1.2**) – essential for volatile and dynamic environments. In case the resulting SLO fulfillment is below expectations, we adjust model training (**C.1.3**).

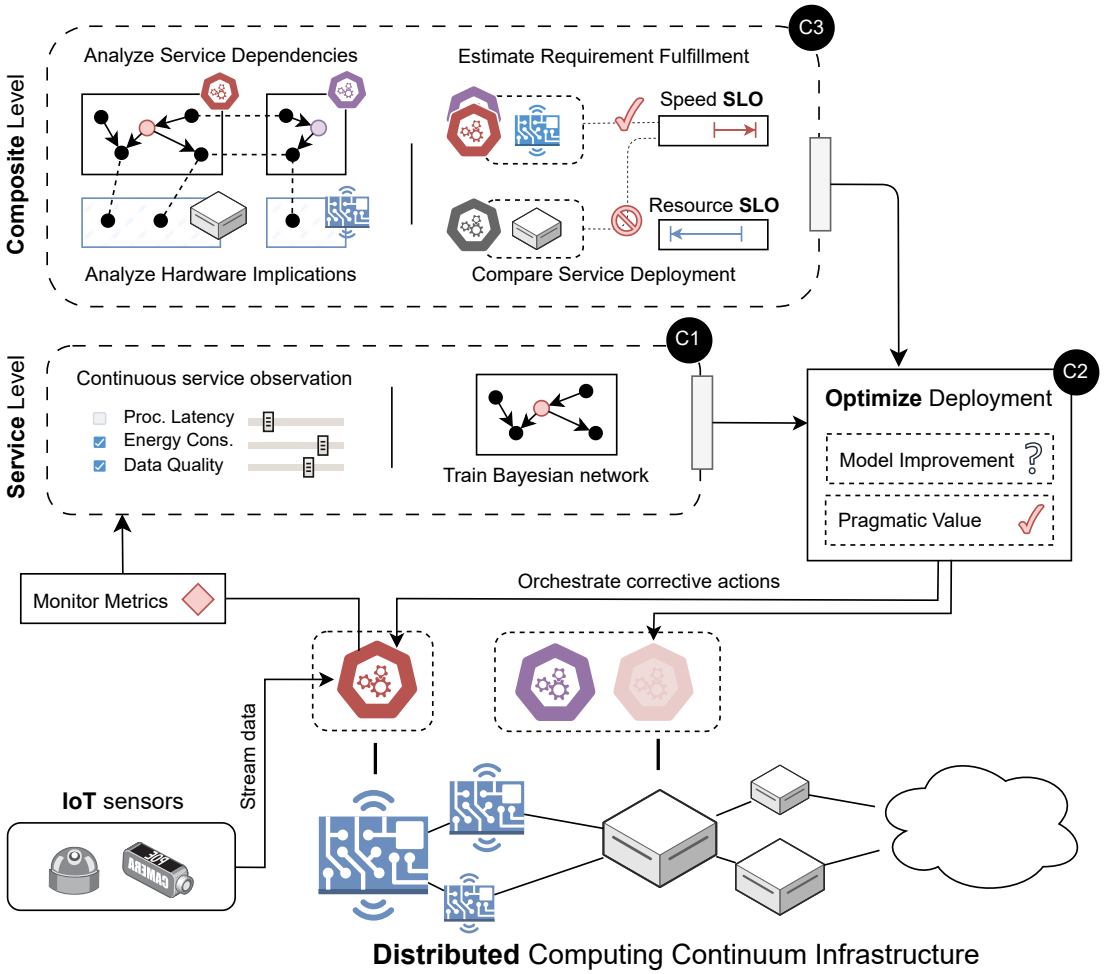


Figure 1.1: Relation of the scientific contributions: (**C.1**) training service representations from processing metrics, which are used to (**C.2**) continuously fulfill SLOs by adjusting the service deployment; (**C.3**) models are composed to quantify implications between dependent services, used to optimize global SLO fulfillment throughout CC systems

C.1.1 Bayesian Network Learning To express how service variables are related, e.g., video *resolution* influences processing *latency*, we train Bayesian Networks (BNs) from processing metrics [SPDD23]. For this, we extract and accumulate metrics during runtime, which need to go through some transformations (e.g., discretize by binning) before we train the structure – a Directed Acyclic Graph (DAG) – and then the conditional probabilities. This methodology is first developed in Chapter 4; it is then refined in Chapter 6, where we evaluate the complexity [SPDD24b] of training BNs.

C.1.2 Continuous Exploration To continuously ensure the accuracy of BNs, we developed AIF agent [SPDD24a] that evaluates the expected model improvement from different actions. Unlike strategies like ϵ-greedy, this precisely weights the expected value from exploration vs. exploitation in each iteration. We evaluated our agent extensively in multiple scenarios [SPDD24b, SPM$^+$24], i.e., video and lidar processing, where we showed its robust accuracy despite external perturbations. This is developed in Chapter 6.

C1.3 SLO-Aware Model Retraining Complementarily to the continuous exploration, we investigated ways to adjust the training interval according to current SLO fulfillment [SMW$^+$24]. This meant accelerating the training intervals in initial stages, when the learned model is not yet representative, or when variable drifts unexpectedly disturbed the inference. Thus, the SLO-aware model retraining did balance the training overhead according to the urgency. This feature is developed in Section 5.3.

C.2 Evidence-Based Runtime Adaptation of Processing Services

To meet the expected service quality in dynamic processing environments, the processing services must undergo runtime adaptations – commonly through elasticity strategies. However, to improve the reactivity of scaling operations, e.g., when uncertain if there are idle scaling resources, we create an flexible behavioral model [SCPDD23] that picks elasticity strategies according to the context in which a service is embedded. For this, we use BN models (**C.1**) to find the adaptation with the highest expected utility.

To that extent, we develop a multi-dimensional scaling solution (**C.2.1**) that evaluates composite SLOs and contextual factors (e.g., resource or quality range) to find the best elasticity strategy. Inferring actions from BNs provide another benefit: it is possible to empirically interpret why a certain action was chosen at a particular time (**C.2.2**).

C.2.1 Multi-dimensional Elasticity Strategies By using the BN of a service, we obtain a transparent view of its internal variable relations [SPDD23], for example, given that a service fails to provide its expected *throughput*, decreasing *resolution* recovers the respective SLO. The notable benefits of our approach are: (1) we identify the strategies as part of the service parameters when learning the BN [SPDD23], i.e., no prior expert knowledge is required, and (2) instead of incrementally scaling a parameter, i.e., setting *resolution* $\pm$ 100, we scale parameters to absolute values, e.g., set *resolution* $\leftarrow$ 720,

which converges faster to the optimal values [SPDD24b]. While this methodology is first described in Chapter 3, the inference mechanism is implemented in Chapter 4.

C.2.2 Interpretable Decision-Making Given the trained BNs, we verified the learned variable relations – which matched our expert knowledge – and showed how the behavior of AIF agent could be empirically debugged [SPDD24b] – answering why a certain action was taken at a specific time. For this, we provided a mechanism that starts from a high-level goal, e.g., minimize energy consumption, from which the respective lower-level SLO thresholds are inferred [SPDD24d]. For stakeholders, this removes the burden of fidgeting with SLO thresholds, as the remaining system would be orchestrated according to their preference. These mechanisms are developed in Section 5.2 and Chapter 6.

C.3 Collaborative Computing Continuum Framework

To ensure the processing requirements of individual services, we have presented mechanisms that allow inferring the optimal elasticity strategy according to the local service context (**C.2**). However, microservices are severely impacted by the QoS provided by their predecessor [Pet21], e.g., their latency or data quality; in such scenarios, an individual service might not be capable of recovering its local SLO fulfillment. Hence, to increase the scope of elasticity, we train higher-level structures [SPDD24c, SPDD24d] that optimize the global SLO fulfillment through coordinated service orchestration.

To that extent, we develop an overarching service representation (**C.3.1**) that quantifies the dependencies between microservices – *will the subsequent service be able to fulfill its SLOs with given resources?* – this composite model is used to orchestrate the services through a collaborative inference mechanism (**C.3.2**). To speed up onboarding of new devices, we convert BN models according to heterogeneous device capabilities (**C.3.3**).

C.3.1 Overarching Service Representation While individual services were represented by their BNs (**C.1**), we evaluated two ways to create an overarching microservice representation [DSCPD23a]: (1) by training it directly from one composite data set [SPDD24d], or (2) by merging their respective BNs [SPDD24c]. In both cases, we provided a composite BN that shows the conditional impact of adjusting one service (e.g., more threads, or more resources) on dependent services. The basic mechanism for this was outlined in Chapter 2, while it was implemented in Sections 5.1 and 5.2.

C.3.2 Collaborative Orchestration Mechanism Given the cohesive service model, we raise the overall SLO fulfillment by (1) deploying microservice pipelines over heterogeneous resources according to the expected SLO fulfillment [SPDD24c], (2) inferring SLO thresholds for predecessors according to local processing requirements [SPDD24d], and (3) offloading tasks between clients according to mutual SLO improvement [SMW+24], i.e., shift load so that global SLO fulfillment is optimized. These mechanisms are developed throughout Chapter 5; each of them improves collaboration between services.

C.3.3 Merging BNs for Transfer Learning While CC systems are characterized by their heterogeneity, i.e., different processing services or hardware, there are many possibilities to embed multiple instances of the same hardware or of the same processing service [DMCP+23]. To quickly raise the SLO fulfillment when starting new service instances, or when onboarding new device types, we provide the following: (1) we collect metrics at a mutable training device, where a BN update is computed and distributed to all respective services [SMW+24], and (2) we provide tailor-made BNs for novel device types, where the respective model is created by merging existing models with most similarity [SPDD24b], i.e., according to a computed hardware distance value. These mechanisms are developed in Section 5.3 and in Chapter 6.

CHAPTER 2

Behavioral Models for the Computing Continuum

Computing paradigms have evolved significantly over the last few decades, moving from large room-sized resources (processors and memory) to incredibly small computing nodes. Currently, Distributed Computing Continuum Systems (DCCS) unleashes an era of computing that unifies various computing resources, including Cloud, Fog/Edge computing, the Internet of Things (IoT), and mobile devices, into a seamless and integrated continuum. This platform provides a holistic solution to meet modern computing needs; however, there remain various research challenges, particularly in the domain of governance and orchestration. While Cloud computing has a centralized view on the state of the system, ensuring Service Level Objectives (SLOs) throughout a vastly distributed computing architecture opens new gaps in research. However, infrastructure providers, applications developers, and multiple tenants at shared computing nodes all have different, if not competing SLOs that must be supported. This calls for novel approaches, inspired by neuroscience, that allow to analyze the dependencies between DCCS components. Thus, it becomes possible to infer how dependent components impact each other and how they must act to fulfill each others SLOs. This is modeled through a behavioral Markov blanket – a probabilistic view into the operation of individual software components.

The remainder of this chapter is organized as follows: Section 2.1 first discusses the emergence of DCCS, current research challenges, and promising application scenarios; Section 2.2 presents different types of SLOs that must be supported in DCCS; and Section 2.3 introduces Markov blankets, which are continuously developed throughout this thesis to engineer the behavior of DCCS.

2.1 Distributed Computing Continuum Systems

DCCS[1]are systems built of a large variety of networked heterogeneous computing devices, which are used to process data generated by devices such as sensors, mobile devices, and IoT devices. Through its integration of cloud, edge, and IoT resources, it enables efficient, real-time, and dynamic computations to meet the needs of today's diverse applications [DSCPD23b, BDF+20]. It performs computations by distributing the workload across multiple devices in the system. Each device performs a portion of the computation, and the results are combined to produce the final output. This allows for faster processing times and increased scalability. With DCCS, computations are accomplished efficiently while adapting to changing demands and optimizing resource utilization outside traditional boundaries [DPD22]. This resource allocation is based on factors like resource proximity, computational capability, and prioritizing time-sensitive tasks. Depending on the task, real-time responses may be offloaded to edge devices, while complex analytics may be conducted in the cloud by default. This dynamic distribution of tasks enhances system performance, processing efficiency, and latency and reduces latency.

As an example of a DCCS architecture, consider the scenario depicted in Figure 2.1: smart devices are embedded in the IoT and edge domain, which perceive their environment through sensors, e.g., temperature or humidity. Data that cannot be processed directly in the Edge tier, e.g., due to limited processing power or battery, can be forwarded to higher tiers, such as the Fog and Cloud. While higher tiers face increasing latency to the data sources, they are at an advantage when it comes to collecting data for training meaningful ML models, which can be shared throughout the network.

2.1.1 Research Challenges in DCCS

DCCS offers numerous benefits and has the potential to transform modern computing, but they are not without challenges. In this section, we outline open research gaps; some of these are addressed in this thesis, for which we indicate the respective chapter.

Interoperability

DCCS is multi-proprietary. This means that the infrastructure resources and their associated middle-ware layers belong to different organizations. One can imagine an application running some services in-house, some services with high computational needs in the Cloud, some latency-sensitive services in fog nodes next to the networking stations, and finally, some last services at the edge to enhance responsiveness and reduce overall bandwidth requirements. Interestingly, each set of nodes might be owned by a different organization. Hence, each has different semantics. Therefore, the application (based on all these services) needs to tackle the usage of very different devices, which, on top, have different owners with, perhaps, different priorities when designing their systems.

[1]Over the course of this thesis, we use the terms Distributed Computing Continuum System (DCCS) and Computing Continuum (CC) system interchangeably, while choosing according to the context.

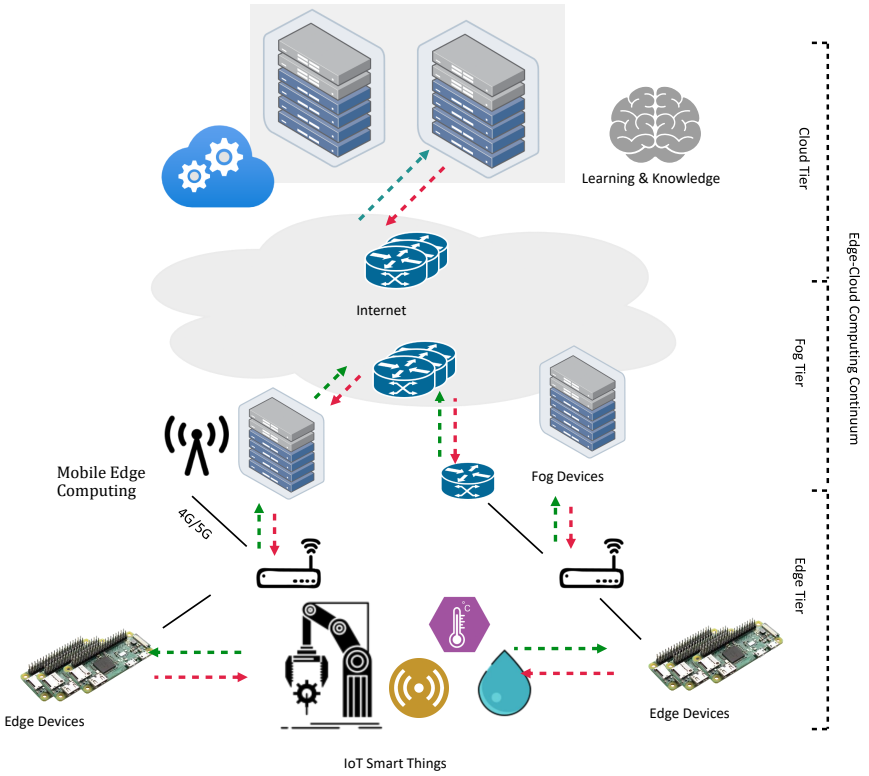


Figure 2.1: General Architecture for Distributed Computing Continuum Systems

To improve the interoperability between computational tiers in DCCS, Chapter 5 introduces mechanisms that (1) answer how actions at different tiers affect each other, (2) how lower-level tiers must be configured to ensure higher-level goals, and (3) how tasks can be offloaded between hierarchical devices according to device capacity and current load. Additionally, Chapter 6 provides a runnable DCCS prototype for a hierarchical solution that ensures processing SLOs throughout the multi-tier architecture.

Complexity of Governance

Currently, Internet-based systems are governed through the application logic and only residually at the infrastructure level by cloud orchestrators, which can basically run more copies of an existing job or schedule new jobs. Also, these are typically centralized entities, which clearly do not fit with the requirements for DCCS.

Another interesting aspect of current Internet-based systems is their usage of Service Level Objectives (SLOs) to set the minimal performance indicators for these systems. Unfortunately, current SLOs are only low-level metrics (such as CPU usage) or time-related metrics (such as end-to-end response time). Using SLOs for DCCS seems appropriate. However, we identify two key aspects that need to be improved:

1. they would need to be able to cover all aspects/components of the system so that the governance strategies are aligned regardless of what is being controlled.

2. their granularity is adequate to perform surgical interventions. Simply put, if the SLO is on end-to-end response time and it is violated, discovering which is the specific service/device/component/aspect that is producing the delay can be an overwhelming task, which cannot comply with time-constraint requirements.

To address that, this thesis provides mechanisms that allow fine-grained control over application behavior: in Chapter 3 we introduce complex behavioral SLOs composed of multiple factors, and multi-dimensional elasticity strategies that improve the reactivity of applications. This is further developed in Chapter 4, where we combine complex SLOs and multi-dimensional elasticity strategies in a video streaming prototype.

Data Synchronization

In DCCS, data is constantly generated, updated, moved, and accessed across a wide range of distributed devices, making it necessary to ensure consistency (through proper synchronization mechanisms [WZL+18]) across the continuum. Maintaining data integrity, coherence, and consistency gets increasingly difficult as data is processed and modified at different locations. Sometimes, end-to-end delays, network issues, and varying computational speeds (due to resource availability or constraints) can lead to inconsistencies or conflicts between data versions. Furthermore, data synchronization across hybrid setups involving diverse computational resources (cloud, edge, constrained IoT, or sensor nodes) presents additional challenges due to varying processing capabilities and connectivity limitations [RCVA22]. In DCCS, sophisticated synchronization mechanisms are required to ensure that all components can access up-to-date and accurate data.

This thesis targets this research gap in Chapter 3 by analyzing the computational overhead of transferring data between entities and the risks from centralized data storage, e.g., a vendor lock-in. Additionally, Chapter 6 discusses how to train device-specific ML models and exchange them between devices through transfer learning.

Sustainability and Energy efficiency

In terms of sustainability, there are *two* key aspects to consider:

1. the vast amount of computing devices and connections, and

2. their energy sources.

For the first consideration, computational infrastructure will keep growing during the next years. However, it is important to reuse existing infrastructure to limit the need to add new resources. Unfortunately, this challenges previous topics such as governance,

interoperability, and others, as dedicated resources are always easier to incorporate into a system than older ones with, perhaps, a different initial purpose.

The second sustainability consideration relates to the energy sources that are used in computing systems. AI-based systems require high amounts of energy; hence, being able to harvest this energy from renewable sources is of great interest. Unfortunately, solutions that can do that require also control over the energy grid, which is usually not the case. Energy efficiency relates to sustainability with the idea of using the minimum energy required for any job. This translates to choosing the right algorithm/service/device/platform for each case, which requires solving complex multi-variate optimization problems. Additionally, energy efficiency is key for energy-constraint devices, such as all those devices that are not permanently linked to the energy infrastructure.

This thesis puts particular focus on constrained Edge devices and how their energy consumption is affected by adjusting the runtime operation of a DCCS. For instance, the framework in Chapter 4 provides the energy consumption under different configurations, e.g., 10W when processing at the highest quality, and Section 5.3 uses this mechanism to offload computation from battery-powered devices that exceed their thresholds.

2.1.2 Promising Applications of DCCS

DCCS is capable of seamlessly integrating a wide variety of computing resources, allowing them to perform applications across various domains. In this section, we discuss a few applications (Industry automation, Transportation systems [WSM+22], Smart cities [CLH22, HMSS+22], and Healthcare [AOAL22]) with a use case example to show the difference between current technologies with computing continuum. By adopting DCCS features, these applications can benefit from improved resource utilization, faster decision-making, and other benefits, depending on the application requirements.

Industry Automation: Separating defective parts in manufacturing

Industrial applications encompass a wide range of sectors and industries where technology is utilized to enhance processes and operations. Common examples are: automated manufacturing, smart grids, food and beverage packaging and quality assurance, environmental monitoring, and process control (for example, oil refining or pharmaceuticals). Most of these applications are automated through machine technology, improving efficiency, productivity, and safety. When machines malfunction, efficiency and safety are reduced, while increasing maintenance costs. To increase efficiency and productivity, preventive measures are implemented, such as regular maintenance, real-time monitoring, and predictive maintenance. In this context, IoT is widely used in industrial applications to collect data from machines and send it to the cloud/edge for further analysis.

In manufacturing industries such as mobile assembling, food packing, or robot manufacturing, identifying low-quality parts is very time-consuming and tedious. There is a huge chance for manual errors to lead to overall quality control tasks becoming hectic. So, most manufacturing industries turn to automation and perform defect parts separation

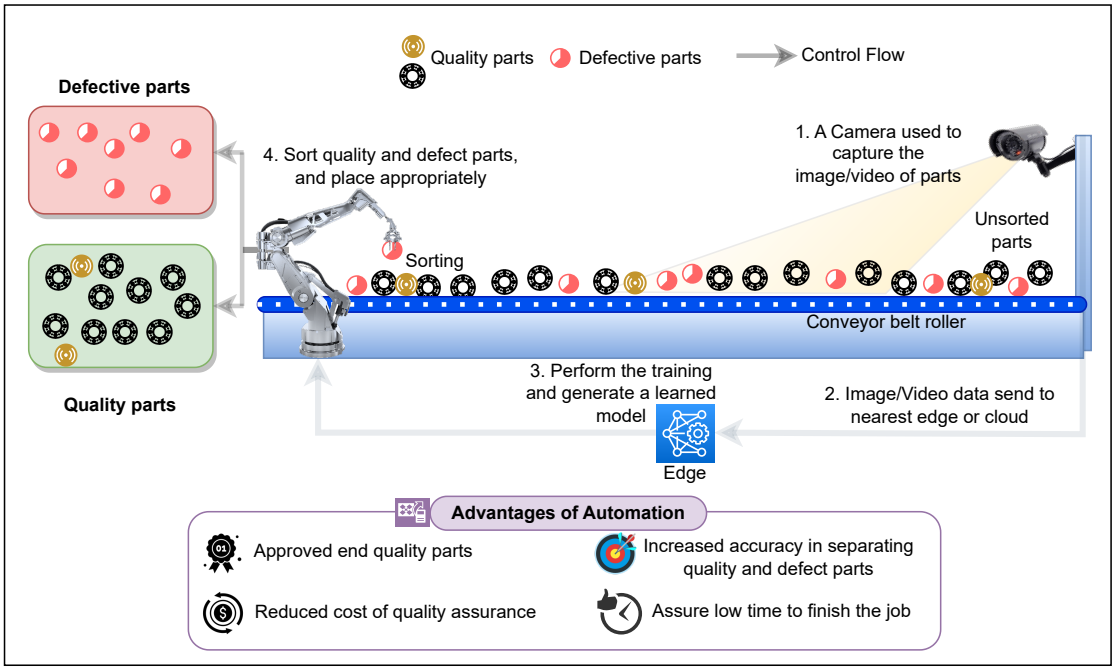


Figure 2.2: Separating qualitative and defective parts in an automated factory

using machinery. A basic quality and effective part separation system in manufacturing automation industries is discussed in Figure 2.2. We consider a rotating conveyor belt that moves unsorted parts (both quality and defective). In order to monitor all objects moving on the belt, a camera is installed to capture images/videos and send them to the nearest computing device. The robot uses this environment to sort out defective parts, e.g., by applying a DNN model for object detection, where the accuracy of the DNN model determines the overall performance and quality of the processing pipeline.

DCCS are capable of analyzing images or videos captured by cameras on conveyor belts in real-time. It distributes the processing load among edge devices and cloud resources according to resource availability. Through real-time analysis, parts can be classified immediately, reducing decision-making time and enhancing separation speed. DCCS facilitates seamless coordination and communication among connected devices. With DCCS, models are continuously refined, resulting in improved accuracy in identifying quality and defective parts. DCCS' distributed nature also contributes to fault tolerance. If one computing system fails, processing can be seamlessly shifted to alternative resources, minimizing downtime and disruptions in the manufacturing process. In spite of upgrading the entire automation system, DCCS can easily adapt and produce results according to upgraded requirements. In this context, DCCS delivers scalability without compromising performance to adapt to changing production requirements.

Transportation Systems: Ensure safety during driving

Transportation systems are organized networks that move people or goods between different locations, e.g., through roads, railways, airways, or waterways [RVM+23, ZLC+19, CLW+20]. DCCS plays a transformative role in modern transportation systems by integrating real-time data processing with the compute continuum. It processes data gathered through sensors equipped in vehicles and infrastructure to enhance traffic flow, reduce congestion, and ensure safe navigation [DGP+23, BW20]. Additionally, they support emergency response, infrastructure maintenance, and efficient mobility.

For example, consider a smart mobility scenario that monitors traffic and vehicle conditions, such as traffic flow, speeding, or crossing red lights [SFBS20]. For this to work, cameras could capture driver activity, radars would measure car speed, ground sensors would count cars, and a light system would allow recording in the dark. Such sensors must be connected to small processing units (e.g., IoT/Edge processors) to compute data, and the cameras will be near AI inference boards. As well as a large server on which to process video data captured by cameras, the application will have a large storage and processing capacity (i.e., cloud). Once results are analyzed and any abnormal events are observed, they will be immediately reported to drivers and traffic inspectors.

Through the seamless integration of edge computing, cloud resources, and real-time analytics, DCCS significantly enhances the efficiency of the above-described use case. DCCS allows for instant analysis of camera recordings, detection of driver activity via AI inference, and processing of diverse sensor data at the source (e.g., the edge) simultaneously with improved accuracy. It minimizes latency in detecting unsafe behaviors, alerts drivers instantly, and notifies traffic inspectors. DCCS also achieves low latency due to its parallel processing capabilities. Furthermore, DCCS can adjust sensitivity levels dynamically based on lighting conditions and traffic flow, enabling the system to adapt to changing conditions. With DCCS, we can distribute data-intensive tasks accurately and process and analyze them efficiently and timely. Ultimately, DCCS increases road safety by detecting safety violations rapidly and accurately.

Mobile Robots: Search and Rescue in Large-scale Disasters

Robots are autonomous machines equipped with sensors, actuators, and navigation systems to interact with environments. These robots have a broad range of applications across various industries, such as automated guided vehicles (AGVs), unmanned ground vehicles (UGVs), aerial drones, autonomous underwater vehicles (AUVs), search and rescue robots (SAR) and wearable mobile robots (WMR). Mobile robots continue to evolve, benefiting from advances in artificial intelligence, machine learning, and connectivity. Hence, they will also be part of the DCCS once several research challenges are solved [PD21]. Potential applications range from industrial automation to healthcare, agriculture, and beyond, where they enhance efficiency and reduce costs [WWD+23, KBHH23].

For example, consider that climate change has increased natural disasters and death rates – the first 72 hours following a natural or human disaster are crucial for locating

and rescuing those affected [MATM23]. Identifying victims in hostile environments is sometimes difficult for the rescue team [FSH+21], despite researchers and industry investigating advanced technological solutions for SAR operations. In SAR, mobile robotic units, such as drones and underwater systems, serve as vital frontline assets [MYG18]. As discussed earlier, these robots can detect victims and hazards in their surroundings by using sensors, cameras, and autonomous navigation capabilities.

With Edge AI, navigation and obstacle avoidance are enhanced due to rapid, informed decisions. By combining these technologies, rescue operations are more efficient, safe, and effective, reducing fatalities and mitigating their effects [LZHH23]. With DCCS, disaster response and rescue operations are enhanced by dynamically allocating computing resources, reducing latency, filtering and prioritizing data, ensuring fault tolerance, and adapting in real-time. Integrated data from IoT sensors and mobile robots streamlines decision-making, thus saving lives and mitigating the impact of natural disasters.

Smart cities: Efficient Waste bin management

A smart city uses a network of sensors and devices to collect real-time information about transportation, energy consumption, waste management, and public services [BMS20]. The respective data can be analyzed and used for decision-making as a means of increasing convenience, improving public services, and improving quality of life for citizens [Zha20, LCLW21, KVT21]. Since DCCS has inherent scalability, it can dynamically scale up or down in response to changes in the smart city ecosystem. The DCCS distributes processing tasks intelligently over this continuum to efficiently capture, analyze, and act on real-time data generated by IoT devices or provided by citizens.

For instance, in metropolitan cities across the world, municipal waste management has become a critical issue due to urbanization, growth, and lifestyle changes [MMD+19, PRD+20]. This issue affects many aspects of life beyond developing nations, including health, the environment, recycling efforts, and multiple industries. It is possible to solve this problem using currently available technologies by adopting smart waste management strategies. Using this strategy, stakeholders will be notified of the type and quantity of waste generated and how smart waste collection will be implemented [PRK+19, SJK+20]. In addition to improving fuel and time efficiency, intelligent route planning contributes to a more sustainable waste management system. There are several approaches available in the literature that use IoT and deep learning to improve smart waste management [SIIA21, WQQ+21]. However, in the currently growing population and cities, more efficient and quick solutions are needed, and DCCS can fill this gap.

The expansion of the city's proximity and population simultaneously increases the number of waste bins. This further generates more sensory data in greater proximity to the city. Since the cloud is located far away, the transmission delay and computational latency increase the delay in decision-making. DCCS integrates diverse technologies and resources to enable real-time data processing, analysis, and decision-making, providing a great advantage for faster decision-making. In addition to ensuring ongoing effectiveness, DCCS'

scalability guarantees its resilience in the face of urban growth without interrupting the current system. Since AI/ML reaches greater use in many applications, their advantages are also grasped through DCCS to predict before damage happens.

Healthcare: Remote Patient Monitoring

Healthcare encompasses a variety of medical services, technologies, and systems designed to prevent, diagnose, treat, and manage diseases and health conditions. Several medical devices have evolved over the last few years, from wearable sensors to high-end machines, all used to collect patient data and process it via smartphones (i.e., edge devices) or in the cloud [DPH+19, AQIR20]. Healthcare industries require accurate and quick analytical results from computing devices. It is sometimes necessary to analyze intensive tasks such as medical images (X-rays or CT scans) or genomic sequencing, but the result is expected to be available within a short period of time [MAGA+19]. In addition to optimizing healthcare efficiency, accessibility, and outcomes, DCCS ensures seamless data flows from point-of-care devices to the Edge-to-Cloud Continuum for further analytics.

For instance, consider that hospitals have limited caretakers, making it challenging to monitor each patient continuously [PMS+20, NGS+20]. Thus, a remote and automatic alert system is useful to monitor patients continuously and record their health status for further analysis [KDKA23]. Numerous wearable sensors and medical IoT devices gather real-time patient data, such as vital signs and health metrics (depending on the patient's diagnosis), and send it to a central server (private cloud) for processing [HMDC19]. In cloud-based computations, advanced algorithms and/or machine learning are used to analyze and compare the collected data with the normalized health metrics. It is also possible to visualize these metrics for quick assessment [RWC+20]. This approach allows healthcare professionals to monitor patients' conditions remotely, detect deviations, and make well-informed healthcare decisions. Some advanced analytics and pre-trained decision-making systems can also recommend prescriptions according to their assessment. As a result of this system, patients receive better care by facilitating continuous monitoring and early intervention based on real-time insight.

As the DCCS allows computation across the continuum, optimized processing, and resource allocation between point-of-care devices and edge-to-cloud, the DCCS can handle large computations in a limited time. The scalability of DCCS allows the system to support an increasing number of patients and data streams without affecting performance and existing patients. Using DCCS, healthcare providers and patients can communicate in real-time, and critical changes in health can be detected and notified immediately.

2.2 Service Level Objectives

Applications can be composed of a large variety of services, such as healthcare monitoring or smart city routing. Understanding SLOs as service requirements means that each service has to be associated with SLOs that guarantee its expected behavior. The

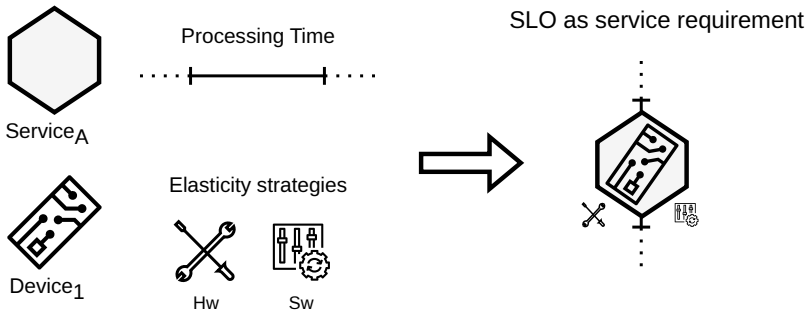


Figure 2.3: The SLO as a service requirement can be cast as a requirement (processing time) over service (A) hosted in device (1), which has elasticity strategies (Hw & Sw).

complexity of the service can demand more than one SLO; hence, it is fundamental that the SLOs properly reflect the service's requirements. Interestingly, this expected behavior of the service implies some knowledge of the type of device that will be hosting it. If we look into current Cloud-based SLOs, they do not need to consider the device in which it will be deployed to adjust the SLO. Actually, if the service requires a specific piece of hardware, e.g., a GPU, this constraint is sent to the scheduler. However, in this new CC paradigm, having a single scheduler (or resource manager) managing all hardware resources is impossible due to their geographical distance or the different ownership of the computing resources. Hence, adding this type of constraint to the service, e.g., a specific hardware need, gives the flexibility that, regardless of the final geographical location of the service, the local resource manager will be aware of the service requirements.

In addition, the continuum consists of a large variety of heterogeneous devices. Hence, expecting the same SLO compliance rate for the same service deployed in the Cloud as in a constraint edge device is unrealistic. In that regard, SLOs need to be aware of the possible deployment options for the service. Similarly, hosting devices can be grouped or classified with respect to their characteristics [PMN23], as has been done with services. Hence, in situations where the behavior of the service is known for a specific device type, it is possible to infer its expected behavior in other types. Figure 2.3 shows the different components required to build SLOs as service requirements for the CC. In the following subsections, each of the aspects shown in the figure will be explained.

Running example

To start, consider a simple machine learning pipeline with three services: data gathering and pre-processing, model training, and inference. Further, imagine that these services are part of an eHealth application able to predict if the monitored person will suffer from an adverse medical condition and, if so, trigger the needed alarms to provide medical assistance as fast as possible. Defining the requirements of the services needs proper dedication. Still, the classification of services, e.g., data gathering or inference, already identifies which requirements can be meaningful for the service.

2.2.1 SLOs definition

We will define an SLO as the probability (P) that its associated Service Level Indicator (SLI) is within a specified (i.e., desirable) range. In such a case, we will say that the SLO is fulfilled, or otherwise, violated. Mathematically, this is expressed as in Eq. (2.1):

$$SLO = P(x \leq sli \leq y) \mid x \leq y\,;\ \forall x, y \in SLI \tag{2.1}$$

Where x is the lower-bound and y is the upper-bound for the SLI, which is interpreted as the set of values that the metric can take. Latency, for example, assumes values in $\mathbb{R}^+$, because time will always be a positive real number. Hence, *sli* describes a specific value of the SLI for a given time. Notice that we have omitted any time reference in the previous equation, but given that *sli* is time-dependent while monitoring the SLO, we will obtain a time series. In general, SLO-based management of DCCS implies adapting the system to maximize the probability that the *sli* is within its range.

To define an SLO, we need to specify its operation range, "$y - x$". This means defining where we center the range and what its length is. For the following discussion we assume that the SLO range is normalized, so it is fair to compare ranges between different SLOs. Intuitively, the center of the range is the expected value for the SLI. Interestingly, this might vary depending on the device in which the service will be deployed. Simply put, the expected response time of a service performing a machine learning inference task in the Cloud will be lower than the same service in the Edge. The "length" of this range describes the criticality of the service. Services with large ranges can adapt easily to many situations and, consequently, are less critical. Conversely, a short range indicates that an SLO is heavily constrained and the service is critical. Hence, it is important to consider the SLO range when elasticity strategies need to be applied.

Running example

Hence, for each service requirement, an SLO must be defined. With the medical example, one could constrain the inference service processing time (T) between T_{min} and T_{max}. Of course, the lower bound for the processing time could be removed, but if there is knowledge about the service behavior, the lower bound can help identify faults or anomalies. When the underlying infrastructure is considered, i.e., the specific type of hardware that shall host the services, the SLOs definition might vary, and other SLOs might be relevant. For instance, considering that data collection and pre-processing services are hosted in an SBC at the user. This might need a constraint on the device's power consumption (e.g., hourly average consumption $< 8W$), which can be reflected in the device's availability. Also, if the inference service is in an Edge device for privacy enhancement and latency minimization, this can shift the SLO on processing time – creating T'_{min} and T'_{max}.

2.2.2 Types of SLOs

SLOs are linked to services and specify their expected behavior. However, we must remember that services are components of a larger application. In that regard, the

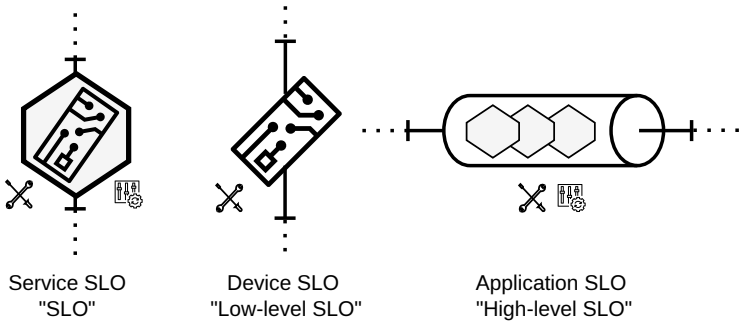


Figure 2.4: Types of SLO

application or significant parts of it might need specific requirements. Imagine a machine learning pipeline (e.g., a set of sequential services that gathers data, pre-process, trains a model, and broadcasts it to edge devices) having specific time-based requirements for each service and an overall requirement of achieving a test accuracy of at least 95%. In this case, the accuracy SLO can be part of any of these services, but it is really an SLO for the pipeline meta-service[2]. In parallel, the underlying infrastructure belongs to other stakeholders. Hence, it is probable that they need to set requirements for their devices to ensure their performance when providing a host for several tenants.

We can define three types of SLOs: infrastructure, service, and application (or meta-services). Figure 2.4 shows a graphical representation of each type of SLO. We will use the term high-level SLO for those related to the application or meta-services and low-level SLO for the ones that relate to the infrastructure components. Regarding the SLOs placement, services' SLOs will be located together with their service. For instance, both entities would share a pod in a Kubernetes-based application. Infrastructure SLOs would be deployed in services hosted in the critical infrastructure, i.e., devices that need to be specifically monitored to ensure their proper behavior. However, the placement of the application SLOs is not that clear. Actually, it brings a novel degree of freedom for system optimization. However, our intuition is that the system architecture will show the best candidate locations for these SLOs. Hence, one can assume that proximity to data and to elasticity strategies is valuable. For clarification, by proximity to the elasticity strategies, we mean that there are no significant delays between the service that requires the elasticity strategy and the one that can apply.

Running example

The eHealth pipeline can include high-level SLOs that consider the overall cost of the application, or the overall success of the alarm system. In contrast, lower-level SLOs are bond to the underlying infrastructure. Edge devices might have a GPU available for

[2]We use meta-service for services that could be grouped, e.g., a pipeline, and for services that provide functions beyond the application scope, e.g., an orchestrator.

inference, but with shared usage, their overall usage throughout the day is limited. Or the SBCs might run on battery power limiting its availability.

2.2.3 Tailored adaptations

One crucial aspect that motivates SLOs, considering both service and device to be deployed, is the capacity to define tailored elastic strategies for service-device pairs. An elasticity strategy or adaptation is a change in the service or device that ensures the fulfillment of the SLO when it has been violated (reactive behavior) or when it is about to be violated (proactive behavior) [DGST11]. In cloud-based applications, the adaptation capabilities are basically horizontal and vertical scaling; the Kubernetes[3] autoscaler is the reference for both scientific and production systems. Horizontal or vertical scaling is not always available at constrained devices; hence, tailoring elasticity strategies to a service-device pair is fundamental for autonomous and decentralized behavior.

When considering the service and the device, we are capable of knowing if the service will scale or if offloading to near and similar devices is an option [SPDD24b]. Further, some devices can modify their characteristics. As an example, most NVIDIA Jetson devices[4] allow runtime configuration of their maximum energy consumption, or they can enable/disable the GPU at runtime. Hence, these configuration options are cast as elasticity strategies. Further, we consider service-based elasticity measures, which go beyond adding or removing more replicas. Services configuration can be changed if they have the proper interfaces to adapt their behavior for the specific moment. It is important to clarify that these measures do not aim to change the service logic but to change its behavior. For instance, one can easily change the granularity of the input data to alleviate data processing tasks [SPDD23]. Similarly, the ML model for inference processes can be selected by trading off accuracy with energy consumption. Hence, services must provide interfaces to trade-off characteristics regarding quality, cost, or performance.

Running example

Executing the inference service at an Edge device with GPU allows the design of two elasticity strategies that go beyond scaling, and might be specific for this type of host. For instance, if the GPU can be utilized on demand, the service can run initially without, and when the required number of inferences increases, the GPU is switched on. Further, if the GPU usage limit is reached, the service can be offloaded (or the requests re-directed) to another Edge device with GPU time available for the application.

2.2.4 SLOs – Areas of Interest

Up to this point, we have been explaining that (1) SLOs have to define service requirements together with the hosting device, that (2) we have different types of SLOs, and that (3)

[3]Production-Grade Container Orchestration, Last accessed: April 30, 2025
[4]Overview of NVIDIA Jetson Modules, Last accessed: April 30, 2025

tailored adaptations for the service-device pair are needed to ensure SLO compliance. However, we have skipped a crucial aspect: decentralization.

Decentralization requires that each SLO knows its needs and capacities as any autonomous agent. We define the area of interest of an SLO as those variables and parameters that the SLO must consider to evaluate its current state autonomously and to take action accordingly. Hence, evaluation and adaptation are performed locally. Indeed, higher-level SLOs will require data coming from different parts of the system. Hence, their area of interest might be more extensive. Regardless, filtering out redundant or irrelevant data for the specific SLO is fundamental to achieving scalable system management.

Interestingly, why would someone need to check influencing variables or parameters to the SLO if directly observing the SLO is feasible? The answer is twofold; on one side, knowing the influencing variables provides the causes of SLO behavior, which leads to explainable and accountable systems. On the other side, the influencing variables and parameters are needed to take the most adequate elasticity strategy. Hence, providing an area of interest per SLO maximizes the level of decentralization for any SLO-based system. Local decisions are framed to an SLO. Therefore, its elastic strategies [ZTL+19] are cast as parameters that the SLO can consider to choose its available elasticity strategies. Hence, the possible adaptations for the SLO are known and available only locally.

Running example

Let's assume that there is at least 1 SLO defined per service, at least 1 higher-level SLO, and another lower-level one. At this point, besides monitoring SLO behavior, other metrics and parameters of the system must be tracked. This consists of metrics of the underlying infrastructure, from CPU/GPU usage to power consumption or requests per second; service metrics such as inference time or pre-processing queue; and finally, parameters that influence the service behavior, which might be used as elasticity strategies. While expert knowledge can used to identify candidate elements to track, in DCCS, ML technologies can help suggest relevant metrics or parameters.

2.3 Behavioral Markov Blankets

The Markov Blanket[5] (MB) is a mathematical concept that defines each SLO area of interest. This concept has two valuable perspectives. On one side, the Markov Blanket, defined by J. Pearl [Pea88b], is purely probabilistic. Conversely, the Markov Blanket used by K. Friston [FKH06] to define the Free Energy Principle has an ontological perspective, i.e., it is used to define what any thing is. The Markov Blanket of a random variable, x, (in the probabilistic sense of the meaning) contains all variables that make x conditionally independent of any other set of variables. In a Bayesian Network, the Markov Blanket of

[5]Formally, there is a difference between the Markov Blanket and the Markov Boundary. The latter is the minimal set of the first. However, to align with previous work and because this distinction is not critical for our work, we indiscriminately use the term Markov Blanket.

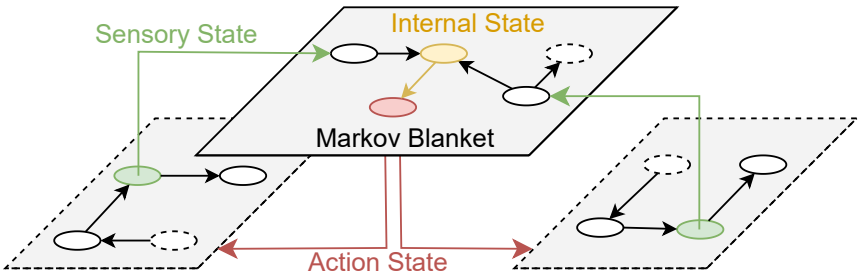


Figure 2.5: Hierarchical dependencies between different Markov blankets

a variable can be visually identified because it is always composed of its parents, children, and co-parents. Formally, if $MB(x)$ are the variables from the Markov Blanket of x and Y contains only variables outside the Markov Blanket, then Eq. (2.2) holds.

$$P(x|MB(x), Y) = P(x|MB(x)) \tag{2.2}$$

It is possible to bring this concept to the DCCS by assuming that this central variable, x, is the SLO at hand. Then, applying the Markov Blanket over this variable provides the set of variables that will affect the behavior of the SLO compliance. Hence, this sets a causality filter over the SLO, identifying only the system metrics that have to be tracked for the SLO, heavily reducing the time needed to assess the SLO status and inferring possible adaptation measures. Focusing on the set of variables contained in the Markov Blanket minimizes the monitoring effort while maximizing its effectiveness.

Discovering the Markov Blanket of an SLO is a complex task that requires the combination of two types of knowledge (1) there is a need for expert knowledge to identify the system variables that might be needed to assess the SLO. On the other hand, (2) it requires the system's data to use Markov Blanket discovery methods to quantify the relation of the selected variables with the SLO, such as presented by Fu et al. [FD08], the interested reader can check this survey for more detail and candidate methods [VCB21].

The second perspective of the Markov blanket uses the concept to build the interfaces of a thing (the SLO) with its environment. On the one side, it defines the *sensory states* – variables affected by the environment that influence the internal state. On the other side, the *active states* – variables affected by the internal state that influence the environment. The environment is composed of many things, such as different computing services, the users, the hardware in which services are hosted, etc. However, all of these entities feature their own Markov blanket, which defines the interface between them and how they react according to each others actions. Consider Figure 2.5, which shows how different Markov blankets use their sensory states to build their internal state, and how the central blanket uses its current state to decide for an action that affects the other blankets. This shows how hierarchical services could take actions based on dependent services' observations; in particular, acting if observations indicate that SLOs (internal state) are violated.

When defining the active states, it is only partially true that these states directly influence the environment. What they do is affect the relation of the SLO with the environment.

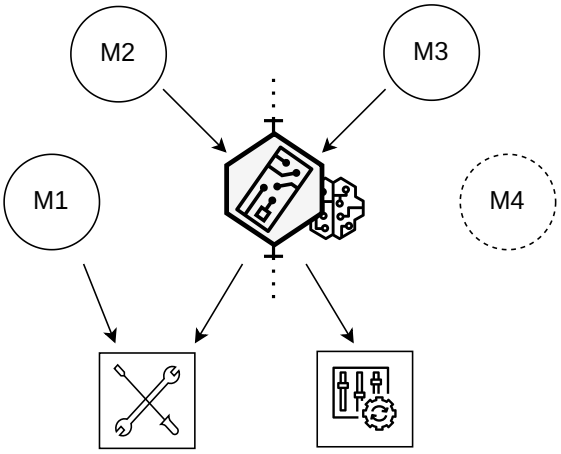


Figure 2.6: Markov Blanket representation of an SLO.

This means that if something in the environment is making the SLO deviate from its equilibrium, the SLO (meaning the autonomous agent controlling it) has to perform an elasticity strategy to revert that trend. This can be on the environment itself (external action), e.g., spawning a new service instance to absorb the high demand. Still, it can also change the service itself (internal action) [SCPDD23], e.g., reduce the granularity of the data being analyzed, adapting the QoS offered to keep the environment in equilibrium.

Figure 2.6 represents an SLO with its Markov Blanket: M1, M2, and M3 depict metrics influencing the SLO behavior, while M4 is a metric that does not influence the SLO. Additionally, the two squares at the bottom represent action states, which can influence the relation of the SLO with its environment. The next section will introduce the brain behind the SLO, i.e., how we provide autonomy to the SLO.

Running example

Now that the metrics and parameters available for each SLO are defined, it is necessary to keep only those directly affecting the SLO at hand. Computing the Markov Blanket of each SLO allows for determining which variables (i.e., metrics and parameters) the SLO is conditionally dependent on. This way, all variables that do not directly affect the SLO can be discarded, reducing the total amount of data to be analyzed. The requirement is modeled as a simplified Bayesian network, where the SLO is the central variable. Further, those parameters that might be used as elasticity strategies are linked with other system metrics, which will help identify the best strategy according to the system status. For instance, the inference processing time will have dependencies only with the GPU status and the model time. Hence, if the GPU status is already ON, the only way to reduce processing time is by using a cached model that requires less computational effort.

2.3.1 Autonomy

At this point, SLOs have all but one required ingredient to behave autonomously: intelligence. Simply put, the capacity of the SLO/service to autonomously decide how to adapt given its current state. Currently, software systems have three main directions to achieve intelligence: rule-based, model-based, or agent-based. Firstly, dealing with large, heterogeneous, and distributed systems precludes the usage of rule-based decisions as the space of possible situations is too large and complex for anyone to predetermine all rules. Interestingly, this is the standard approach for state-of-the-art Cloud systems (i.e., the Kubernetes autoscaler[6]). This works due to Cloud homogeneity and centralization.

However, research is already going beyond this when considering the Edge. For example, Toka et al. [TDFS21] develop AI-based models to manage Edge resources. Further, model-based requires previously specifying the model by using the underlying laws of physics of the system or its data to build the model. For instance, Liang et al. [LHAES23] build models for different Edge devices that perform machine learning tasks using queue theory. However, this results in developing and validating specific models for each service and device type combination. Model-based approaches using deep learning have also great success. However, the amount of data to train these models is huge; for instance, Jeong et al. [JBP+23] used 30 days of data for training, while the generalization capabilities on dynamic environments, such as in the CC, still need to be proven.

Lastly, agent-based systems can learn a behavioral model progressively while performing actions. As explained by J. Pearl [PM18], performing actions on the systems does incorporate information about the system's behavior that cannot only be observed. The most common agent-based intelligence is the one brought by reinforcement learning. Specifically, most of its applications for Edge systems are model-free [XZLH20, TW22], which means that the consequences of actions are not evaluated before the action is taken. Formally, the probability of the new state (s') given the current state (s) and the action taken (a) is not assessed ($P(s'|s,a)$). In any case, agent-based techniques usually require time to learn properly and suffer from the exploitation-exploration trade-off.

Hence, we opt for using the combination of both, agent-based intelligence with a system model. Later, in Chapter 6, we will focus on Active Inference (AIF) [PPF22] – an agent-based solution that we use for fulfilling SLOs. This promises the following benefits: First, agent-based solutions are well-suited for decentralized systems, where components have autonomous, self-adaptive agents. Second, the formulation of AIF is perfectly aligned with the Markov Blanket representation of the system. This enables a complete and more straightforward integration into the SLO-based model that we are proposing. Third, its objective function is not reward-based but aims to improve its model of the environment [SBPF21]. This subtle difference better suits systems that might need to change their requirements with time. Fourth, AIF allows injecting the expected observations into the model, i.e., fulfilling its SLO becomes the main driver for the agent actions. Hence, it can learn how their actions affect the SLO fulfillment.

[6]Kubernetes Horizontal Pod Autoscaling, Last accessed: April 30, 2025

However, as previously mentioned, the exploitation-exploration trade-off is always challenging for agent-based intelligent systems. In that regard, we can quantify the risk of new observation balanced against the information gain (e.g., model improvement) that it can provide, and depending on the criticality of the SLO (i.e., the length of its operation range), we can weigh the risk against the gain.

Running example *Now that each SLO is modeled as the central node of a Markov blanket, we can use AIF to improve the SLO behavioral model and find the best policy to keep SLOs fulfilled. Improving the model means closing the gap between the predicted outcomes and actual behavior when using elasticity strategies. Simply put, what one believes will happen if they turn on the GPU, against what really happens when doing it. In that regard, AIF can build policies that balance model learning (when the system behavior is stable) with the SLO stability optimization. Returning to the running example, the effect of GPU usage on the processing time is favorable. However, the model can be refined so that the effect can be quantitatively measured, and hence, when the GPU capacity is exceeded, an alternative elasticity strategy can be used.*

CHAPTER 3

From Metrics to Multi-Dimensional Elasticity

The growing amount of data generated at the edge of the network, e.g., by Internet of Things (IoT) devices, made it indispensable to relocate computational power close to the data source. Meanwhile, data tends to accumulate in chunks and is frequently subject to resource-intensive transformations, such as privacy enforcement. These phenomena, which are summed up as “data gravity" and “data friction", have an impact on data processing and the overall system. However, whereas cloud centers are able to dynamically adapt services, e.g., by provisioning additional resources, edge devices provide fewer options to react to changing workloads. To retain the option to process data locally, we present the idea of controlling data gravity and friction with Service Level Objectives (SLOs). We introduce Markov SLO Configurations (MSCs) as a novel approach to organizing performance metrics and elasticity strategies. MSCs, in conjunction with our presented architecture, enable the evaluation of SLOs, the context-based selection of elasticity strategy (i.e., corrective measures), and the execution of strategies directly on edge devices. Thus, we lay the foundation for a new generation of SLOs that can operate across multiple elasticity dimensions, e.g., by scaling quality of service (QoS).

The remainder of this chapter is structured as follows: Section 3.2 provides background knowledge on data gravity and data friction. It also contains an illustrative example of an SLO on data gravity and data friction. In Section 3.3 we identify metrics to capture friction and gravity, as well as elasticity strategies to counter them; within Section 3.4 we assess to what extent SLOs on gravity and friction are supported by the state of the art and what architectural extensions they demand. Related work, including the Polaris framework, is considered in Section 3.5. Finally, Section 3.6 summarizes the chapter.

3.1 Introduction

A Service Level Objective (SLO) is a commitment to maintaining a system in a desired state over a certain period of time [KL03]. It determines a system's status by evaluating one or more Service Level Indicators (SLIs), usually performance metrics, and compares the result against a benchmark. If they diverge, the SLO is violated, which is corrected by a chain of countermeasures (i.e., elasticity strategies [PMP+21a]). SLOs thus provide a system with "elasticity" – a degree of self-determination to adapt to changes in workload [HKR13]. Elasticity strategies can span multiple dimensions, for example, by adjusting the amount of resources provisioned or by scaling the quality of service (QoS) [DGST11]. However, to date, most SLOs are tied to a single elasticity strategy (e.g., scale resources in AWS EC2[1]) and are thus limited to one elasticity dimension [NMP+20]. This drastically limits the versatility of SLOs to react to more complex behavior within distributed systems, for example, compensating data gravity and data friction.

Data friction is a resistance that impedes data transfer between systems. It may be caused, for example, by incompatible data formats or privacy enforcement, resulting in processing delays, increased costs, and higher energy consumption [Edw10]. Data gravity refers to the tendency of data to accumulate and attract further data and applications; thereby, it becomes increasingly difficult and costly to move the data. To cope with this, there is a tendency to relocate processing facilities to the edge of the network, i.e., where data is created [Cam19, DPD23]. Processing data close to its source is motivated by numerous benefits, such as low latency and high bandwidth [SD16]. Friction-generating tasks (e.g., privacy enforcement) are equally assumed by edge or fog devices because, thus, unprotected data is less exposed to unauthorized access.

However, edge devices provide few options to scale provisioned resources [NPM+21]. To retain the option of processing data locally, edge devices must limit data gravity and data friction; otherwise, they find themselves unable to deal with growing data chunks and resource-intensive transformations. This inability to scale resources makes it attractive to explore other elasticity dimensions, e.g., by scaling the quality of generated data [FFACP18]. In this context, we present the idea of limiting data gravity and data friction by employing SLOs that extend into multiple elasticity dimensions. This chapter conceptually builds upon the state of the art for constructing SLOs, in this case, the Polaris framework [NMP+20], and addresses challenges that arise when attempting to control data gravity and friction with SLOs. Within this chapter we did not evaluate these concepts but laid the foundation for an upcoming implementation in future work. Our main contributions towards SLOs for data gravity and friction include:

1. A *mechanism to generate SLO configurations*, which uses Markov blankets (MB) for evaluating a system according to SLOs. This novel approach constructs around a central entity (e.g., data gravity) a graph of relevant metrics, elasticity strategies, and contextual information.

[1] `https://aws.amazon.com/ec2/`

2. The *Markov SLO Configuration (MSC)* as a method to organize the SLO lifecycle from the collection of metrics to the enforcement of elasticity strategies. Information contained by the MSC (i.e., which metrics to collect and which elasticity strategies to apply) can be administered within a distributed system to create hierarchical SLOs that extend into the computing continuum.

3. The *context-based planning of elasticity strategies*, which regards the edge environment to select an elasticity strategy. Thus, it becomes possible to compare elasticity strategies that operate in different elasticity dimensions and pick one, e.g., depending on the corrective impact.

Although the goal of this chapter was to limit data gravity and friction, the results are transferable for SLOs that pose similar requirements, e.g., planning multidimensional elasticity strategies or orchestrating strategies directly on edge devices.

3.2 Data Gravity and Data Friction

We consider data gravity and data friction to be fairly new concepts for most readers; therefore, we use this section to provide background information about these phenomena and explain in more detail how they impact data processing. To illustrate the need for SLOs based on data gravity and friction, we further present (1) an exemplary use case that is referenced within the remainder of the chapter and (2) an overview of motivating research challenges for creating SLOs that treat data gravity and friction.

Data Gravity

Data gravity describes the tendency of data to attract additional data. It is based on the idea that the more data and applications are stored at a particular location, the more attractive it becomes for other data to be stored there [Mac10]. This phenomenon usually draws data toward the cloud but could occur anywhere along the computing continuum, e.g., the network edge. It promotes the creation of central data storage, which provides numerous benefits [AW15]: fewer inconsistencies, data unification, and improved security due to fewer attack vectors. Services and applications are equally drawn to larger amounts of data, which is motivated by two essential benefits: low latency and high bandwidth [Mac10]. For measuring data gravity, the authors in [Dig22] provide a formula based upon four network metrics: (1) data mass, i.e., the size of data; (2) data activity, i.e., the number of movements and interactions with the data; (3) bandwidth; and (4) latency. The authors raised these metrics for thousands of enterprises to compare data gravity between different countries and regions.

Data Friction

Data friction is a resistance that impedes data transfer, for example, when exchanging data between institutions [Edw10]. It is frequently generated by preprocessing tasks,

such as data enrichment [XKK20] and privacy enforcement [FFACP18]. Data friction can be introduced by either socioeconomic or regulatory factors. Socioeconomic factors can be different understandings of data or metadata in scientific environments [EMB+11], or, more culturally, the simple desire to keep personal information confidential. Regulatory factors, on the other hand, are introduced to provide legal guidance, for example, the EU's General Data Protection Regulation (GDPR) [Bat17]. When transferring information, data friction demands additional resources, such as time, computational power, or personal effort. Consider, for example, privacy enforcement for a data stream: Transforming data according to privacy policies provides benefits to stakeholders [GWK+19], it is thus evident that the transformation cannot be omitted. Nevertheless, data friction can be optimized by employing more efficient techniques or dividing it between individual nodes. On the lines of data gravity, such a phenomenon must be measurable using a set of metrics, which will be explored further in Section 3.3.

3.2.1 Illustrative Scenario

To underline the benefits that emerge from SLOs on data gravity and friction, we present a motivating example. Although the scenario is tailored to smart health, the concepts introduced can be applied to any field that uses distributed edge architectures, such as industrial automation or smart cities. For now, imagine a scientific institution conducting medical experiments; therefore, they require medical data from patients who are located in hospitals or home care. Depending on the experiment, different data is required, for instance, internal values (e.g., pulse, blood pressure, etc.) or skin mutations. The first is provided as a numeric stream, and the second as an image stream. Data is provided by IoT devices that are equipped with sensors, which stream it to the institution, where data is accumulated until the experiment is evaluated and closed.

Patients agreed to participate if personal information was removed from the data; therefore, medical data must be transformed according to privacy policies before being stored centrally. To prevent unauthorized access to personal data, this transformation must occur directly on the IoT device. Technically, the smart health device could also provide additional data that is not part of the experiment, though this data is not authorized to leave the device. However, patients can decide to accumulate such data locally on the device, creating a personal data lake [AW15], and contribute this data to another experiment. This may be motivated by a monetary incentive.

However, IoT devices can be very restricted in terms of storage and computational power. Devices that fail to transform the data within a given time frame are facing "high data friction". The personal data lake on the IoT device further complicates the situation because increasing amounts of data become more difficult to manage. This can be summarized as "high data gravity." To deal with these issues, we introduce SLOs on data gravity and data friction. We organize IoT devices into clusters by grouping devices that perform similar tasks and have low latencies to each other. As far as possible, we add fog nodes (e.g., gateways and routers) to each cluster. Within each cluster, we elect

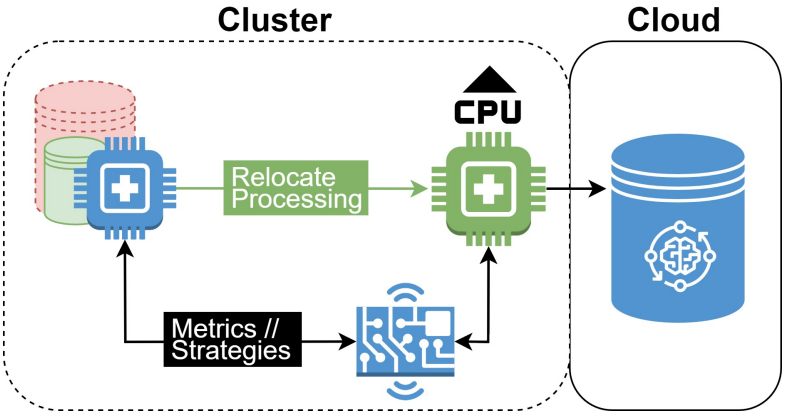


Figure 3.1: Decrease data gravity and friction by applying SLOs

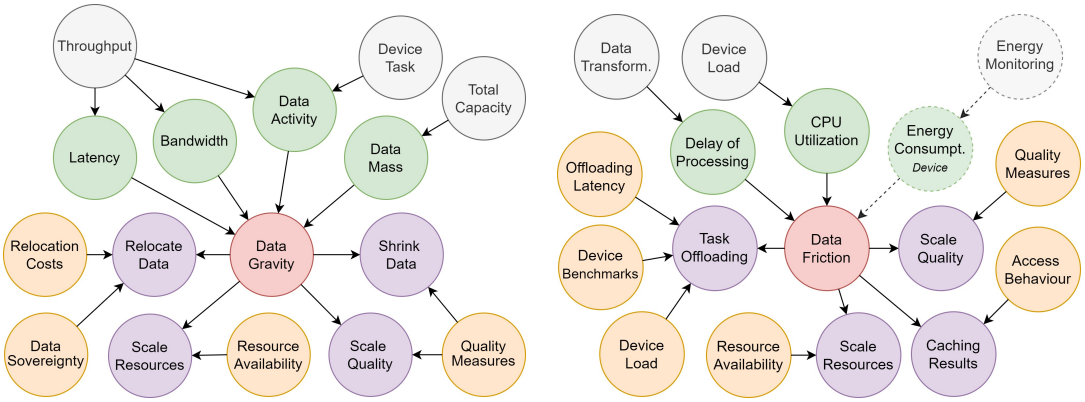
the most powerful device as the cluster leader [Mur22], which will be responsible for evaluating SLOs and orchestrating elasticity strategies.

The scenario is depicted in Fig. 3.1: Medical IoT devices are grouped into a cluster, which contains a fog device that assumes the role of the cluster leader. To evaluate the SLOs, IoT devices provide metrics to the cluster leader, which, once combined, reflect the high data friction and gravity. Due to this result, the cluster leader suggests the following elasticity strategies: (1) Transforming data is now performed nearby on a more powerful device, reducing data friction perceived by low-resource IoT devices; (2) scaling down the QoS (i.e., the data quality) reduces the data size and thus also data gravity within the personal data lake. The SLOs thus provided measures to detect and control high gravity and friction through multidimensional elasticity strategies. This improves the self-healing abilities of IoT devices [KC03].

3.3 Modeling Complex SLOs and Elasticity Strategies

Although mostly used in statistics and machine learning so far, we use Markov Blankets (MBs) [Pea88a], as known from Section 2.3, as a structured approach for exploring composed network metrics – such as data gravity and friction. Within an MB, we include the metrics required to determine the state of an SLO, elasticity strategies to correct a faulty system state, and contextual information for selecting between these strategies. For constructing an MB we assume a Directed Acyclic Graph (DAG), in which a central node x is connected to a set of incoming and outgoing nodes, i.e., its parents and children. Each parent node $p_1 ... p_n$ represents a random variable that influences the state of x at a certain time n; if for an arbitrary node y there exists no edge $y \rightarrow x$ in the graph, it means that y does not have an influence on the state of x at the time n.

To form an MB of data gravity, we (1) establish data gravity as central node x, and (2) arrange metrics as parent nodes $p1 ... p_4$. Each of these parent nodes has an edge $p \rightarrow x$ that expresses its direct influence on the data gravity. Child nodes of x are random



(a) Data Gravity (b) Data Friction

Figure 3.2: Constructing a Markov Blanket for (a) Data Gravity (b) Data Friction

variables that are influenced by the state of x; thus, we (3) introduce elasticity strategies to treat data gravity. To construct a Markov blanket for a node x, we must further include parents of its children; we thus (4) identify additional factors that influence the probability of performing elasticity strategies. The resulting DAG is shown in Fig. 3.2a, which is created by performing steps (1-4): Data gravity as our composed metric (red); its parent variables that directly influence the gravity (green); further factors related to random variables (gray); however, these are not necessarily part of the MB since their influence can be derived indirectly through the colored nodes; child variables that contain elasticity strategies (purple); factors that provide contextual information for these strategies (yellow).

By definition, the MB must contain all colored nodes from Fig. 3.2a; thus including network metrics, elasticity strategies, and factors that influence these strategies. We define such a selection as Markov SLO Configuration (MSC), a set of variables that provides sufficient information for evaluating an SLO and planning which elasticity strategies to apply based on the context. Equally, Fig. 3.2b contains the DAG for constructing an SLO for data friction, i.e., all information required to build an MSC. The graph follows the same color code as Fig. 3.2a; any set that includes at least all colored nodes meets the requirements of an MB.

While an MSC determines how an SLO is evaluated and executed at a time n, this configuration can change over time, including: (1) adding or removing metrics; (2) adding or removing factors influencing the elasticity strategies; or (3) adding or removing entire strategies. The MSC can thus develop and adapt over time.

3.4 From Metrics to Elasticity Strategies

Based on the information contained in an MSC, it becomes possible to evaluate an SLO and plan which elasticity strategies to apply. However, from a technical perspective,

there remain a variety of challenges to implementing these SLOs, including the following:

C1. SLOs were traditionally based on metrics that are generated in the cloud, for example, resource consumption [Bel16]. SLOs on data gravity or friction are based on metrics generated on the edge. To that extent, it requires the possibility of collecting metrics from edge devices, accumulating them close to the data source, and accessing them where the SLO is evaluated.

C2. Continuously collecting metrics produces a considerable amount of data. Transferring this information to the cloud for evaluation increases the overall network traffic. To that extent, it lacks an architecture that collects metrics and evaluates SLOs directly on the network edge. Evaluating SLOs close to where metrics are created would also foster timely reactions to SLO violations.

C3. Traditionally, only a single predefined elasticity strategy is applied to return the system to its desired state [NMP+20]. Contrarily, with the multitude of elasticity strategies contained in the MSC, it becomes possible to select the most suitable one. However, there exists no mechanism to compare elasticity strategies and select one of them.

C4. Elasticity strategies proved useful for cloud-based processing (e.g. provisioning virtual resources); however, elasticity strategies that scale the QoS must be enforced directly at the data source, i.e., at the network edge. To that extent, it requires an architecture that evaluates SLOs along the computing continuum, from where elasticity strategies can be orchestrated to edge devices.

The given challenges can be summarized by the MAPE+K (Monitor, Analyze, Plan, Execute, Knowledge) cycle [KC03], which the authors in [PMP+21b] used to capture all phases of an elastic cloud application. The cycle consists of (1) *monitoring* the system and collecting metrics, (2) *analyzing* whether the SLO is fulfilled based on this information, (3) in case the SLO was violated, *planning* which elasticity strategy to apply, and (4) *executing* the strategy to restore the system state. To pass information, but also to persist *knowledge*-based information, a state is shared between the stages.

In the remainder of the chapter, we move through the MAPE+K cycle and address the given challenges in their respective phases. We assume that the network is structured as presented in Section 3.2.1: edge devices and fog nodes are combined into clusters, each containing a powerful cluster leader. We use the Polaris runtime from Section 3.5 as a technical reference for specifying SLOs with *SLO Script* and evaluating and enforcing SLOs with their *SLO Controller*.

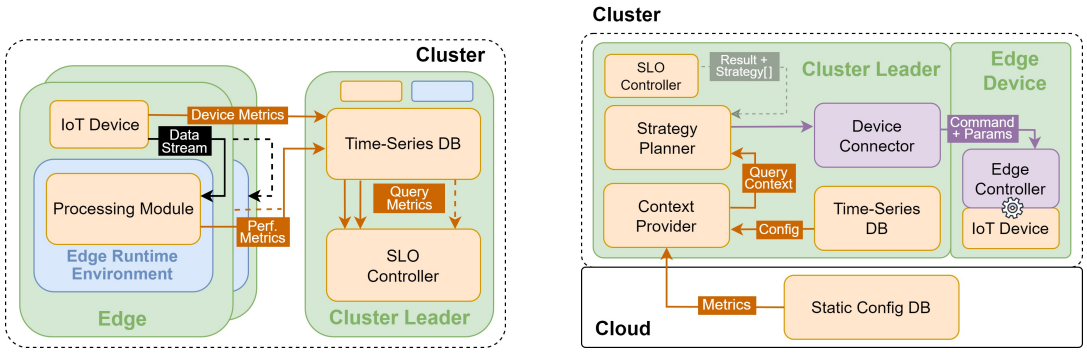


Figure 3.3: Architecture for multi-dimensional elasticity strategies: (a) Collecting and evaluating metrics from edge devices (b) Context-based planning of elasticity strategies

3.4.1 Metrics in the Computing Continuum

Collecting metrics requires a data store deployed close to the data source. To that extent, each cluster leader hosts (1) Prometheus[2], a time-series database (DB) used to store metrics, and (2) an instance of the *SLO Controller*, which will be required to evaluate the SLO. Unlike some edge devices, the cluster leader provides sufficient resources to run the *SLO Controller* and Prometheus, thus covering heterogeneity within the edge environment. The structure of the cluster is shown in Fig. 3.3a: Device metrics and operational metrics are generated on the edge devices and continuously ingested into the time-series DB on the cluster leader.

Following our approach, we collect metrics close to where they are created and provide them to the *SLO Controller* whenever the SLO is evaluated. We would thus solve challenge *C1*, by capturing metrics that represent data gravity and friction and accumulating them close to the edge device.

3.4.2 SLO Specification and Analysis

The second step in the MAPE+K control loop consists of analyzing the system state, i.e., evaluating the SLOs based on the provided metrics. As depicted in Fig. 3.3a, this is the responsibility of the *SLO Controller*: It first queries low-level metrics from a configurable data source, composes the SLOs of data friction and gravity, and then compares the result against a benchmark. The central entity in the *SLO Controller* would be the `DataGravitySLO` or `DataFrictionSLO` class, a direct representation of the SLO logic. It includes for each low-level metric a reference to a data source (i.e., Prometheus) and how it can be extracted.

Within the `DataFrictionSLO`, metrics are combined as in Eq. (3.1) and Eq. (3.2): *processingDelay* and *cpuLoad* are multiplied to accumulate their values. The *cpuLoad*

[2]https://prometheus.io/

maintains a neutral factor until it rises above a certain degree (t_x); only then does it have a decisive impact on the composed metric. While *processingDelay* and *cpuLoad* are a representation of the metric parameters, the target CPU load (t_x) is configured freely.

$$processingDelay_{ms} \times f\left(cpuLoad_{\%}\right) \tag{3.1}$$

$$f(x) = \begin{cases} (x/t_x)^2, & \text{if } x \geq t_x \\ 1, & \text{otherwise} \end{cases} \tag{3.2}$$

We use *SLO Script* to configure the SLO and link it to an elasticity strategy. Listing 3.1 shows how `DataFrictionSLO` is mapped to `SensorQualityScale`, the corresponding elasticity strategy. Whether the SLO is met or not, is specified through `frictionThreshold`; the desired CPU utilization over `targetDeviceLoad`. Although the configuration in Listing 3.1 statically maps a single elasticity strategy to the outcome of the SLO evaluation, we will break up this connection in the planning phase.

```
export default new DataFrictionSloMapping({
  metadata: ...
  spec: new DataFrictionSloMappingSpec({
    targetRef: ...
    elasticityStrategy:
      new SensorQualityScale(),
    sloConfig: {
      frictionThreshold: 50,
      targetCPULoad: 70} }) });
```

Listing 3.1: Configuring an SLO on data friction with *SLO Script*

Following the architectural consideration presented, we evaluate the SLO directly on the edge, close to where the metrics were created and stored; thus, solving challenge *C2*.

3.4.3 Context-aware Planning of Elasticity Strategies

Suppose that an SLO on data gravity or friction was violated, the third stage of the MAPE+K cycle consists of planning corrective measures. To select between multiple elasticity strategies, we introduce a component that receives the result of the SLO evaluation, queries contextual information, and identifies the most beneficial elasticity strategy. Depending on the scenario, “beneficial" could mean e.g., lowest energy consumption or highest corrective impact on the system.

The architectural extension is illustrated in Fig. 3.3b: Instead of mapping only one strategy to an SLO, an array of strategies can be supplied. The *Strategy Planner* resolves contextual information (e.g., costs for relocating data) through the *Context Provider*. Static configurations (e.g., quality measures) are queried from a separate DB, which can be hosted in the cloud. Since this information rarely changes, it can be cached in the *Context Provider* and updated by a trigger function.

Based on contextual information, the *Strategy Planner* compares the impact of elasticity strategies and selects one. Thus, we answer challenge *C3*. As a side note, the *Strategy Planner* could even indicate the top n strategies to accumulate their effects on the system. However, for now, we assume that either `SensorQualityScale` or `TaskOffloadKind` was planned to decrease data gravity or friction.

3.4.4 Distributed Execution of Elasticity Strategies

After planning an elasticity strategy, it must be orchestrated; therefore, two components were added to the architecture depicted in Fig. 3.3b: the *Device Connector* and the *Edge Controller*. The *Device Connector* is hosted on the cluster leader and is responsible for communicating elasticity strategies to edge devices, where they are received by the *Edge Controller*. In the planning phase, the *Strategy Planner* picked as elasticity strategy either `SensorQualityScale` or `TaskOffloadKind`, which are described below:

Quality Scaling adjusts the quality of the data that is produced on an edge device. Processing a stream of lower quality has been shown to decrease the computational load of devices [FFACP18, SMD22]; thus, reducing the data friction perceived by these devices. The managed IoT device must therefore support dynamic changes in the generated stream. Decreasing the quality of data reduces its size (i.e., mass); in cases where data is accumulated locally on the edge device, this decreases data gravity at the same time.

Task Offloading moves processing to a different device in the cluster, such as powerful fog nodes [PGPA+18]. Offloading processing is motivated primarily by minimizing the streaming delay, but it can consider other factors, such as lower energy consumption [CH18, GLL20].

Offloading load to more powerful or less used devices relieves individual devices of excess load, while decreasing latency [RMBG21]. This reduces overall data friction within the cluster because devices are less likely to be pushed beyond their operational limits. The two strategies presented can correct high gravity or high friction within a distributed system. For orchestration, they rely on the *SLO Controller*, which communicates the instructions to edge devices. Therefore, we declare challenge *C4* as solved.

3.4.5 Knowledge Transfer and Markov SLO Configurations

Components involved in SLO enforcement frequently need to share information with each other; for example, the result of the SLO evaluation must be communicated from the *SLO Controller* to the *Strategy Planner*. This type of information is transient and can be passed between components without persisting. However, if we consider the content of an MSC (i.e., metric composition and elasticity strategies), this information must be federated between cluster leaders and manageable somewhere along the computing continuum [DSCPD23a]. Whenever the MSC changes, e.g., by adding new types of

[3]The graphic is an extension of the master-worker pattern in [CGGN+18], our main modification is maintaining knowledge-based information (K) separately and providing it to the remaining steps.

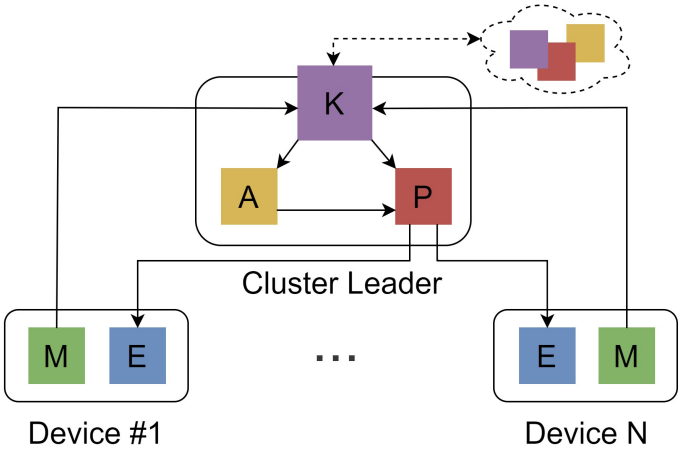


Figure 3.4: Distribute MAPE+K steps over computing continuum[3]

metrics or elasticity strategies, cluster leaders must be able to retrieve the latest version of the SLO configuration.

Fig. 3.4 visualizes how edge devices and the cluster leader exchange information depending on the MAPE+K stages: Edge devices monitor the status of the system (M) by ingesting metrics to intermediary storage. These metrics, along with other knowledge-based information (K), are accessible through an interface in the cluster leader. Whenever the system status is analyzed (A), metrics are queried to evaluate the SLO. Supposed the SLO was violated, contextual information is regarded to select an elasticity strategy (P). Eventually, elasticity strategies are orchestrated to edge devices for execution (E) to move the system back into its desired state.

The MSC can be administered by another entity, which hierarchically stands above the cluster leader, e.g., in the cloud. Thus, it becomes possible to erect multiple layers of SLOs that each react to changes in their respective environments.

3.5 Related Work

Although there exists work on data gravity and data friction that discusses their institutional and technical impact, as of our knowledge, none of them is related to their implementation as SLOs. Identifying metrics that reflect these forces can be compared to the work in [Bel16, LEB15], which discusses performance metrics for cloud computing environments. However, they focused solely on the computational load of the system to scale resources, thus, only operating in one elasticity dimension. This is similar to [ASLM13], where the authors discuss metrics for the elasticity of cloud databases. Complementarily, Fürst et al. [FFACP18] introduced a programming model that supports dynamic adaptations (comparable to elasticity strategies) within edge environments. Depending on the resource consumption of an IoT device, it was possible to dynamically

adjust the QoS; however, it lacked the options to consider other elasticity dimensions.

The Polaris SLO Cloud project[4] provides a runtime environment that enables the combination of custom SLOs and elasticity strategies. Their work in [PMP+21b, PMP+21a, NPM+21, NMP+20] provides fundamental concepts that are reused and extended over the course of this chapter: Within [NMP+20] they introduced next-level SLOs, that is, SLOs that compose multiple low-level metrics. Data gravity and data friction demand such measures because, as explained in Section 3.2, it requires a combination of metrics to capture such complex network behavior. The authors in [BBD+14] also support this thought; they state that "elastic behavior should be determined by a combination of factors", which is similar to next-level SLOs.

A central component for evaluating next-level SLOs is the *SLO Controller* presented in [PMP+21a] and [PMP+21b], which provides the following features: i) Create and update mechanisms for SLOs, configurations can thus change over time; ii) SLOs and elasticity strategies are loosely coupled, that is, SLOs and elasticity strategies can be replaced and reused in multiple SLO mappings; iii) SLOs are evaluated periodically according to a configurable interval; iv) metrics required for the SLO evaluation are queried through a service that relies on native DB controllers; and v) elasticity strategies are translated to orchestrator-native representations, which are submitted to the orchestrator through an integrated controller. For the specification and configuration of SLOs, the authors have provided a language called *SLO Script* [PMP+21a]. It is an extensible framework built on TypeScript, which provides type safety, that is, ensuring compatibility between SLOs and elasticity strategies at the time of configuration.

Existing work provided a framework for creating complex SLOs [NMP+20], measures for dynamically adapting services based on the system state [FFACP18], and identified the rising importance of treating data gravity and data friction [Cam19, EMB+11] at its source, i.e., the network edge. However, we can conclude from the presented related work that there exist no solutions that evaluate and resolve data friction or gravity within a distributed system, which we aim to address with the proposed SLOs.

3.6 Summary

This chapter presented the autonomous control of data gravity and data friction through SLOs. This was motivated by increasing data gravity, which requires relocating computational power to the network edge, and ubiquitous data friction, which demands additional resources when transferring data.

To construct SLOs based on data gravity and friction, we introduced Markov blankets as a novel approach to identify metrics, elasticity strategies, and contextual factors. Thus, our approach provides all information needed to evaluate the SLOs. Selecting a preferred elasticity strategy (i.e., one that operates in a certain elasticity dimension), considers the context of the edge environment. Evaluation of SLOs, planning of an elasticity strategy,

[4] `https://polaris-slo-cloud.github.io`

and orchestration of a strategy are assumed by a single powerful node. This responsibility can rotate within the distributed system, cluster leaders can maintain a state and recover SLO configurations in case of failure. Thus, we foster the creation of hierarchically organized SLOs that each react to changes in their respective environments.

For future work, we plan to provide a prototype that combines the presented components and features the entire MAPE+K lifecycle of an SLO. Certain aspects, such as the accumulation of multiple elasticity strategies, will further require a sophisticated orchestration model.

CHAPTER 4

Designing Reconfigurable Systems from Markov Blankets

Compute Continuum (CC) systems comprise a vast number of devices distributed over multiple computational tiers. Evaluating business requirements, i.e., Service Level Objectives (SLOs), would require collecting data from all those devices; if SLOs are violated, devices must be reconfigured to ensure correct operation. If done centrally, this dramatically increases the number of devices and variables that must be considered, while creating an enormous communication overhead. To address this, we (1) introduce a causality filter based on Markov blankets (MB) that limits the number of variables that each device must track, (2) evaluate SLOs decentralized on a device basis, thus increasing the granularity of the system state, and (3) infer optimal device configuration for fulfilling SLOs. We evaluated our methodology by analyzing video stream transformations and providing device configurations that ensure the Quality of Service (QoS). While average configurations violated between one and three SLOs during ten minutes of execution, the device would not violate any SLO while operating with our inferred configuration. The devices were thus able to perceive their environment and act accordingly – a form of decentralized intelligence.

The remainder of the chapter is structured as follows: Section 4.2 presents our methodology from the training of Bayesian networks to the extraction of knowledge; within Section 4.3 we present a case study used to evaluate our methodology; Section 4.4 provides an overview of existing research in this field. Finally, Section 4.5 summarizes the chapter and outlines future work directions.

4.1 Introduction

Computing Continuum (CC) systems as envisioned in [B+20, DPD23, T+22] are large-scale distributed systems composed of a wide variety of devices. Applications running in

the CC pose ambitious requirements, e.g., near real-time latency while dealing with huge volumes of data. Additionally, these requirements may change over time; to provide the best possible service, the CC system must be able to adapt accordingly. For example, a car tracking application can have night-specific requirements for precise tracking, while during the day the challenge is to deal with the total amount of cars. However, given the highly distributed nature of the CC, it is a challenging task to dynamically reconfigure all contained devices, while ensuring high-level system objectives.

While it was one benefit of Cloud Computing that requirements could be evaluated centrally, it was precisely the incapability of the Cloud to provide time-sensitive services close to consumers that drove the emergence of Edge computing. The CC, as a composition of these computing tiers, is now investigating new methodologies to manage such integrated systems, which need to be decentralized, given the system's scale and each tier's requirements. In this regard, we envision CC systems employing decentralized intelligence, which allows parts of the system to make decisions independently in favor of the application running on top. Starting with the smallest unit in the CC – a single edge device – the device would obtain the ability to evaluate its own state to ensure requirements are fulfilled.

It is important to clarify that intelligence, as described here, refers to the capacity to understand a situation and react in order to keep needs satisfied. One promising option to model this, is the behavioral concept introduced by Friston et al. [Fri13, KPP+18, PRP+20]. Essentially, it comprises sensory information and actions within a Markov blanket (MB) [Pea88a], through which a *thing* interacts with its environment. The MB itself shields the *thing* from all the variables it is conditionally independent of. Therefore, to determine the state of the *thing*, only the variables in the MB must be considered. Transferring this concept to the CC would allow modeling each device's behavior through MBs and evaluating device requirements by considering a limited amount of variables.

To model requirements, Cloud Computing introduced Service Level Objectives (SLOs) as a measure to achieve business agreements between infrastructure provider and application developer. However, we propose to expand SLOs to requirements that directly influence the system behavior and the application performance. Inspired by the work of Friston et al., continuing the research agenda set in [CPRD21, DPD23] and the ideas from [SCPDD23], we aim to leverage the behavioral concept of MBs to represent SLOs throughout the CC. The causality filter of the MB could reduce the scope of variables that each device must analyze; thus, decreasing the computational effort of analysis. This allows transferring intelligence from the Cloud to the Edge because it empowers resource-constrained devices along the Edge to evaluate SLOs themselves. Knowing the correlations between variable assignments and requirement fulfillment, the device's environment can be analyzed to infer configurations that best ensures SLOs.

In this chapter, we propose a methodology to evaluate application requirements through Markov-blanket-based SLOs. The method is able to constrain each SLO to a set of influencing metrics and infer the optimal device configuration given variable require-

ments. Further, the output is explainable due to the graphical model used. Hence, the contributions of this chapter are the following:

- A statistical reasoning model for analyzing conditional dependencies between metrics in distributed systems. Whenever requirements change, the model may thus itself answer which metrics are related to their fulfillment.
- The graphical representation of the device state as MB, which allows interpreting the device behavior. The state can be broken down into several SLOs; in case any of them is violated, it can be explained why.
- A mechanism to infer optimal configurations from MBs given mutable system requirements. It was evaluated under two scenarios in which our approach provided the only configuration that did not violate any SLO.

4.2 Bayesian Network Learning & Inference

With this section, we provide a novel degree of decentralized intelligence to devices in CC systems. Thus, edge devices may themselves ensure SLO fulfillment without relying on central control. From a high-level perspective, we plan to analyze the device state, map contained variables to the SLO fulfillment, and provide adaptive device configurations. Our three-step methodology to achieve this is visualized in Figure 4.1:

Edge devices produce metrics about ongoing processing; then Bayesian Network Learning (#1) is used to quantify correlations between these metrics and reflect the impact of environmental changes (e.g., increased incoming requests). Next, we introduce system requirements (i.e., SLOs) and extract a minimum subset of metrics for SLO fulfillment (#2). Ultimately, we use these MBs to estimate the probability of SLO violations and (#3) infer the device configuration with the highest compliance level.

While the proposed methodology describes a sequence of actions, the tools themselves (e.g., algorithms for structure learning) can be optimized depending on the data. In the following, the three steps of the methodology are explained in the respective subsections.

4.2.1 Bayesian Network Learning

Bayesian network learning (BNL) is an efficient way to generate an accurate structure for data; these approaches are mainly categorized into constraint-based (e.g., parent-child or grow-shrink) and score-based (e.g., Hill-Climb or genetic algorithm)[SSS19]. In this chapter, we consider the Hill-Climb Search (HCS) algorithm because of its rapid convergence, low complexity, and efficiency when considering limited attributes.

HSC starts with an empty graph; by adding or removing edges between variables, it creates a set of neighboring structures and selects the structure with the highest score. In this way, HCS repeats the adding and removing of edges until it reaches a maximum

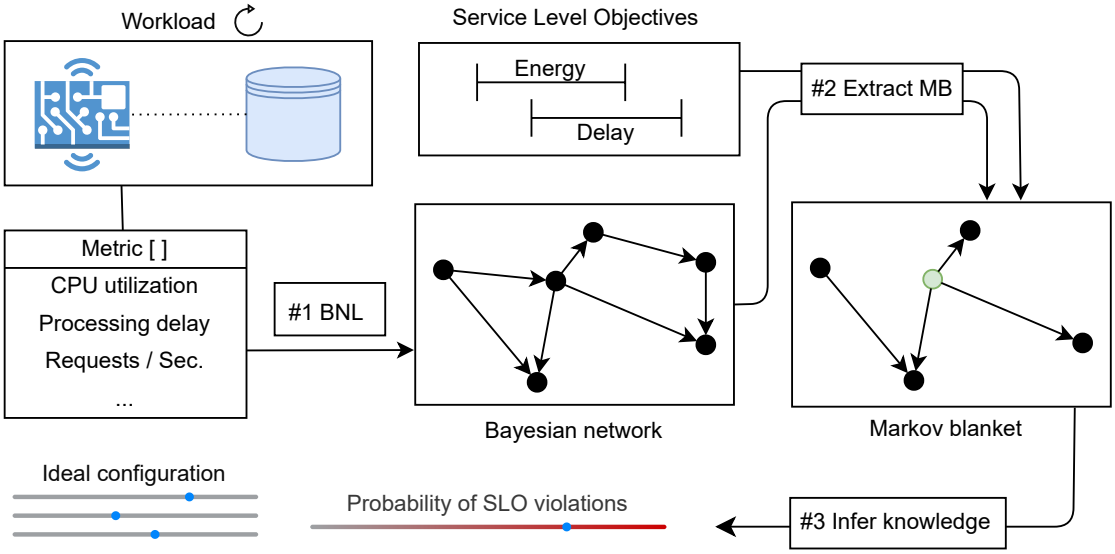


Figure 4.1: Training a Bayesian Network from processing metrics (#1); this is used to extract the minimum number of variables related to SLO fulfillment (#2) and a configuration that satisfies them (#3)

score, which describes the best DAG. Afterward, a conditional probability table for each attribute is constructed for the DAG through parameter learning using Maximum Likelihood Estimation (MLE), which is required for future inference and decision-making.

4.2.2 Markov Blanket Selection

A Bayesian network contains by design directed correlations and conditional dependencies of random variables; however, to determine the state of an individual node x, only a part of the network nodes are influential. This promotes the application of MB [AST+10, CPRD21, DPD23], which shield a variable x from all nodes that are conditionally independent of it. Specifically, the MB for x would contain (1) child nodes of x, (2) parent nodes of x, and (3) child nodes' remaining parents. Although this process appears simple for small Bayesian networks, larger networks require sophisticated algorithms for efficient MB extraction [TAS03, WLYW20].

Essentially, the MB is a causality filter to mask out variables that do not directly affect the node x; we use this to identify metrics related to SLO fulfillment. Suppose we specify an SLO (e.g., processing delay $< t$) and evaluate it using a single metric (e.g., *delay*), we might be interested in the factors that influence the evaluated metric to better fulfill the SLO. Namely, these are all variables contained in the MB of *delay*. In this context, we distinguish between metrics that statically reflect the system state (e.g., *request_count*), and those that represent a parameterizable variable (e.g., *active_cpu_cores*). However, we summarize both using the term "metrics" from a BNL perspective. While static

metrics are essential to explain why an SLO is in its current state, only parameterizable ones can be dynamically reconfigured. Overall, the sum of metrics in the MB provides a clear understanding of why an SLO is in its current state.

4.2.3 Knowledge Extraction

There exist two main categories of algorithms for extracting knowledge from Bayesian networks, namely Approximate Inference (AI) and Exact Inference (EI). Given a Bayesian network and system requirements specified as SLOs, we seek to extract probabilities of SLO violations for a certain device configuration, e.g., $P(delay > t)$ given $active\_cores = 4$. For dynamic reconfiguration, we require results to be (1) accurate, (2) converge reliably, and (3) fast for large networks. We argue that EI and, in particular, Variable Elimination (VE) [ZP94] as an instance, fulfill these constraints. In the following, VE is explained:

For a Bayesian network with a node set $\{v_1, v_2, v_3, v_4\} \in V$, VE accepts a list of target variables $T = \{v_1, v_2\}$, variable assignments $A = [(v_3 : a_3)]$, and an elimination order $O = \{v_4, v_3\}$. The query provides the conditional probabilities of T variables given assignments A. Each variable must either be eliminated or within the target set, thus $\forall v \in V, v \in T \oplus v \in O$. VE iterates over O and eliminates variables from V while updating the beliefs of remaining nodes; V thus eventually contains only T. In the given case, v_4 is eliminated first and v_3 second; the difference is the assignment of v_3, which introduces evidence in the form of $P(\{v_1, v_2\}|v_3 = a_3)$. While the elimination order has no functional consequence, it is relevant for the efficiency of VE and thus its scalability.

Given a parameter assignment $A = [(v_3 : a_3), (v_4 : a_4)]$ and a target variable $T = \{v_2\}$, we can construct an SLO that seeks to limit v_2 and obtain its conditional probabilities. We then introduce a threshold t and infer the probability of v_2 exceeding t given A. For instance, recall *active_cores* as c, *request_count* as r, *delay* as QoS metric d, and a minimum level of quality t. Suppose that we have a Bayesian network with $\{c, r, d\} \in V$, we can specify $T = \{d\}$, $A = [(c : 2), (r : 5)]$, any O, and $t = 8$. Running VE, we obtain the conditional probability of T given A, and from this we obtain the probability of $d > t$.

This will be our central mechanism for identifying the probabilities of SLO violations given a device configuration. If an SLO is violated due to an environmental change, e.g., higher *request_count* and thus exceeded *delay*, we can compare all possible configurations and provide the one with the highest probability of fulfilling the SLO, e.g., set $active\_cores = 4$. This matches our envisioned level of intelligence, i.e., "understanding a situation and reacting according to needs", and neatly fits the principles of elastic computing [DGST11].

4.3 Use Case: Video Processing

The following case study will be used to evaluate our methodology. In particular, we present two video streaming scenarios that require privacy-preserving transformations. We analyze device metrics to build a Bayesian Network, specify SLOs that characterize

Table 4.1: (Parameterizable) Metrics captured during workload execution

Name	Unit	Description	Param
delay	ms	processing time per frame	No
CPU	%	utilization of the CPU	No
memory	%	utilization of the system memory	No
pixel	num	number of pixel contained in a frame	Yes
fps	num	number of frames received per second	Yes
bitrate	num	number of pixels transferred per second	No
distance	px	relative distance of object between frames	No
transformed	T/F	if the model detected a pattern (i.e., face)	No
GPU	T/F	if the device employs a GPU	No
config	nominal	mode in which the device operates	Yes
consumption	W	energy pulled by the device	No

the QoS, extract the MB around each SLO, and finally, infer system configurations that have the lowest chance of violating SLOs.

4.3.1 Setup

Training a Bayesian network requires data; therefore, we use the framework introduced in [SMDD23], which allows edge devices to detect privacy-violating patterns (e.g., screen, face, or voice) in a stream and transform it continuously to resolve possible privacy violations. As a workload, it fits our methodology because it (1) provides an ample set of metrics reflecting the QoS of ongoing processing, (2) can be parameterized, and (3) can be executed on edge devices. Using the framework, we specify a privacy model that detects faces within a video stream and blurs the respective region, a scenario useful for office monitoring or AR setups [BBL+20, PK18]. Within this setup, a producer (e.g., IoT device) provides a video stream to the edge device, which transforms video frames according to the privacy model and then streams them to one or multiple consumers.

During execution, 11 metrics are captured, which we briefly introduce in Table 4.1 as they will be essential for defining SLOs. Each metric row contains a short description, the measurement unit, and if it can be parameterized. For example, *pixel* and *fps* are video stream properties; however, the producer can adapt them to create a variable *bitrate*. *Config* determines the device operation mode; devices such as Nvidia Jetson Xavier NX[1]thereby limit their maximum energy consumption and number of active CPU cores. In addition, the metric *distance* tracks the relative position of a detected face between frames, indicating how fluent/sluggish an object is tracked.

To explore correlations of parameterizable metrics, we simulate an adaptive bitrate; precisely, the producer periodically switches between different *fps* (12, 16, 20, 26, 30) and *pixel* (120p, 180p, 240p, 360p, 480p, 720p), while the edge device moves through available

[1]https://docs.nvidia.com/jetson/archives/r34.1/DeveloperGuide/text/SO/JetsonXavierNxSeries.html

config modes. Current parameter assignments are part of the set of metrics, which together are persisted with every processed frame. Except for *consumption*, metrics are directly observable by the device; *consumption*, on the other hand, is captured through an external power plug[2], which measures the energy consumption over a telemetry period of 10 seconds. Metrics are accumulated in a CSV file on the edge device, which will contain 189,000 metric rows for each evaluated device, captured within 2.5 hours.

We identified five SLOs that describe the system state in terms of QoS and Quality of Experience (QoE); however, each applicable scenario can have its own subset of relevant SLOs. We assign a name to each SLO and highlight the metrics from Table 4.1 (e.g., *bitrate*) that are used to evaluate the state of the SLO. Some SLOs are constructed by combining metrics (i.e., **within_time**), others are compared against a customizable threshold (e.g., **pixel_distance**), while other SLOs directly mimic the value (True/False) of the metric (i.e., **transform_success**).

network_usage Edge devices have limited network interfaces, and in some cases, limited network bandwidth. Since video streams are transferred over the network, *bitrate* is important to control network congestion.

energy_cons Edge devices are restricted in terms of resources and thus must economize or limit their energy *consumption* while ensuring compliance with the remaining system requirements (i.e., other SLOs).

within_time Video processing introduces a considerable streaming *delay*, which can lead to dropping frames and consequently poorer QoE. Hence, the stream's *fps* can be adjusted to limit/avoid dropping frames.

pixel_distance Measures the quality of the object tracking capacity; we expect the tracked object not to jump, but to have a smooth trajectory. Hence, we define a range for the acceptable *distance*.

transf_success Private or confidential information must not be disclosed; therefore, *transformed* should be maximized to increase utility of privacy transformation.

The workload was executed on three devices, which are listed in Table 4.2: Although all devices possess a GPU, only the Jetson Xavier NX supports the correct NVIDIA CUDA version to accelerate video processing. To explore correlations between the GPU and other metrics, we execute the entire workload twice on the Xavier NX, once with and once without CUDA acceleration.

4.3.2 Model Construction

For constructing the Bayesian network, we leverage *pgmpy*[3], a Python-based training framework. *Pgmpy* supports an ample set of algorithms for structure and parameter learning, including HCS and MLE (see Section 4.2.1). Training the Bayesian network

[2]https://www.delock.com/produkt/11827/merkmale.html, accessed June 13th 2023
[3]`https://pgmpy.org/`, accessed June 14th, 2023

Table 4.2: List of devices used for workload execution and training data generation

Name	CPU	RAM	GPU
Jetson Xavier NX	ARM Carmel v8.2 (6 core)	8 GB	NVIDIA Volta (383 core)
Jetson Nano	ARM Cortex A57 (4 core)	4 GB	NVIDIA Maxwell (128 core)
ThinkPad X1 Gen 10	Intel i7-1260P (16 core)	32 GB	Intel ADL GT2

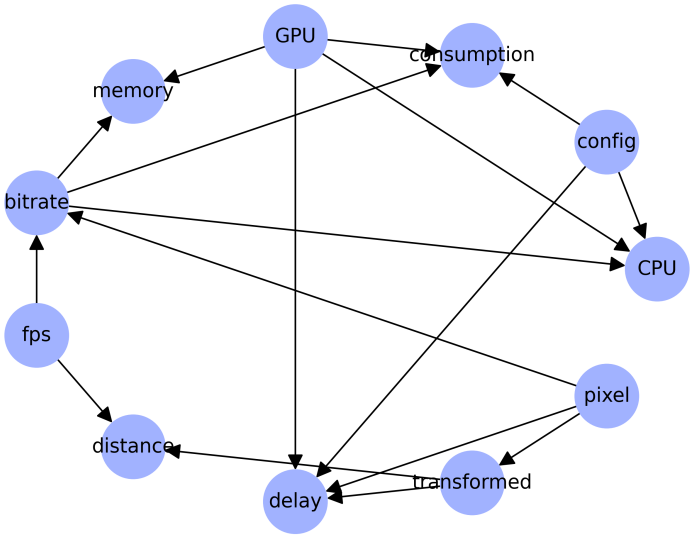
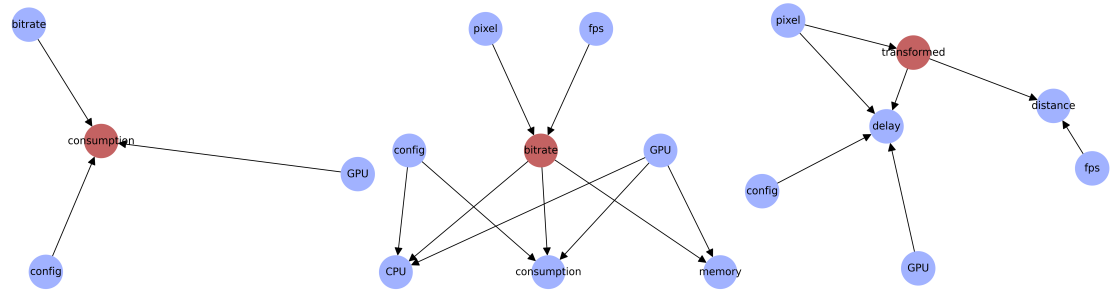


Figure 4.2: DAG after structure learning with Hill-Climb Search

with HCS and MLE on 756,000 rows with 11 columns of metrics takes approximately 30 seconds on a ThinkPad X1 Gen 10. The resulting DAG is presented in Figure 4.2: Nodes in the figure represent metrics, while edges between nodes indicate a causal relationship or at least a correlation.

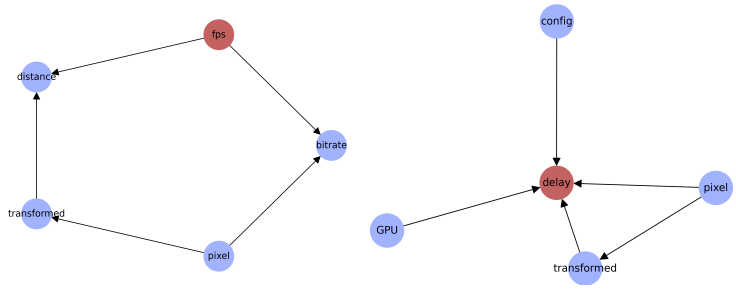
After training the Bayesian network, we extract for each SLO the MB around its central metric; the resulting MBs are visualized in Figures 4.3 & 4.4: Figure 4.3 shows three of the four simple SLOs, i.e., such that require exactly one metric for evaluation. For example, **energy_cons** must evaluate *consumption* to determine the state of the SLO; however, the 3 metrics surrounding *consumption* (i.e., *bitrate*, *config*, and *GPU*) could likewise be used to derive the probability that the SLO is violated. The fifth SLO, **within_time** in Figure 4.4, is composed of two metrics and thus features two MBs. Complex SLOs [NMP+20], i.e., such that consist of n metrics, produce n MBs; therefore, increasingly complex SLOs require mechanisms to merge and compress MBs.

We argue that the MBs extracted for each SLO are plausible because contained edges can be rationally explained. However, there is one particular case, which points toward a central issue – do edges represent causation or merely correlation. What is THE real cause, and what is a side effect, thus only increasing the complexity of the model?



(a) **energy_cons** (b) **network_usage** (c) **transf_success**

Figure 4.3: Markov blankets of the simple SLOs extracted from the Bayesian network



(a) *fps* (b) *delay*

Figure 4.4: Markov blankets of the **within_time** SLO combined from two metrics

Precisely, the processing delay increases linearly with resolution ($pixel \rightarrow delay$), and low resolution ($\leq$ 180p) negatively affects the transformation rate ($pixel \rightarrow transformed$). However, while transformation success (True / False) should not affect delay, the DAG still contains an edge $transformed \rightarrow delay$ in Figure 4.3(c). This is because low success (i.e., mostly encountered at low *pixel*) is correlated with low processing delay.

We make another observation: All MB SLOs contain at least one parameterizable metric within their sensory state, i.e., among the variables that influence the SLO outcome. From a requirements perspective, this is essential because it allows a device to adapt dynamically to fulfill the given SLOs.

4.3.3 Device Configuration Inference

We aim to infer device configurations with a high statistical probability of complying with SLOs; to achieve this, we use *pgmpy* to run VE queries (see Section 4.2.3) for the five SLOs defined. Instead of querying the entire Bayesian network, we execute the queries on the minimum subset of relevant variables, i.e., the MB of each SLO. Since the MBs of the SLOs contained all three parameterizable metrics (i.e., *fps*, *pixel*, and *config*), a device must include these parameters in an inferred configuration; otherwise, there is no full control over the SLOs. However, suppose we would only trace a subset of the SLOs (e.g., **network_usage** & **transf_success**), a configuration must only include the

respective parameters contained in the MBs, e.g., *fps* & *pixel*, but not *config*.

The presented version of VE computes the probability of SLO violations for exactly one parameter assignment; we repeatedly apply this approach for all possible assignments. To be precise, the parameter space for (*pixel* : *fps* : *config*) consists of (5 : 6 : 3) possible assignments. Iterating over $5 * 6 * 3 = 90$ possible assignments and 5 SLO-MBs produces $5 * 90 = 450$ inference queries, which require roughly 250ms on the Thinkpad X1 Gen 10, 500ms on a Jetson Xavier NX, and 650ms on a Jetson Nano. The result is a list of configurations that fulfill the given SLOs, e.g., one could be (240p : 20fps : 4C_20W). To deal with changing requirements and heterogeneous characteristics of CC devices, it is possible to provide additional constraints to the VE (e.g., *GPU*=False), or customize SLOs to rank a metric rather than limiting it (e.g. minimize *consumption*).

4.3.4 Evaluation

So far we trained a Bayesian network on processing metrics, from which we extracted MBs for each specified SLO. Afterward, we created a querying mechanism based on VE that infers device configurations depending on SLO thresholds. To evaluate the quality of inferred configurations, we will now compare the number of SLO violations between devices that apply inferred configurations and those that use arbitrary configurations.

We envision two scenarios with different system requirements that are based on the workload for face blurring. These requirements are reflected through specific SLO thresholds, which we will use to infer device configurations suitable for each scenario. The two scenarios are described below, while the corresponding SLO thresholds are presented in Table 4.3. We intend to minimize **energy_cons** for both scenarios regardless of whether the energy supply would be constrained.

- **Scenario A:** To create a virtual map (similar to Google Street View[4]), a camera-equipped car captures videos of streets. The car has an edge device installed (i.e., Jetson Xavier NX) to transform the stream, i.e., blur faces. The result is directly rendered to a local map and only accessed remotely in case of inspection, so **network_usage** is of less importance. We assume the rendering process to run in the background; therefore, the GPU is not available for processing. To create a detailed map, **pixel_distance** must be low, and **within_time** fulfilled in most cases. However, the stream can be re-rendered to blur undetected faces, thus **transf_success** is less critical.

- **Scenario B:** Within a smart factory, employees equipped with head-mounted cameras conduct an audit. For privacy protection, faces and displays are blurred before streaming the video to remote consumers. Video streams are transformed on edge devices (i.e., Jetson Xavier NX) that the factory employs. Video content is intended for live inspection only; therefore, **pixel_distance** and **within_time** are

[4] `https://www.google.com/streetview/`, accessed June 18th 2023

Table 4.3: SLO thresholds that reflect the scenarios' requirements

Scenario	**transf_success**	**distance**	**network_usage**	**within_time**	**energy_cons**	GPU
A	$\geq 90\%$	≤ 35	$\leq 8.2\ Mio.\ px/s$	$\geq 95\%$	$min(x)$	No
B	$\geq 98\%$	≤ 60	$\leq 1.6\ Mio.\ px/s$	$\geq 75\%$	$min(x)$	Yes
B*	$\geq 98\%$	(≤ 60)	—	—	—	Yes

Table 4.4: List of configurations generated by exact inference or picked naively

Scenario	Source	Resolution	FPS	Mode	GPU
A	inferred	240p	20	4C_15W	No
	naive	360p	30	6C_20W	
	random #1	120p	16	6C_20W	
	random #2	720p	12	2C_10W	
B	inferred	240p	16	2C_10W	Yes
	naive	180p	26	4C_15W	
	random #1	360p	20	2C_15W	
	random #2	480p	30	6C_20W	
B*	inferred #1	240p	26	—	Yes
	inferred #2	720p	—	—	

less important, while high **transf_success** prevents immediate privacy breaches. To support various providers and consumers, low **network_usage** is desired.

To create a condensed configuration, we adapt Scenario B to track only a subset of the SLOs – this is added as Scenario B*. Due to this, Scenario B* provides fewer SLO thresholds and configuration parameters because some parameters (i.e., *fps* & *config*) are no more correlated to the device state and may thus be removed from the configuration. To infer optimal device configurations, we now supply the SLO thresholds of the Scenarios (i.e., presented in Table 4.3) to the inference mechanism. The resulting configurations are presented in Table 4.4: The first line contains the inferred configuration, the second line a naive assumption; the third and fourth lines randomly generated configurations.

To evaluate the number of SLO violations, we configured the system according to Table 4.4 and measure the results in separated test runs. Performance was measured over a duration of 10min for each configuration, e.g., the inferred configuration for Scenario A would produce $10 * 60 * FPS = 12,000$ metric rows. The results are presented in Table 4.5: Over the measurement course, the inferred configurations fulfilled the SLOs for both scenarios. The naive assumption, on the other hand, violated one SLO within each scenario (red cell), i.e., in Scenario A it failed to fulfill **within_time**, while in Scenario B **transf_success** was violated. The randomly generated configurations committed two SLO violations in Scenario A and one in Scenario B each. Further, we calculated the

Table 4.5: Fulfillment of SLOs depending on scenario and configuration

Scenario	Source	**transf_success**	**distance**[5]	**network_usage**	**within_time**	**energy_cons**
A	inferred	98%	15 (97%)	2.0 Mio.	100%	6.0W
	naive	100%	10 (100%)	6.9 Mio.	92%	8.0W
	random #1	4%	127 (2%)	0.4 Mio.	100%	7.0W
	random #2	100%	28 (89%)	11 Mio.	100%	6.0W
	average	81%	73 (83%)	6.0 Mio.	81%	7.1W
B	inferred	98%	18(98%)	1.6 Mio.	100%	6.0W
	naive	92%	11(99 %)	1.5 Mio.	100%	6.5W
	random #1	99%	15 (100%)	4.6 Mio.	100%	6.0W
	random #2	100%	10 (100%)	12.3 Mio.	97%	7.5W
	average	81%	73 (86%)	6.0 Mio.	91%	6.7W
B*	inferred #1	98%	12 (100%)	—	—	—
	inferred #2	100%	—	—	—	—

number of SLO violations over all possible configurations (i.e., 90 combinations), which was on average even higher than those committed by the random configurations.

The results showed that our inferred configurations fulfilled all given SLOs while also consuming the least energy. The fact that Scenario B* did not violate any SLO also showed that it is possible to assemble SLOs or pick a share of them without violating any of the remaining SLOs. Thus, the number of variables and the time required to infer device configuration can gradually be decreased.

4.4 Related Work

Adapting a system to changing environments is itself not novel. Elastic Cloud Computing [DGST11], as an example, scales system properties (e.g., resources) depending on environmental variables (e.g., request count). This concept was transferred to the Edge [FFACP18, NMP+20], where it is necessary to scale other elasticity dimensions, e.g., the Quality of Service (QoS). Overall, this is a feasible approach if mapping single variables to corrective measures, but seems still very restrictive compared to the human mind, which uses its senses to select the best possible action. To combine multiple sensory inputs with corrective actions, the authors in [SCPDD23] introduced MBs as a behavioral model for edge devices. However, they assumed prior knowledge of how metrics are related to the system state; this approach is not scalable if requirements would change during operation and especially not if metric correlations are unknown at design time. Thus, existing approaches fail to ensure the intricate requirements of CC systems.

Analyzing variable relations ourselves to provide evidence about the system state is what differentiates our vision from existing work. The resulting MB – essentially a Bayesian network – can predict if environmental changes would violate SLOs and how the system must be reconfigured to fulfill them. In this context, there exists numerous work [AST+10,

[5] **distance** was implemented so that 95% of the measurements must lie above the threshold

BJ12, CGK+02] that addresses how Bayesian networks can be trained to extract MBs. In particular, Aliferis et al. discussed how to compose causal structures around target variables to form MBs. The extraction of MBs itself has inspired [TAS03, WLYW20] to provide efficient mechanisms for increasingly large Bayesian networks. However, to the best of our knowledge, there exist no attempts to embed MBs into the CC, which would allow devices to reflect on their own sensory state and reconfigure autonomously.

Taking this into account, we envision an unprecedented degree of decentralized intelligence to handle changing requirements; namely, identifying variables that give evidence about the system state, and automatically inferring a device configuration that fulfills SLOs. This is unlike [NMP+20, SCPDD23], where it was the operator's responsibility to specify these relations. The existing work on Bayesian networks [AST+10, BJ12, CGK+02] will be part of this intelligent behavior and help to make the state of CC systems more explainable. We thereby aim to solve core issues of the research timeline set in [CPRD21, DPD23] for accelerating the development of CC systems.

4.5 Summary

This chapter proposed a methodology for analyzing the system state according to a set of requirements and providing a configuration that meets these constraints. In particular, we created a statistical reasoning model for explaining the correlations between workload metrics and the system state, which is reflected by a set of SLOs and their MBs (i.e., the metrics most influential to the SLO). Given the MBs, we evaluated our methodology under two scenarios, for which we inferred the device configurations with the highest statistical probability of fulfilling SLOs, e.g., 98%, 97%, and 100% for Scenario A. We compared the frequency of SLO violations between these configurations and a baseline, i.e., naive assumptions, random configurations, and average violations over all possible configurations. The configurations inferred from the MBs did not violate any SLOs, while all baselines violated between one and three SLOs. The quality of the baseline and the scenarios themselves clearly influenced this observation; however, it provides confidence in continuing to develop our methodology.

Meanwhile, to maintain high-level system requirements fulfilled, CC systems, which are composed of multiple device tiers, must ensure that each tier's requirements are equally fulfilled. However, the heterogeneity of devices, the manifold of requirements for each device type, and the continuous changes in the environment make it impossible to centrally evaluate requirements. In this context, we envision transferring the concept of SLOs from the Cloud to the CC landscape and advocate our methodology to provide a modular explainable structure: instead of evaluating device health centrally, we use the concept of MBs to create cellular structures (i.e., modules) which are each characterized by a set of SLOs. For each module, it becomes thus possible to provide an ideal configuration that is most probable to fulfill the respective SLOs, while minimizing energy consumption. Essentially, this provides CC systems with decentralized intelligence, which will be able to cope with its scale and intricate requirements.

Our methodology was able to provide configurations that would not commit SLO violations; however, the scale of CC systems makes it necessary to assess the impacts of increasingly large Bayesian networks in terms of performance and precision. In this context, we selected Bayesian network training algorithms from a theoretical point of view, i.e., because literature advised so. However, it remains to compare the graphs generated by different training algorithms (e.g., explainability and modularity) and the quality of inferred configurations. Furthermore, to cover heterogeneity among CC devices, we aim to infer configurations for arbitrary devices.

CHAPTER 5

Orchestration of Computing Continuum Services

The governance and orchestration of DCCS represent a major open challenge, as discussed in Section 2.1.1. In that context, three of the most common problems [DMCP+23] are: (1) deployment of services or applications in the DCCS – finding the right host; (2) configuration of services and components – inferring SLOs that guarantee user satisfaction; and (3) recovering service quality – dynamically offloading computational tasks between devices. This chapter is a collection of three subchapters [1] to tackle these problems:

– in Subchapter 5.1 we analyze the interactions between microservices deployed over a CC infrastructure. This allows modeling how actions of one service, i.e., changing the service configuration, affect the remaining microservice pipeline. We propose a model for estimating the SLO fulfillment for a set of heterogeneous processing resources and, as a result, find the optimal service deployment for each microservice in the pipeline.

– in Subchapter 5.2 we move to more complex microservice architecture and focus on the precise implications of fulfilling high-level SLOs. For instance, if an end user wants fast response time for a user-facing service, what are the SLOs that backend services (e.g., processing, storage, etc) must ensure. As a result, we constrain all parts of a distributed processing architecture with lower-level SLOs and parameter assignments.

– in Subchapter 5.3 we focus on the resource limitations of heterogeneous edge devices and answer how overall SLO fulfillment can be improved by moving computation between devices. As a result, a platoon of autonomous vehicles could offload perception tasks (e.g., mobile mapping through Lidar) according to their current availabilities.

Finally, in Subchapter 5.4 we summarize the implications of these three contributions and discuss what challenges in the orchestration of DCCS applications remain open.

[1] We use the term "subchapter" for a stronger separation of the contents in this chapter

5.1 Markov Blanket Composition of SLOs

Smart environments use composable microservices pipelines to process Internet of Things (IoT) data, where each service is dependent on the outcome of its predecessor. To ensure Quality of Service (QoS), individual services must fulfill Service Level Objectives (SLOs); however, SLO fulfillment is dependent on resources (e.g., processing or storage), which are scarcely available within the Edge. Hence, when distributing services over heterogeneous devices, this raises the question of where to deploy each service to best fulfill both its own SLOs as well as those imposed by dependent services. In this subchapter, we maximize SLO fulfillment of a pipeline-based application by analyzing these dependencies. To achieve this, services and hosting devices alike are extended with a Markov blanket (MB) – a probabilistic view into their internal processes – which are composed into one overarching model. Given a mutable set of services, hosts, and SLOs, the composed MB allows inferring the optimal assignment between services and edge devices. We evaluated our method for a smart city scenario, which assigned pipelined services (e.g., video processing) under constraints from subsequent services (e.g., consumer latency). The results showed how our method can support infrastructure providers by optimizing SLO fulfillment for arbitrary devices currently available.

The remainder of this subchapter (see footnote 1) is structured as follows: Section 5.1.2 introduces background knowledge and an illustrative scenario, and Section 5.1.3 presents our methodology including MBC, which is implemented and evaluated in Section 5.1.4. Section 5.1.5 provides an overview of existing work in this field; finally, Section 5.1.6, concludes this subchapter with a future scope.

5.1.1 Introduction

Public spaces are increasingly covered with sensor networks, e.g., road surveillance or parking sensors, which are used to build compound services such as offered by smart cities [RRS+22] or smart homes. In particular, this can include microservice pipelines, where one service (e.g., road analysis) feeds its results to multiple other services (e.g., traffic routing). While data collection is often carried out through Internet of Things (IoT) devices, the tendency is to locate data processing services at nearby edge devices; this promises low latency and improved privacy. However, whereas the Cloud counted on vast amounts of virtualized scalable resources [HKR13], the limited amount of Edge resources promoted the rise of the Computing Continuum (CC) [DPD23, NRRC24] – a coherent integration of multiple computational tiers, starting from the IoT, over Edge and Fog, up to the Cloud. Thus, from data provisioning, over processing, up to consumption, services can be allocated and scaled over this large-scale multi-tenant distributed system [DMCP+23].

To ensure high-level requirements, such as availability or response time, the Cloud allows clients to specify Service Level Objectives (SLOs); by evaluating Service Level Identifiers (SLIs), e.g., system metrics, it is decidable if SLOs are fulfilled. To trace a system's behavior at finer granularity, high-level SLOs are diffused into smaller chunks [CPMM+23,

SPDD24b]; for compound services (e.g., pipelines), this boils down to requirements that individual services must fulfill. Within a pipeline, each service poses its own SLOs: a processing service, for example, might aim for efficiency, whereas a consumer service could aim for high video resolution, i.e., instances of Quality of Service (QoS) and Quality of Experience (QoE). Services, however, depend on the quality provided by predecessors and expected by successors; this constrains the actions of individual services because they must consider how local changes influence dependent services. Furthermore, SLO fulfillment is affected by hosting infrastructure [SCPDD23], i.e., low-resource devices (e.g., Edge) might not be able to fulfill performance constraints to the same extent as high-resource devices (e.g., Fog/Cloud). Hence, service deployment has strong implications not only for its own SLO fulfillment [Cao23] but also for dependent services'.

To maximize SLO fulfillment throughout a pipeline, the infrastructure provider must know where to deploy each service – we call this an "assignment" between services and hosts. Estimating the quality of an assignment requires (1) an understanding of how dependent services constrain each other's requirements and (2) the implications for the service and the deployed host, i.e., hardware utilization and SLO fulfillment. However, existing works [ZZL23, XDTZ20] do not consider transitive dependencies (i.e., imposed by dependent services) nor estimate resource utilization per service. Without the latter, deploying multiple services on one device (i.e., multi-tenancy) is risky because it is uncertain whether multiple services can coexist with their respective resources [ZMC+22]. Brute-force comparing all possible assignments empirically cannot be the solution: first, the underlying optimization problem is NP-hard [SPJC18], but secondly, even though you find the optimal assignment for one pipeline, the insight is not transferable to other scenarios. Services and hosts change over time due to demand and availability; hence, any solution should be able to repeatedly infer optimal assignments for changing setups.

In this subchapter, we present a 4-step methodology that finds the optimal service assignment by analyzing intersections between dependent services and their impact on hosting devices. In the first step, services and devices are extended with a Markov blanket (MB) – a probabilistic representation of their internal processes [Pea88a]. This allows predicting how changes to one variable (e.g., video resolution) change the conditional probabilities of another variable (e.g., throughput) [SPDD23]. To consider external factors, i.e., SLOs of dependent services, we propose the Markov Blanket Composition (MBC): a MBC comprises an entire pipeline and different processing hardware (i.e., spread over the CC) in one coherent and modular structure.

Following our vision laid out in [DPD23, PSDD24], we use MBC to model hierarchical dependencies between services, in particular, how their actions and perceptions influence each other. Given this MBC, we estimated how assigning one service to a particular device will impact its SLO and those of dependent services; by doing this repeatedly, the assignment with maximum SLO fulfillment can be inferred. Hence, the contributions of this subchapter are:

1. The MBC as a mechanism for finding dependencies between services and their

implications to processing hardware. Thus, services can incorporate and consider SLO fulfillment of dependent services and their hosts.

2. The generalization of services' resource utilization over heterogeneous hardware. This extrapolates the resource usage of a service at a particular device type and estimates the implications for comparable services or devices.

3. A collective inference mechanism that maximizes SLO fulfillment for a service pipeline by assigning services to CC devices. This considers dependencies in terms of QoS and QoE that services pose on their direct neighbors.

5.1.2 Preliminaries

This section provides an overview of concepts and definitions used throughout the subchapter; in particular, this involves existing tools and techniques required as background knowledge. Furthermore, this section provides an exemplary smart-city use case that will help to put these concepts into practice.

Concepts & Definitions

In this subchapter, the most important entities are the (pipeline) services and hosting devices. We provide a formal representation of them, as well as their joint assignment.

Definition 1 (Service, s)**.** *A service is a utility offered to other (micro-)services or end users to fulfill a dedicated function; a service is described as* $s = \langle in, f, out, M_s, Q \rangle$*, where* $\underline{in}$ *and* $\underline{out}$ *are the data ingested and produced, respectively, and* $\underline{f}$ *is the operation on the data.* $\underline{Q}$ *is a list of SLOs that must be fulfilled during operation, and* $\underline{M_s}$ *contains a list of metrics observable during service execution.*

Microservice architectures (e.g., [CQH19, Pet21]) form sequential processing pipelines by chaining together services. The implications between services are either known upfront or can be extracted with existing techniques [VF23]; the result is a dependency graph $K = (S, E)$, where directed edges (E) represent logical dependencies between services (S). Each dependency consists of a predecessor (p) and a successor (q). For two nodes $\{p, q\} \in S$, $pq \in E$ indicates that q is dependent on p. Also, q operates on results produced by p; hence, q is dependent on the time for executing f_p, and the network latency (nl) to transfer the result to q. The time for providing p's results (wt) to q is expressed as

$$wt(p, q, d) = time(f_p(d)) + nl_{p \to q} \tag{5.1}$$

where input data d is ingested to p. Any property of *in* and *out* (e.g., data size), as well as $time(f)$ and consequently wt, can be observed as part of M_s. Naturally, nl depends on the location (l) of host devices; however, notice that in the context of this subchapter, we assume nl to be independent of data size.

Definition 2 (Host, h)**.** *A device provides infrastructure for hosting services; a host is described as $h = \langle l, R_h, M_h \rangle$, where $\underline{l}$ represents the geospatial location of the device, $\underline{R_h}$ characterizes its entire processing resources, and $\underline{M_h}$ contains a list of metrics that gives evidence about ongoing operation.*

The current resource utilization is available as part of M_h. While hosts do not have an explicit representation of SLOs, their imperative requirement is to cap maximum utilization, e.g., maintain *cpu* load below 100%. The SLO fulfillment emerges from the deployments (e.g., hardware capabilities) and the current environment (e.g., service demand or network issues); given M_s, M_h, it is possible to evaluate all SLO.

The set of available hosts (H) is sampled from the CC's global pool of devices [PMN23]; ideally, these hosts possess the desired characteristics (i.e., to fulfill SLOs), but depending on availability, it must optimize assignments for arbitrary hosts.

Definition 3 (Assignment, *as*)**.** *An assignment indicates that a service is executed on a specific device; the assignment is described by $as = \langle s, h, R_s \rangle$, where $\underline{s}$ is the service executed at host $\underline{h}$, and $\underline{R_s}$ is the share of the host's resources (R_h) utilized by executing $\underline{s}$. A short notation for any as is $\underline{s} \diamond \underline{h}$.*

Resources of multi-tenant devices are commonly partitioned into VMs and containers, i.e., services deployed at the same device do not access the same share of physical (processing) resources [Pet21]. Nevertheless, the entirety of resources dedicated to n services assigned to a host h, plus all idle resources (R_I) equal the total amount of device resources (R_h).

$$R_h = R_I + \sum_{i=1}^{n} R_{s,i} \tag{5.2}$$

Each tenant is treated as an individual deployment; nevertheless, services interact indirectly by pooling resources from the same host. Hence, when assigning services over heterogeneous devices, the question extends from "*which device can fulfill SLOs to the maximum degree*," to "*how much resources would individual services demand.*" The precise implications of assigning a service to a host ($s \diamond h$) we call a "footprint"; both sides will be composed into one model, i.e., the MB.

Background

Bayesian Networks (BNs), as applied by Pearl [Pea09], are structural causal models represented as a Directed Acyclic Graph (DAG): edges between variables (e.g., $quality \rightarrow latency$) indicate conditional dependencies. For example, given that $quality = x$, what is the probability that $latency = y$, can in Bayesian terms be expressed as

$$P(latency \mid quality) = \frac{P(quality \mid latency) \times P(latency)}{P(quality)} \tag{5.3}$$

The causal edges in BNs are a distinctive feature that extends a system with explainability. Suppose we are interested in the behavior of a specific variable (x), we can explain x's state given its parents, children, and co-parent nodes. This subset is called its Markov Blanket (MB) – formally $\texttt{MB}(x)$ – as was introduced in Section 2.3.

Regardless of their size, systems can be divided into smaller modules (i.e., MBs); thus, managed and controlled on a convenient scale. Within previous work, we built explainable MBs around SLO-governed components [SPDD23, SPDD24b]. To understand observable processes, we trained a causal representation of a system's states – all contained within a MB. Conditional variable dependencies could answer questions like "if *quality* rises, what is the probability that *latency* rises too?", expressed by Eq.(5.3). Given $\texttt{MB}(latency)$, conditional SLO fulfillment (i.e., $latency \leq x$) was analyzed to indicate the optimal system configuration (i.e., $quality = z$) that fulfills SLOs.

SLOs follow the grammar $var \rightarrow rel \rightarrow thresh$, where $var \in \{M_s \cup M_h\}$, $rel \in \{\leq, \geq\}$, and $thresh \in \mathbb{Q}$; hence, possible examples are $latency \leq 10$, or $cpu \leq 95$. The second SLO type is supplied as $obj \rightarrow var$, where $obj \in \{min, max\}$. These objectives represent soft boundaries that are optimized during operation, such as $min(energy)$. Recall, that SLOs characterize the QoS and QoE of individual services and their interfaces, i.e., the quality expected of *in* and *out*, but there is no knowledge how SLOs affect other services. By design, these MBs contain only internal system variables.

Extracting the MB around multiple variables forms larger subsets. For a service (s), its host (h), and m SLOs (Q), its MB is the composition of the subsets expressed as

$$\texttt{MBE}(M_s, M_h, Q) = \bigcup_{i=1}^{m} \texttt{MB}(\texttt{BNL}(M_s \cup M_h), Q_i) \tag{5.4}$$

where MB and BNL are algorithms for MB extraction and BN learning, as presented in [SPDD23]; M_s and M_h contains multidimensional metrics monitored by executing $s \diamond h$. When it is impractical to provide data for Bayesian Network Learning (BNL) upfront, or there occur variable shifts, Active Inference, as in Chapter 6, provides a remedy.

Illustrative Scenario

City spaces are increasingly crowded with sensor networks, most of them even publicly accessible [GH18]. This provides application developers access to vast amounts of data and allows them to combine and assemble smart city services at will. Figure 5.1 exemplifies how a service pipeline might be plugged together – data flows along the red errors. In that context, the following services and requirements (written in bold) could be envisioned by an application developer:

- *SmartCamera* (grey) provides batches of images from IoT cameras at a traffic junction. Can customize video quality.
- *RoadAnalysis* (red) analyzes video streams of a road scene. Can get congested with **high number of frames**.

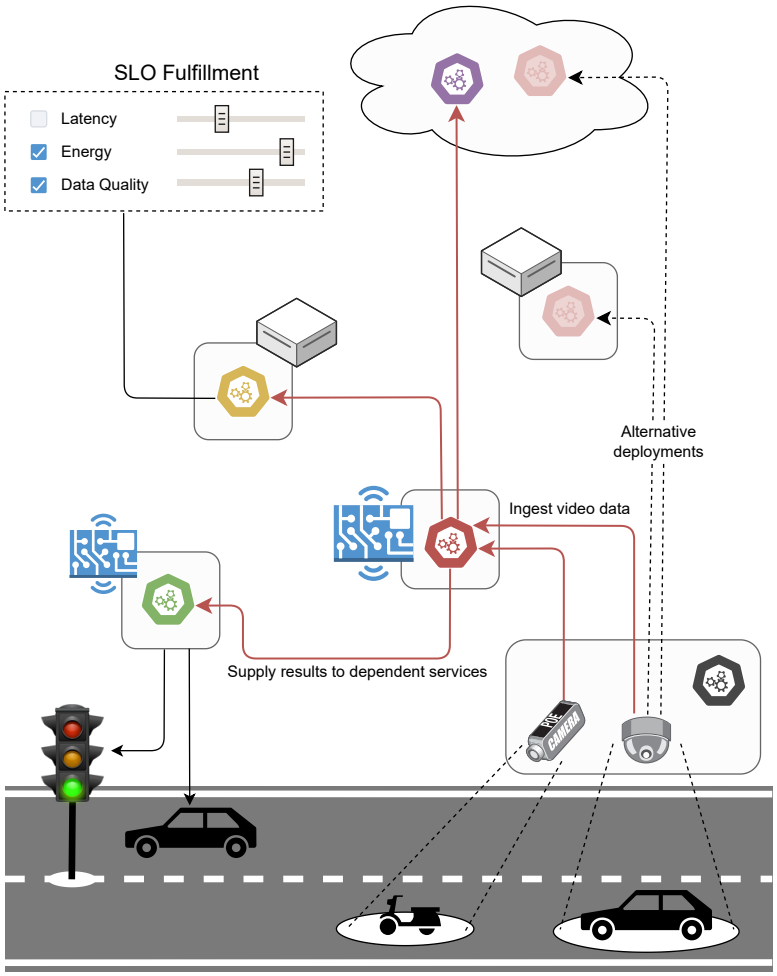


Figure 5.1: Smart-city services chained together as a compound application

- *TrafficRouting* (green) consumes information about traffic conditions and can control traffic lights or reroute vehicle navigation systems. Requires information **timely**.
- *TrafficPrediction* (yellow) assists local governments in estimating how traffic evolves throughout the day. Requires **large amounts** of **fresh** data to detect irregularities.
- *LiveMonitoring* (purple) allows remote inspection of analyzed road scenes. Requires **high video quality**.

The application developer combines the services as *SmartCamera* → *RoadAnalysis* → {*LiveMonitoring*, *TrafficPrediction*, *TrafficRouting*}; hence, the first two services in the pipeline provide their results to subsequent services. The immediate question here is how the requirements that each service poses constrain the QoS that *RoadAnalysis* and *SmartCamera* must provide – these are the transitive dependencies. The second question

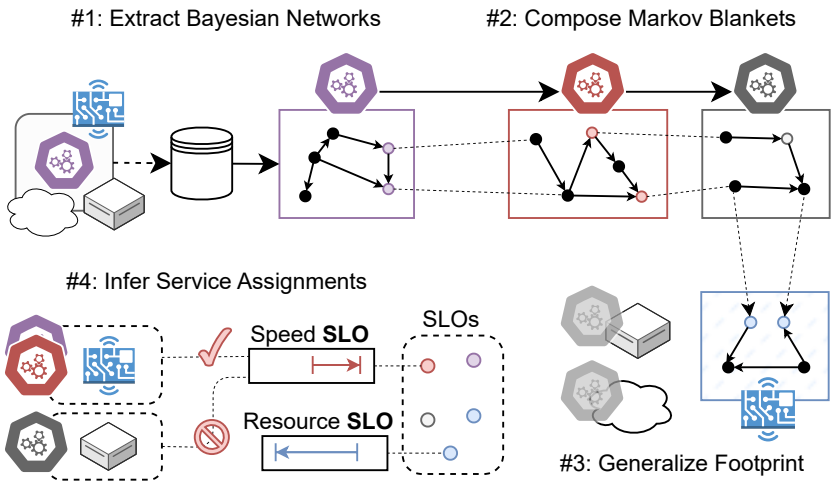


Figure 5.2: Methodology for maximizing the SLO fulfillment of service pipelines by composing MBs and assigning services over heterogeneous CC hardware

this unfolds is where to deploy individual services; preferably, application developers can stay agnostic and rely on the underlying CC infrastructure: depending on availabilities, services get assigned to hosts so that SLO fulfillment is maximized over the pipeline.

Both questions will be addressed as part of our methodology: application developers can customize *how* they would like their applications to operate (i.e., expressed in terms of SLOs) and rely on the assignment mechanism that finds the optimal hosting device for each service. To facilitate multi-tenancy for CC devices, this involves in-depth knowledge of the amount of resources required under a specific configuration.

5.1.3 Methodology

In the following, we present our 4-step methodology that assigns a microservice pipeline over heterogeneous resources, while maximizing SLO fulfillment. The sequential steps are embedded into the respective subsections below; to accompany explanations, Figure 5.2 provides a visual representation of the methodology: first, it (1) extracts MBs of individual services, then (2) performs MBC for the entire pipeline, (3) generalizes the service footprints, and (4) infers the assignment with maximum SLO fulfillment.

Markov Blanket Extraction

The training data for BNL is collected from past or ongoing operations; in the best case, services and their hosts were monitored over an extended period so that the underlying processes can be modeled accurately. Metrics of $s \diamond h$ are collected simultaneously (M_s, M_h); this tuple is appended to D, where the data of all empirically evaluated assignments is collected. To generate training data, each service should be executed

at least once at an arbitrary host $h \in H$; to avoid perturbation through concurrent services, each service requires an isolated environment that gives clear evidence about how many resources (R_s) a service utilizes. Further, metrics of dependent services must be captured under equal configurations, e.g., if a pipeline contains two sequential microservices *CameraWrapper* $\rightarrow$ *StreetAnalysis*, the data produced by *CameraWrapper* must be the exact same received and processed by *StreetAnalysis*. This can be assured by (1) capturing both services' metrics at the same time, or (2) maintaining their interface variables aligned, e.g., by processing the same video resolution. If this is impractical or there is not sufficient data available, Section 5.1.3 can provide a remedy.

The BN training data is clearly created at the hosting devices; however, the infrastructure provider can choose freely where to execute any substep of the methodology, including the BNL. Ideally, it is a central location in the CC (i.e., low latency to all hosts) that can spare sufficient resources for training; otherwise, the methodology could be split up and executed separately by different hosts.

Given the training data (D), each tuple (M_s, M_h) is ingested to `MBE`, which trains the BNs and extracts the MBs around multiple SLOs (Q). This is also contained in Algo. 5.1, which serves as a wrapper for the 4-step methodology; nevertheless, Lines 2-5 are dedicated only to MB extraction. Afterward, X contains all empirically evaluated footprints. The remaining functions (i.e., `MBC`, etc) are introduced in the next steps.

Algorithm 5.1: Wrapper for the 4 Methodology steps

Require: D, H, Q, $K = (S, E)$
Ensure: Z {Assignment with highest SLO fulfillment}
1: $X, V_d, V_c, F \leftarrow \emptyset$
2: {MB Extraction – Step 1}
3: **for each** (M_s, M_h) **in** D **do**
4: $\quad X \leftarrow \texttt{MBE}(M_s, M_h, Q_s) \cup X$ {Equals $s \diamond h$}
5: **end for**
6: $V_c, V_d \leftarrow \texttt{MBC}(E, Q)$ { – Step 2}
7: $F \leftarrow \texttt{FPG}(S, H, X)$ { – Step 3}
8: $Y \leftarrow \texttt{SASS}(E, V_c, V_d, H, Q, F)$ { – Step 4.1}
9: $Z \leftarrow \texttt{MAX\_AS}(Y, \text{“}joint\text{”})$ { – Step 4.2}
10: **return** Z

It is possible to logically separate the MBs of services and hosts, or rather, contained variables. However, none of the two exists without the other: service metrics (M_s) can only be observed during runtime; device metrics (M_h) give little insight when no service is executed. Hence, it makes no sense to train their MBs separately. Extracting the MB of a service also provides the impact on a hosting device – this is already their composition, what remains is to run MBC for services.

Markov Blanket Composition

Dependencies between services indicate that their internal states are influential to another service; thus, when aiming to maximize the SLO fulfillment of multiple services, it must be precisely determined *how* they affect each other. However, individual MBs cannot give evidence about variables external to their environment; also, training large BNs (e.g., [YKAQ22, WWC+21]) from one composite data set poses considerable requirements for BNL [SPDD24b]. To this extent, we create an overarching system model by composing the MBs of pipelined services. This assembles a common behavioral model where it is evident how one service's state affects another service's SLOs.

We assume that changes to individual services perpetuate along the service pipeline – this path is encoded in the dependency graph (K). Given K, we test dependencies between any two services $\forall p, q \in S, (pq \in E)$; we compose the MBs of $\bar{E}$ instead of $\bar{S}^2$ intersections, i.e., only along the pipeline instead of pairwise for all $s_1, s_2 \in S$. For p, q and two variables $v_p \in M_p$ and $v_q \in M_q$, we test the statistical dependency between v_p and v_q. Therefore, we compute the difference between their probability distributions by applying two-sample Wasserstein distance (WSD) [RGC15]. Given that there is a dependent variable v_d shared by p and q, Eq. (2.2) does not hold anymore; hence, their composition is linked by v_d.

Factors that, in reality, influence a service can be wrongfully missing from its MB because BNL could not consider them, e.g., the locations of dependent services are external. Such factors are also called confounding variables because they decrease model accuracy whenever they *would* have a conditional impact. Due to this, WSD is a good choice because it does not fail due to linear shifts in the distributions. For instance, $time(f_p(d))$ from Eq. (5.1) translates to $delay \in Q_q$, but $nl_{p \to q}$ cannot be represented without knowing both p's and q's location. Nevertheless, when testing their dependence, WSD can detect that they are dependent and that there is a linear confounder. Whether nl actually is *the* confounder is another question; however, the evaluation does not contain other confounders by design; hence, we will take it for granted in this subchapter and address this issue in future work.

More commonly, there is no confounder involved and two dependent variables can be directly mapped, e.g., a video $resolution \in M_p$ likely resembles the respective consumer's $v_q \in \texttt{MB}(q)$. To differentiate between confounded relations and the latter type, we compute the mismatch between states in the probability distributions. Since the variable distributions in the MBs represent discrete values, we use Jaccard similarity to quantify the divergence between states of v_p and v_q.

Pairwise dependency tests would have to cover $\texttt{MB}(p, Q_p) \times \texttt{MB}(q, Q_q)$ variable combinations; however, we are only interested in the implications to the successor's SLOs (Q_q) – reducing complexity to $\texttt{MB}(p, Q_p) \times Q_q$. For two dependent variables $v_p v_q$ with $v_q \in Q_q$, the SLO around v_q constrains the states that p can take without violating q's SLO; p does not differentiate between its "own" SLOs (i.e., Q_p) and transitive ones imposed by q – both restrict p's potential actions. Algo. 5.2 summarizes how the pipeline edges

Algorithm 5.2: Markov Blanket Composition (MBC)

Require: E, Q
Ensure: V_c, V_d {Optimal assignment from all options}
1: **for each** (p, q) **in** E **do**
2: **for each** (v_p, v_q) **in** $mb(p) \times Q_q$ **do**
3: **if** $\texttt{WSD}(v_p, v_q) < 0.1$ **then**
4: **if** $\texttt{JD}(v_p, v_q) > 0.9$ **then**
5: $V_c \leftarrow v_q \cup V_c$
6: **else**
7: $V_d \leftarrow v_q \cup V_d$
8: **end if**
9: **end if**
10: **end for**
11: **end for**
12: **return** V_c, V_d

(E) are traversed to run pairwise dependency tests. The results are two lists of (simply) dependent (V_d) or confounded variable relations (V_c).

Generalize Service Footprint

The implications of running a service at a particular host are unique for this pair, i.e., how much and which resources are utilized. To find the assignment with the highest SLO fulfillment, a simple method is comparing the implications of all combinations [SPDD23]. Empirically testing all these pairs, however, is exhaustive for $S \times H$ combinations, but limiting potential assignments to those run empirically – X in Eq. (5.1) – must also be avoided. For example, $\{h_x, h_y\} = H$ and $\{s_x\} = S$ are available for assignment; however, there exists no empirical information on $s_x \diamond h_y$ – a likely situation if h_y was added recently to the device fleet – so that $s_x \diamond h_x$ is the only assessable assignment. The infrastructure provider, however, disposes of rich contextual information about services and devices, which allows estimating the SLO fulfillment of hypothetical assignments that were not tested empirically.

In particular, we use two information sources for this: The first is the metadata description of services [MPN+23a], e.g., the primary purpose or the position in the pipeline. Our assumption is that services have similar hardware implications depending on their position (pos) in K, e.g., consumers located at the end have low hardware impact. The second source is a sampling mechanism for CC infrastructure [PMN23], which provides a relative comparison of the hardware capabilities for devices in H; $h_1 > h_2$ implies that h_1 has more resources available. This information is used to extrapolate from empirically evaluated assignments (X) to hypothetical ones (F); hence, F will contain $(s \diamond h)$ for $\forall s \in S, h \in H$.

As depicted in Algo. 5.3, we distinguish four options to estimate the footprint $s_x \diamond h_x$

Algorithm 5.3: Footprint Generalization (FPG)

Require: S, H, X
Ensure: F {Generalized footprints for services $\diamond$ hosts}
1: **for each** s_x, h_x **in** $S \times H$ **do**
2: **if** $(s_x \diamond h_x) \in X$ **or** $\exists h_y(h_y \leq h_x \wedge s_x \diamond h_y \in X)$ **then**
3: $F \leftarrow (s_x \diamond h_x) \cup F$ {Case 1 and 3}
4: **else if** $\exists s_y(pos(s_y) = pos(s_x)$ **and** $s_y \diamond h_x \in X)$ **then**
5: $F \leftarrow \texttt{MERGE}(s_x \diamond h\_, s_y \diamond h_x) \cup F$
6: **else**
7: {assumes $\exists h\_(s_x \diamond h\_ \in X)$}
8: $F \leftarrow \texttt{PENAL}(s_x \diamond h\_, r(h_x, h\_)) \cup F$
9: **end if**
10: **end for**
11: **return** F

of any $s_x, h_x \in S \times H$: **(i)** the assignment was empirically evaluated (Line 3); **(ii)** a comparable service (i.e., same pos) was evaluated at h_x (Line 5); for this, we merge $s_x \diamond h\_$ (i.e., any host) and $s_y \diamond h_x$ by replacing all $\bar{M}_h$ variables of $h\_$ with those of h_x. If **(iii)** s_x was empirically evaluated at a weaker device (h_y), we reuse $s \diamond h_y$ because we assume it cannot perform worse on h_x (Line 3 **or**), whereas for $\forall h_y(h_y > h_x)$, we cannot know the implications to more constrained devices. This case can be adapted for comparable services as well – case *(ii)*. The last option **(iv)** is using the relative hardware difference (r) between h_x and an arbitrary host ($h\_$), where s_x was empirically evaluated, and penalizing the SLOs contained in $s_x \diamond h\_$ according to r (Line 8). The interested reader finds implementations of MERGE and PENAL in the code artifact.

Service Assignment

The last step remaining is to identify the assignment that maximizes overall SLO fulfillment for all pipeline services; to that extent, we compare all hypothetical assignments and choose the one with the highest fulfillment as the sum

$$\sum_{s=1}^{S} Q_s + \sum_{h=1}^{H'} Q_h \tag{5.5}$$

where H' are all assigned hosts; notice that $|H'|$ can be 1 for a multi-tenant scenario. For $\forall s \in S$ and $\forall h \in H$, its footprint is retrieved by $F(s \diamond h)$ – this comprises the conditional variable assignments for the service's $\texttt{MB}(s, Q_s)$ and the respective hardware utilization of the host as $\texttt{MB}(h, Q_h)$. Transitive dependencies imposed by other services are contained in $\{V_c, V_d\}$; for a service s, all variables that share dependencies with other services are contained in ext_s (Line 2).

Given the footprints and the dependent variables, the SLO fulfillment of each service (i.e., constrained by ext_s) is obtained by ingesting the SLO thresholds (Q). As depicted

Algorithm 5.4: Service Assignment (SASS)

Require: E, V_c, V_d, H, Q, F
Ensure: Y {Estimated SLO fulfillment per service × host}
1: **for each** $(s, \_), h$ **in** $E \times H$ **do**
2: $ext_s \leftarrow \forall v(v \in mb(s)$ **and** $v \in (V_c \cup V_d))$
3: **if** $(ext_s \cap V_c) \neq \emptyset$ **then**
4: **for each** $(s_1h_1, ..., s_Sh_H)$ **in** H^S **do**
5: $Y \leftarrow \texttt{INF}(F(s \diamond h), (Q_s \cup Q_h \cup ext_s \cup s_1h_1)) \cup Y$
6: **end for**
7: **else**
8: $Y \leftarrow \texttt{INF}(F(s \diamond h), (Q_s \cup Q_h \cup ext_s)) \cup Y$
9: **end if**
10: **end for**
11: **return** Y

in Algo. 5.4 (Line 5 & 8), we infer this from a footprint by providing variable assignments of the thresholds, and hypothetical deployments of dependent services. Internally, INF applies variable elimination (as in Section 4.2.3) to marginalize all variables $\notin Q_s \cup Q_h$; hence, ending up with SLO fulfillment only. The distinction (Line 5 **or** 8) takes care of the confounded variable, i.e., *latency*. Estimating the conditional fulfillment of an SLO ($latency \leq x$) requires considering the hypothetical location of all services in the pipeline, i.e., H^S possible assignments.

Once SLO fulfillment under all possible assignments is inferred (Y), it remains to identify the optimal assignment; this is the role of MAX_AS. Here, our approach is to act either *greedy* or *joint*: *greedy* assigns services sequentially by marginalizing the hypothetical deployments of other services; hence, it estimates the expected SLO fulfillment of a service without considering where dependent services will be assigned, then, it chooses the host with the highest one. The *joint* approach, however, assigns services collectively by calculating the overall SLO fulfillment that all services would reach for a hypothetical assignment. Whenever a combination requires assigning multiple services to a single host h, the respective hardware utilization (e.g., CPU load) is appended to the existing one; as shown later in Table 5.4 (red), such combinations can exceed the available resources, and hence, are disregarded because they violate the host's SLOs (Q_h).

When using *joint*, comparing all assignments in Y makes it evident which combination promises the highest SLO fulfillment. An advantage of *joint* is that it can precisely evaluate *latency* SLOs and estimate the hardware utilization of multi-tenant assignments upfront, which avoids sub-optimal *greedy* assignments; however, the drawback of *joint* is its high combinatorial complexity. The interested reader will find the implementation of MAX_AS in the attached code artifact.

This concludes our 4-step methodology, which started by analyzing the dependencies between services and devices according to their MBs. To infer the optimal assignment

between services and hosts, we use composed MBs and estimate the expected SLO fulfillment of different hypothetical assignments. In the next section, we now present how this methodology was implemented and evaluated.

5.1.4 Evaluation

To evaluate the ideas presented in this subchapter, we focus on the individual steps of the methodology and highlight whether the outcome fulfills the research goals. For this, we first outline how the methodology and the evaluation environment were implemented; then we present the experimental setup (incl. services and hosts) and the results of our experiments. Lastly, we summarize and discuss the implications of these results.

Implementation

We provide a Python-based prototype[2]that implements our methodology; this includes all services used to generate BNL training data. To extract the MBs, our prototype requires tabular CSV files that are internally processed with pgmpy [AT23]; this step combines the data for each service, i.e., regardless of whether it was hosted at *Xavier* or *Orin*. This increases the general validity of trained BNs and adds conditional information on different device types, e.g., $P(cpu = x \mid type)$.

For $\forall p, q \in K; pq \in E$, we run pairwise variable tests to identify dependent variables; the required statistical tools (e.g., WSD) are native to Python. Whenever WSD ≤ 0.1[3], we flag it as a potentially confounded relation. Depending on the Jaccard similarity, i.e., states match $\geq 90\%$, we declare whether it is confounded or simple. In both cases, additional constraints (i.e., external SLOs) are added to the dependent service's MB, which will be provided to `INF` (Algo. 5.4). Next, the footprint generalization is integrated into the inference: we iterate over all $S \times H$ combinations and estimate (1) expected SLO fulfillment and (2) hardware utilization. The default case is to perform *joint* service assignment, which internally compares assignments for H^S combinations; whenever this exceeds the maximum complexity, *greedy* is an alternative.

To determine the network latency (nl) between different hypothetical hosts, we rely on the knowledge of the infrastructure provider: we assume a tabular representation (i.e., $H \times H$) for this, which provides for $h_x, h_y \in H$ the respective $nl_{h_x \to h_y}$.

Experimental Setup

To embed and evaluate our implementation in a realistic setup, we rebuild the scenario presented in Section 5.1.2; this means, that we use the five discussed services to instantiate our service set (S): *RoadAnalysis* was implemented and executed physically – for this we used the YOLOv8[4] model to detect and highlight objects within a road scene[5]. The

[2]Prototype artifact available at GitHub, accessed Feb 28th 2024
[3]These two thresholds (i.e., 0.1 and 0.9) rendered satisfactory results.
[4]YOLOv8 model from ultralytics GitHub, accessed Feb 28th 2024
[5]Road racing video scene from YouTube, accessed Feb 1st 2024

other services were simulated based on the data ingested or produced by *RoadAnalysis*. According to services' position and purpose in the pipeline, we will use shorter synonyms: Producer (P) for *SmartCamera*, Worker (W) for *RoadAnalysis*, and Consumer ($C\_$) for {*LiveMonitoring*, *TrafficPrediction*, *TrafficRouting*}. Services depend on the data provided by their predecessor; hence, the dependency graph follows the inverse data flow from Figure 5.1, i.e., $K \sim \{C_1, C_2, C_3\} \rightarrow W \rightarrow P$.

SLO variables and thresholds are chosen according to the requirements in the service description. For simplicity, we assumed that $C_1, C_2,$ and C_3 have the same tractable variables; two of them are image *size* and data *rate*, which reflect the video properties received by *Consumers*. Table 5.1 contains an overview of all SLOs: W and $C\_$ must ensure QoS SLOs, e.g., maintain $latency \leq x$ or ensure image $size \geq y$; P must minimize *energy*, i.e., a soft-boundary SLO, that is optimized as long as it does not violate any hard SLO (e.g., *latency*).

Table 5.1: SLOs inherent to each service

	P	W	C_1	C_2	C_3
latency	–	–	$\leq$ 1s	$\leq$ 70ms	$\leq$ 40ms
image size	–	–	$\geq$ 720p	–	–
data rate	–	–	–	$\geq$ 25f	–
delta	–	$\leq \frac{1}{fps}$	–	–	–
*energy**	$min()$	–	–	–	$min()$

We provide a sampled set of devices (H) to host services, as shown in Table 5.2: for each device, it contains a short ID, hardware stats, and how these stats (p, q) are classified relative to other devices in H. Given this information, it is evident how heterogeneous the devices are, e.g., the processing resources of *Server* dwarf *Nano*'s. Additionally, devices equipped with a GPU can accelerate *Worker*'s video processing through NVIDIA Cuda; this underlines the importance that hosts have on services and SLO fulfillment. The last column contains the networking delay (nl): we assume that devices are perfectly aligned on a single line so that the nl between two hosts can be computed based on their nl to *Nano*, e.g., communicating from *Xavier* to *Orin* takes $5ms - 3ms = 2ms$.

We create different evaluation scenarios by repeatedly selecting subsets of $S \times H$, as visible in Table 5.3: for each of the 8 scenarios, it shows available services and hosts. For example, for t_0, $\{P, W, C_1\}$ must be assigned over $\{S, L, O, X, N\}$. The scenarios could reflect different positions in time, where more or fewer hosts are available for different services. Nevertheless, we evaluate scenarios separately and do not update assignments at runtime. Depending on the service descriptions, we decide that P must always be assigned to *Nano* – the local device that bundled IoT video streams; further, we assume that smart-city infrastructure (i.e., fed by C_3) is located close to N, hence, C_3 must be assigned to any $h \in \{N, X, O\}$.

[6]Prices adopted from sparkfun, accessed Feb 13th 2024

Table 5.2: Hosting devices used for implementing and evaluating the methodology

Full Device Name	ID	Price[6]	CPU [1,4]	GPU [0,3]	$nl_{N\rightarrow\_}$
Custom Server Build	*Server (S)*	2500 €	Very High (4)	High (3)	20 ms
ThinkPad X1 Gen 10	*Laptop (L)*	1700 €	High (3)	None (0)	10 ms
Nvidia Jetson Orin	*Orin (O)*	500 €	Medium (2)	Medium (2)	5 ms
Nvidia Jetson Xavier	*Xavier (X)*	300 €	Medium (2)	Low (1)	3 ms
Nvidia Jetson Nano	*Nano (N)*	200 €	Low (1)	None (0)	——

Table 5.3: Services and hosts available for assignment

	Services					Hosts				
t_i	P	W	C_1	C_2	C_3	S	L	O	X	N
0	✓	✓	✓	–	–	✓	✓	✓	✓	✓
1	✓	✓	–	✓	–	✓	✓	✓	✓	✓
2	✓	✓	–	–	✓	✓	✓	✓	✓	✓
3	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
4	✓	✓	✓	✓	✓	–	–	✓	✓	✓
5	✓	✓	✓	✓	–	–	–	✓	–	✓
6	✓	✓	✓	✓	–	✓	–	–	–	✓
7	✓	✓	✓	✓	–	–	✓	✓	–	✓

Results

We execute the experimental setup on the prototype of our methodology; first, we show the resulting MB composition, explain dependencies between services, and present the inferred assignments. Afterward, we assess the quality of the assignments according to three factors: (1) we observe their empirical SLO fulfillment at runtime, and (2) compare the runtime fulfillment to the expected fulfillment. Additionally, we provide (3) an exhaustive comparison of how inferred assignments score compared to all alternative assignments. These three factors evaluate our solution in terms of QoS and QoE and allow us to judge the feasibility of our approach.

Inference through MBC Figure 5.3 shows the intermediary outcome after two steps, i.e., after MB extraction and composition. The purple, yellow, and green services represent *Consumers*, red the *Worker*, and grey the *Provider*. The upper colored squares contain services' MBs, including SLO variables (fully-colored nodes) or such related to SLO fulfillment (black nodes). The blue squares contain hosts' respective MBs, and how the services impact its variables. Service dependencies are represented by dashed lines and colored margins, e.g., *size* (purple) is found dependent on *pixel* (red), and in further consequence on *resolution* (grey). Hence, constraining the service provided by *SmartCamera*. Now whenever red looks to infer a variable assignment that fulfills its own SLO (i.e., *in_time*), it constrains this to states that fulfill purple's SLOs as well.

The "soft" SLO, i.e., $min(power)$, does not pose hard constraints to `INF`; however, given

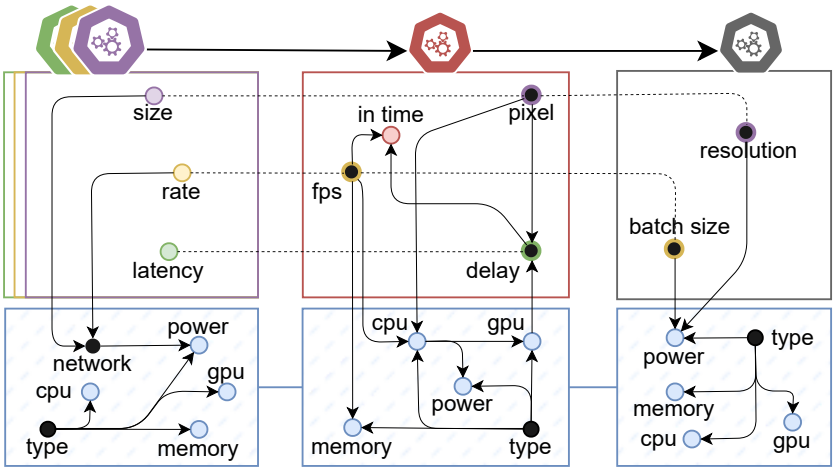


Figure 5.3: Composed Markov blankets for *Consumers*, *Worker*, and *Provider*

Table 5.4: Select assignment given service & host SLOs (t_2)

#	SLO Σ	W	CPU	GPU	Mem	C_3	Power Σ
1	1.70	*Orin*	50	30	119	*Orin*	8 W
2	1.52	*Orin*	24	30	73	*Xavier*	15 W
3	0.92	*Server*	3	31	12	*Laptop*	97 W
...	...	...	...	...	...	...	...
24	0.00	*Nano*	122	35	93	*Laptop*	26 W
25	0.00	*Laptop*	54	0	27	*Laptop*	21 W

multiple assignments with equal SLO fulfillment, the one with lowest *power* is chosen: For example, the MBC between P and any host has encoded that low *resolution* and *batch* decrease *power*; hence, from the parameter space that fulfills these SLOs, it chooses *480p* and *15fps*, i.e., the state with lowest *power*. Notice, the most influential variable for the host's MB is always the device *type* – whether *Xavier* or *Server* hosts the service has a big impact on the conditional fulfillment of hosts' SLOs (Q_h).

We perform the remaining two steps of the methodology and provide the inferred assignments for each scenario in Table 5.5. Each scenario's first line (i.e., *infer*) shows how services should be assigned (i.e., to optimize SLO fulfillment) given the available hosts. For example, at t_2, the pipeline $S = \{P, W, C_3\}$ had to be assigned over $H = \{S, L, O, X, N\}$; the inferred assignment is $\{P \diamond N, W \diamond O, C_3 \diamond X\}$, where $P \diamond N$ is preconditioned.

Table 5.4 exemplifies why W was assigned to O at t_2: under the hood, the SLO fulfillment of all possible assignments (Y) was compared in a *joint* fashion. This evaluates the fulfillment given all 25 ($= H^2$) hypothetical deployments of W and C_3. In #1, we estimate that the collective SLO fulfillment (i.e., $Q_W + Q_{C3}$) is 1.7; however, this assumes that W and C_3 are both deployed at *Orin*, which is, on one hand, desirable because keeping W close to C_3 benefits its *latency* SLO, but on the other hand, it is estimated that this

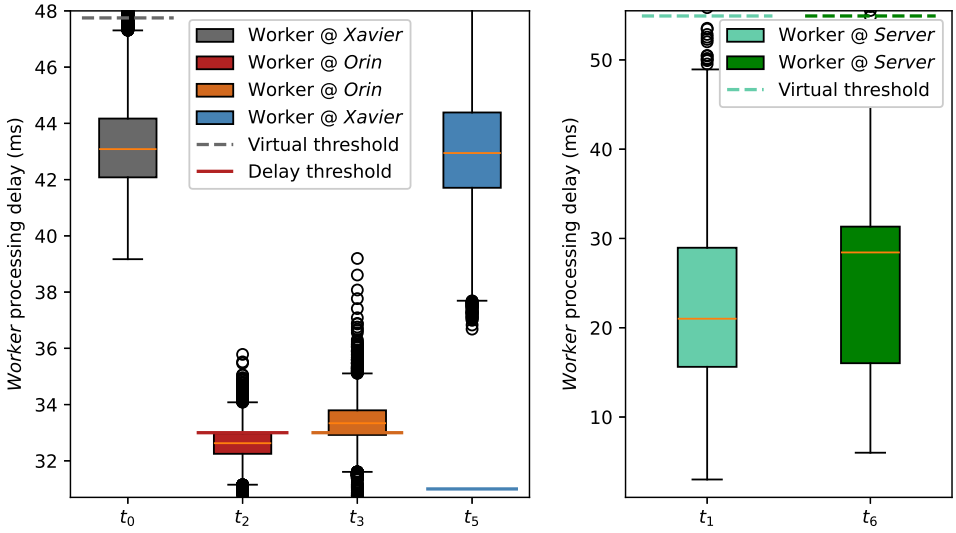
Table 5.5: SLO fulfillment of assignments (*infer* / *eval*)

		Services			
t_i	Mode	W	C_1	C_2	C_3
0	*infer*	X : 0.99	O : 1.00	–	–
	eval	X : 1.00	O : 1.00	–	–
1	*infer*	S : 1.00	–	S : 1.00	–
	eval	S : 0.98	–	S : 0.99	–
2	*infer*	O : 0.70	–	–	X : 0.82
	eval	O : 0.75	–	–	X : 0.76
3	*infer*	O : 0.31	L : 1.00	L : 1.00	X : 0.36
	eval	O : 0.28	L : 1.00	L : 1.00	X : 0.28
4	*infer*	O : 0.32	X : 1.00	X : 1.00	O : 0.39
	eval	O : 0.29	X : 1.00	X : 1.00	O : 0.29
5	*infer*	X : 0.00	X : 1.00	N : 1.00	–
	eval	X : 0.02	X : 1.00	N : 0.99	–
6	*infer*	S : 1.00	S : 1.00	S : 1.00	–
	eval	S : 0.97	S : 1.00	S : 0.99	–
7	*infer*	O : 0.95	L : 1.00	L : 1.00	–
	eval	O : 0.99	L : 1.00	L : 0.99	–
	Err ⌀	0.02	0.00	0.01	0.06

would exceed *Orin*'s *memory* (red cells). Hence, assignment #2 (green) promises the highest SLO fulfillment, whereas #24 shows that *Nano* would be incapable of running *Worker*, both in terms of service requirements and hardware limitations (orange).

Quality of Assignments Apart from the expected SLO fulfillment, Table 5.5 also provides the experimental results of the assignment at runtime (*eval*). For each scenario, we tracked the services' performance at their respective hosts for 10 min, which generated in total roughly 70.000 observations. The last table row contains the average prediction error (i.e., over all scenarios) between expected and actual SLO fulfillment.

Figure 5.4 shows the distribution of the processing *delay* in different scenarios; the plots are separated due to the y-axis scale – *Server* has significantly lower processing *delay* per frame. The solid lines express the threshold (i.e., minus overall nl) that W has to meet to satisfy the *latency* of dependent *Consumers*. For example, in t_2 and t_3 the most restrictive *latency* is imposed by C_3, i.e., $40ms$ in Table 5.1; given the inferred assignment $\{W \diamond O, C_3 \diamond X\}$, we subtract $nl_{N \to O} = 5ms$ and $nl_{O \to X} = 2ms$, and set the bar to $33ms$. Table 5.5 confirms the validity of these distributions: t_2 reached 0.75 fulfillment and t_2 only 0.28. The difference between t_2 and t_3 occurs due to t_3 demanding higher *resolution*, which impacts *delay* (see Figure 5.3). Dotted thresholds are virtual boundaries that fall outside the y-axis (i.e., 70 or $1000ms$).



(a) t_0, t_2, t_3, and t_5 (b) t_1 and t_6

Figure 5.4: Processing delay of *Worker* at their assigned host $\in\{X, O, S\}$, combined with the threshold they must meet to fulfill all their consumers' SLOs

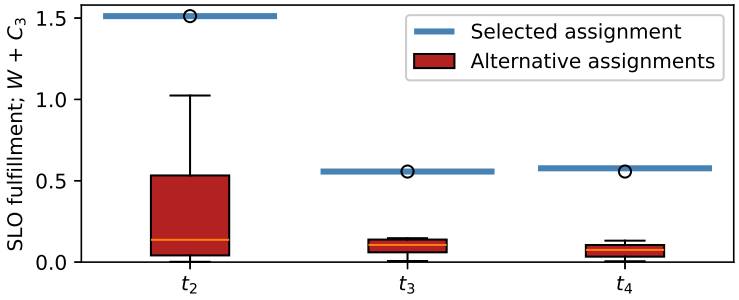


Figure 5.5: Combined empirical SLO fulfillment of *Worker* and C_3 for the selected assignment, compared to the SLO fulfillment of all alternative assignments

Finally, Figure 5.5 shows the SLO fulfillment of the inferred assignment (blue line) in comparison to all alternative combinations. Since P does not pose any hard SLOs, we calculate overall SLO fulfillment as $Q_W + Q_{C3}$. The boxplots contain the 25 combinations of how these services can be deployed over H, all evaluated empirically over 10 min.

Discussion

This section summarizes presented results and highlights their implications: We report that (1) the MB extraction provided interpretable relations within individual MBs – the *in_time* SLO was correctly attributed to *fps* and *delay* (see Figure 5.3); the MB composition was able to (2) detect dependencies between services and flag *latency* as

confounded due to the underlying *nl*. Further, while W was not empirically evaluated at *Nano*, we (3) correctly estimated that *Nano* exceeds SLOs from service and hosts (orange table cells), which was estimated from the relative device capabilities.

Given the composed MBs, it could (4) consider the transitive SLOs imposed by other services – a good example is comparing t_2 and t_3 in Figure 5.4, which shows how *delay* changed due to more restrictive SLOs from C_1. Further, we (5) maximized the SLO fulfillment given a heterogeneous list of hosts, e.g., t_1 in Table 5.5 could roam freely and save energy by assigning W to X, whereas for t_5 the best option was still unsatisfying due to tight constraints (from Table 5.1). The fact that (6) we identified the optimal assignments (Figure 5.5) was mainly driven by low prediction errors (see Table 5.5); ideally, this error will be fed back to improve predictions.

Limitations

While this subchapter presents a novel approach for raising SLO fulfillment of microservice pipelines, there remain limitations that must be addressed in future work; in the following, we will discuss three of them in more detail. Firstly, the initial process of determining the MBs of individual microservices can be computationally expensive and difficult to scale in large, dynamic networks. In order to avoid any overhead impeding regular device operation, it requires dedicated experiments that analyze the scalability of the approach. This must include a larger number of services and variables, as well as the methodology's performance on heterogeneous hardware.

Secondly, training an accurate BN and its corresponding MB requires substantial and high-quality data. In environments where data is sparse, noisy, or non-representative, the reliability of the MB and any respective inference decisions can be compromised. It remains to evaluate the presented approach in such an environment. Thirdly, IoT and Edge computing environments are often dynamic, with changes in node availability, service requirements, and network conditions. Static BNs might not adapt to such changes, making the MB outdated and less accurate over time. This issue was partly addressed in previous work [SPDD24b, SPDD24a], which focused on capturing changes in variable distribution. Nevertheless, dynamic retraining of the BN structure remains an open challenge.

5.1.5 Related Work

In the context of this subchapter, we identified two main areas of related work that intersect with our research: (1) modeling large-scale BNs to estimate how system changes perpetuate or can be countered, and (2) optimizing service deployments for constrained devices according to QoS requirements.

Large-scale Bayesian Network Modelling

To assess the resilience of a pipeline system, Yazdi et al. [YKAQ22] presented a dynamic BN that provides insights under which conditions QoS can be assured. Extending to

compound systems, Chen et al. [CQH19] provided a dynamic causality graph called *CauseInfer* that pinpoints issues during runtime. *CauseInfer* uses a two-tier mechanism that splits the system into a device and a service layer. To trace fault propagation within a vehicle control network, Wang et al. [WWC+21] transformed a dynamic fault tree into a BN; this could infer the probability of faults under different hypothetical setups. Multiple works use BNs for anomaly detection in IoT systems [Tog22, SAR+21, OSF22]; however, they are more focused on detecting, instead of mitigating them. The largest BNs in comparable literature were provided by Mengshoel et al. [MPK09] as a large-scale diagnostic system for simulating aircraft parts; however, individual blankets were separated without intersections. BNL is still an actively developing field, which is underlined by Kitson et al. [KCG+23]; they provide a comprehensive overview of BNL techniques and algorithms that help create accurate causal models.

Given these works, we conclude that BNs are used for fault detection or system behavior prediction. Most graphs featured a single large model, which appears feasible given a central data set; however, in the CC, services (i.e., data sources) are distributed, and training large BNs poses high requirements to edge device. *CauseInfer* composes a model from multiple subgraphs; nevertheless, it lacks a notation of hardware utilization based on deployed services. Contrarily, our method assembles a model at finer granularity.

QoS-Aware Deployment in the Computing Continuum

To find optimal services configurations for multi-tenant edge devices, Zhang et al. [ZZL23] presented *Octopus*, which predicts SLO fulfillment of two variables based on a deep neural network. To avoid resource contention, Qiu et al. [QBJ+20] created *FIRM*, which predicts the resource usage of services for a multi-tenant device. Cardelli et al. [CGGN+18] designed an autonomous elasticity mechanism for Cloud and Edge that ensures QoS of service chains; however, they did not implement it. Khoshkholghi et al. [KM22] presented a deployment and load-balancing mechanism that assured QoS of Edge function pipelines through deep learning. To maximize user satisfaction, Sheu et al. [SPJC18] propose a model deployment algorithm for the Edge that considers hardware limitations. The work of Zobaed et al. [ZMC+22] allows to meet the latency constraints of multi-tenant applications. Confronted with the erratic activities of mobile users, Lu et al. [LWL+23] predicted how QoS could be assured through service updating. Their work provisioned services for multi-tenant deployments. To assure high QoS for composite services, Mehdi et al. [MBB13] selected individual services based on a computed trust score. They would then construct a composite service through BNL.

Considering presented work, we conclude that multi-tenancy is common for the Edge; there are several works that estimate resource implications of services. However, none of them would consider the generalization of service footprints or transitive dependencies between services (i.e., tenants). The exception is [MBB13]; however, they lack implications on the underlying hardware. Contrarily, our method provides the precise utilization per tenant for composite service pipelines.

5.1.6 Summary

This subchapter presented a statistical reasoning model for assigning a microservice pipeline over a heterogeneous set of devices, which are located from the Edge to the entire CC. To maximize the requirement fulfillment throughout a pipeline, our methodology analyzes dependencies between services; this constrains the operation of individual services according to the quality expected by dependent services. The evaluation of our prototype showed that we could infer the optimal assignments given a mutable list of services, hosting devices, and SLOs that had to be ensured. We envision our methodology as a central tool to simplify the development of compound services, e.g., in smart cities, where overall SLO fulfillment is optimized for whatever resources the CC has available. In that regard, future research will focus on runtime mechanisms that allow services to scale vertically or horizontally over the hosting devices.

5.2 Diffusing High-Level SLOs in Microservice Pipelines

Complex interactions within microservice architectures obfuscate the implications of individual services to high-level requirements. This becomes even more grave for multi-tenant and multi-vendor scenarios, like Edge computing, where different stakeholders might specify opposing Service Level Objectives (SLOs), e.g., minimizing both energy consumption and response time. To avoid contradictions within SLOs and to infer how SLOs can be fulfilled, this subchapter presents a methodology that diffuses high-level SLOs into multiple lower levels of SLOs and parameter assignments. Thus, it becomes clear how individual sub-processes contribute to high-level SLOs, and how these must be configured to foster their fulfillment. We evaluated our methodology for several microservice pipelines, where the challenge is to ensure multiple high-level SLOs (e.g., customer satisfaction) by finding and constraining all influential factors. The results show that by inferring multiple layers of lower-level constraints, we can fulfill high-level SLOs up to 100%. Notably, we could extract that the restrictiveness of low-level SLOs and the occurrence of conflicts have a severe impact on SLO fulfillment.

The remainder of this subchapter (see footnote 1) is structured as follows: Section 5.2.2 introduces background knowledge and related work; Section 5.2.3 presents our methodology for diffusing SLOs, which is implemented and evaluated in Section 5.2.4. Finally, Section 5.2.5, concludes this subchapter with a future scope.

5.2.1 Introduction

Many current internet-based applications are composed of a network of microservices, each providing specific functionality to the application; common instances are data transformation pipelines or machine learning pipelines. These instances benefit particularly from service-oriented architecture [HS05], which improves both modularity and flexibility, while keeping services loosely coupled – boosting scalability. However, each service's performance depends on its neighboring services, i.e., those that send or receive data from it. Hence, if its performance deteriorates, this affects neighboring services and, ultimately, the overall application performance.

To assess the application's overall performance, Cloud computing uses Service Level Objectives (SLOs); typical SLOs are response time or availability, which refers to the entire application, but not to individual services [ZZL23]. Whenever an SLO is violated, services are scaled to reestablish the expected performance level. However, Cloud providers are generally unaware of which services should actually be scaled; simply scaling all services (or candidates) can turn out extremely inefficient. In this sense, there exist works that pinpoint which services to scale by finding applications' critical path [ZRR+22] or performing causal analysis on service architectures [CQH19]. Applying such methods requires considerable time, which can propagate failures in large and distributed applications [SMB21].

However, when looking into novel computing paradigms, such as Edge computing [SCZ+16] or the computing continuum [DCPD23], some Cloud-based rules simply do not apply:

First, only parts of their infrastructure can be scaled; secondly, both paradigms assume a multi-tenant and multi-vendor scenario [CPDM+23a], i.e., infrastructure is used to host multiple applications, which belong to different stakeholders. When stakeholders set their SLOs, it is challenging to identify whether these are compatible; in many cases, SLOs of different stakeholders can be opposing – causing conflicts. For instance, infrastructure providers could aim at hosting several applications on devices, and hence limit processing time available per tenant (i.e., the applications); application developers, on the other hand, want maximum quality for end users. Attempts to satisfy both result in a contradiction, which must be circumvented to avoid undesired system behavior.

SLOs can be used to constrain different levels of abstraction, from high-level goals such as response time and client satisfaction, down to hardware utilization of individual devices. For application stakeholders, the most intuitive choice is to start posing SLOs that look at the overall performance of the system [NMP+20]; we call the resulting constraints “high-level SLOs". These high-level SLOs can target different aspects of QoS or Quality of Experience (QoE), such as high video stream resolution, or decreased energy consumption, but also cost. The question remaining is how to determine under which conditions a system can actually fulfill them. For example, what does it take to minimize energy consumption? The answer might be to restrict CPU load or other resource utilization; we call these derivative constraints “low-level SLOs". However, it is tedious for application developers to specify SLOs for increasingly large microservice applications; in most cases, they would also lack in-depth knowledge of how to diffuse a high-level SLO into the corresponding low-level SLOs.

To decrease the overall complexity of system design, we present a 3-step methodology that diffuses high-level SLOs throughout an application, which means splitting them up into a set of lower-level SLOs. To control all of these SLOs and maintain them within bounds, the methodology identifies parameters that causally influence the required SLO fulfillment, and how they should be assigned. Finally, the methodology detects conflicts caused by high-level SLOs, which might occur at any abstraction level. If possible, these conflicts are resolved autonomously; otherwise, it is indicated to stakeholders that they require amendment. Thus, the contributions of this subchapter are the following:

1. A service-oriented methodology that describes application requirements through multiple layers of SLOs. This enables fine-grained control of the overall system in multi-tenant and multi-vendor scenarios.

2. A diffusion mechanism that propagates high-level SLOs into lower-level ones. To fulfill the associated high-level SLOs, the algorithm defines adequate performance ranges for lower-level SLOs. Further, it identifies parameters (if they exist) that are able to control lower-level SLOs.

3. A conflict identification algorithm for high-level SLOs based on diffused lower-level SLOs. The algorithm is able to resolve conflicts (if they can be solved autonomously), and otherwise alert stakeholders.

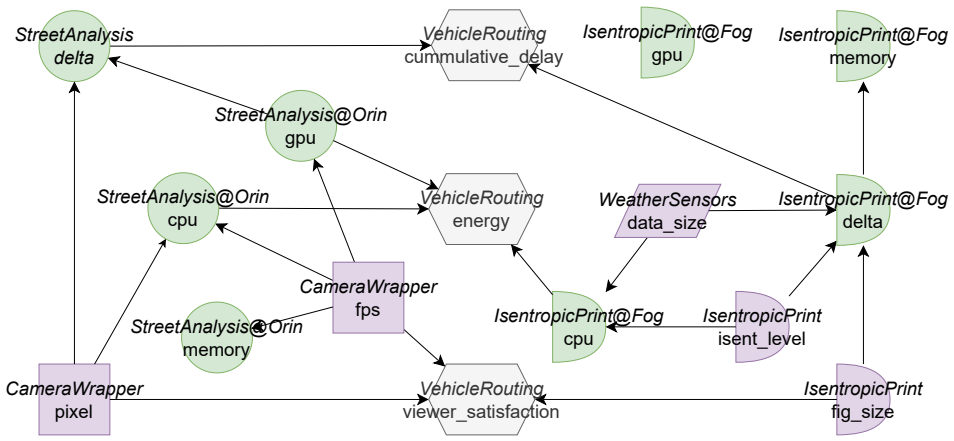


Figure 5.6: Combined BN for a microservice pipeline that consists of the following evaluate services: *VehicleRouting* (yellow center), *CameraWrapper* and *StreetAnalysis* (left), and *WeatherSensors* and *IsentropicPrint* (right)

5.2.2 Preliminaries

This section provides an overview of background knowledge on Bayesian networks and how these can be used to specify SLOs. Furthermore, it contains related work that applies SLOs and Bayesian networks for describing system requirements.

Background

Bayesian Networks (BNs), as introduced in Section 5.1.2, can be used to model relations in real-world processes: numerous works (e.g., [YKAQ22, OSF22, Tog22]) train BNs from historical observations (i.e., metrics) to model the probabilities of different system states. Thus, BNs can answer how likely it is to observe a certain variable assignment, e.g., a system runtime state, given historical observations.

Notable, in this subchapter, we will use BNs to express the dependencies between microservices; see Figure 5.6 for an example graph that is trained in Section 5.2.4. By using the variable relations in the graph, we can evaluate SLO fulfillment as presented in Section 5.1.2. To train BNs from microservice logs, we use `BNL` – a customized algorithm for Bayesian Network Learning (BNL) – that was introduced in Section 4.2.1.

Given a BN, such as Figure 5.6, we note two fundamental properties that will be exploited in this subchapter: (1) any variable (v) that describes a high-level SLO is a leaf node, i.e., it has only incoming edges, otherwise, v's child (or grandchild) would be constrained; since BNs are acyclic, there is always a leaf. Hence, edges in BNs point toward the high-level SLO, which means that fulfilling them is a consequence of maintaining all parent variables in a desired range – these are low-level SLOs. Further, (2) parameters are root nodes, i.e., without incoming edges, because they are conditionally independent

of other variables; if there were some, actively setting a parameter would remove any parent edge. The diffusion algorithm in Section 5.2.3 will build upon these properties.

Related Work

We identified two main areas of related work that intersect with this subchapter: (1) SLO-aware service description to continuously ensure system requirements, and (2) modeling systems as large-scale BNs to estimate how changes propagate or can be countered.

SLO-Aware Service Description Pusztai et al. [PMP+21a, PNM+22, NMP+20] provide *next-level SLO* descriptions, i.e., they are composed of multiple variable thresholds; hence, SLOs can reflect more complex conditions. Their central contribution – an edge-based workload scheduler – is similar to Guan and Boukerche [GB22], which presents QoS-aware processing methods through different AI methods, though BNs were not discussed. To ensure latency SLOs, Seo et al. [SCK+21] provide a dynamic decomposition of ML tasks into subtasks; however, SLOs were not diffused further. Cao [Cao23] outlines a research agenda for an SLO-oriented management layer for cloud-edge infrastructure; Cardelli et al. [CGGN+18] design an autonomous elasticity mechanism to ensure QoS in cloud-edge service chains. The authors in [PD23] discuss the importance of controlling distributed systems with *DeepSLOs*, i.e., such that span multiple infrastructure layers; their vision, however, was not implemented yet, neither for [WOK17].

Given these works, we summarize that SLOs are the state-of-the-art solution to specify requirements for cloud computing; nevertheless, there is an ongoing translation of SLOs from the cloud to the edge. Although authors like [GB22] and [SCK+21] recognize the importance of AI to ensure edge-based SLOs, none of the presented would further diffuse high-level SLOs to identify respective lower-level SLOs.

Large-scale Bayesian Network Modelling Yazdi et al. [YKAQ22] presented a dynamic BN to assess the resilience of a pipeline system – providing insights under which conditions QoS can be assured. Extending to compound systems, Chen et al. [CQH19] provided a dynamic causality graph called *CauseInfer* that pinpoints issues during runtime. *CauseInfer* uses a two-tier mechanism to split a system into device and service layers. Wang et al. [WWC+21] transformed a dynamic fault tree into a BN to trace fault propagation within a vehicle network. This could infer fault probabilities under hypothetical setups. BNL is still an actively developing field: Kitson et al. [KCG+23] provide a comprehensive overview of techniques and algorithms that create accurate causal models, whereas Vowels et al. [VCB21] provides a survey on that topic.

Given these works, we conclude that numerous works focus on training accurate BNs from observations; the use cases behind them are manifold. Although most of them apply the BN to extract some sort of knowledge, none of them used its conditional dependencies to infer how target states can be assured through lower-level requirements.

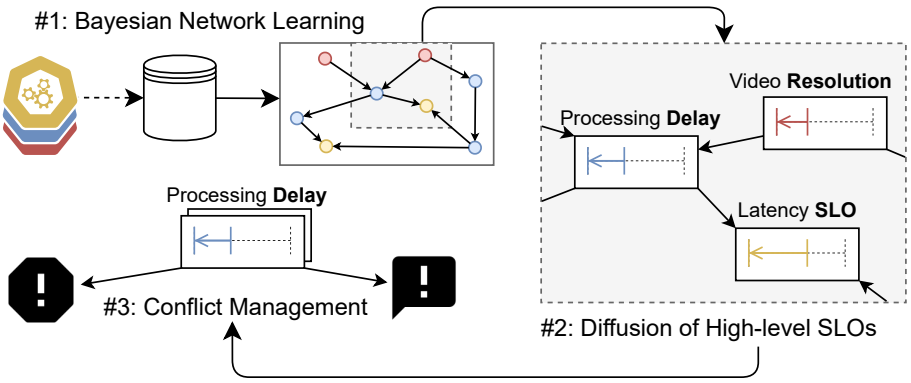


Figure 5.7: 3-Step methodology for ensuring high-level SLOs through diffusion

5.2.3 Methodology

In this subchapter, we present a set of research questions. Then, we illustrate a 3-step methodology that ensures high-level requirements by disseminating them into lower-level subcomponents; it first trains a BN for a service composition or pipeline, then diffuses low-level SLOs and parameter assignments, and lastly indicates and resolves conflicts within low-level SLOs and parameters. Figure 5.7 provides an overview of this methodology; the sequential steps are embedded into the respective subsections 5.2.3 to 5.2.3.

For all following algorithms, Table 5.6 presents a summary of variable notations used in this methodology section.

Research Questions

In the following, we describe three research questions extracted from the introduction, each accompanied by a motivating description. These questions will guide both the methodology as well as its evaluation

***RQ-1**) How can high-level SLOs be translated to lower-level objectives?* Fulfillment of high-level SLOs emerges from a wide equilibrium among system components; this can be ensured by maintaining sub-processes (or components) under desirable conditions. However, to the best of our knowledge, there exist no mechanisms that translate stakeholders' high-level SLOs into lower-level SLOs. As an answer to that, our methodology infers low-level SLOs by leveraging in-depth knowledge about system dynamics.

***RQ-2**) How restrictive should low-level SLOs be?* The more hierarchical and dense a list of SLOs becomes, the less trivial it is what values a low-level SLO should assume to fulfill high-level ones. In reality, predicting the behavior of complex systems will not yield a single possible outcome, but a probabilistic list of states. To that extent, low-level SLOs can hardly be expressed in "black-or-white logic", but the question is how to decide if a low-level state is desirable or not.

Table 5.6: Frequently used variable notations

Notation	Meaning
s	An individual microservice
M_s	Multidimensional metrics describing s state
M	Wrapper for all metrics in the application
D	Training data set joint for all services
G	Bayesian network graph trained from D
Q	List of all high-level SLOs
q	An individual high-level SLO $q \in Q$
p	The parent node of another variable (e.g., q)
p_p	A grandparent node, i.e., parent of p
S_{hl}	List of desired states to fulfill SLOs
hl	A state of a high-level SLO variable
ll	A state of a lower-level parent variable
ll_q	Total probability of fulfilling q with $p = ll$
X	Dictionary to store ll_q according to ll
t	Probability threshold for including a state ll
λ	Hyperparameter to customize acceptance range
L	List of raw low-level SLOs and parameters
v	A random variable in G, might be q, p, etc
L_v	Duplicate constraints for v in L
k	Intersection between multiple constraints
A	List of constraints without duplicates (easy)
B	List of constraints that presented minor conflicts
U	List of low-level SLOs and assignments (final)
C	List of major conflicts that were not resolved

***RQ-3**) Where do conflicts among SLOs occur and how can they potentially be resolved?* When reasoning rationally, it is intuitive that a system cannot fulfill two competing requirements at the same time, e.g., minimizing *energy* while maximizing *customer_satisfaction*. However, with an increasing number of SLOs, stakeholders cannot always maintain an overview; hence, the question is in which part of the system conflicts will actually occur, and to what extent, or under which conditions, they can be resolved autonomously.

Bayesian Network Learning

Given a microservices application, e.g., a sequential processing pipeline, the objective of this first methodology step is to provide a causal understanding of the dependencies between the services. To achieve this, we reveal the relations of different services through BNL – this combines all their variables in one graph. Before that, however, we must collect the necessary training data. Therefore, we observe all applied microservices during runtime and collect multidimensional metrics (M_s) that describe each service's (s) internal processes.

Training data can be collected periodically or in one operation; in any case, the data from all microservices is combined within D. Notice, that metrics from different services must be captured under equal configurations, e.g., if a pipeline contains two sequential microservices *CameraWrapper* → *StreetAnalysis*, the streaming data produced by *Cam-*

eraWrapper must be the exact same received and processed by *StreetAnalysis*. This can be assured by (1) capturing both services' metrics at the same time and joining rows over their timestamp, or (2) maintaining comparable conditions for data sets and joining them over interface variables, i.e., such that describe the same transmitted data on both sides, like *video resolution*.

Next, metrics are processed with BNL to turn them into a composite graph; this is reflected by Algo. 5.5 (Lines 1-4), which also provides the wrapper for the overall methodology. While `BNL` is known from Section 5.2.2, `JOIN` combines training data of different services incrementally into one data set (D); feeding D to `BNL` turns it into a graph G. Afterward, G contains the dependencies between service variables and their conditional probabilities of assuming certain variable states.

Algorithm 5.5: Wrapper for the 3 methodology steps

Require: M, Q {Service metrics and high-level SLOs}
Ensure: U, C {Low-level SLOs, params, and conflicts}
1: $L, D \leftarrow \emptyset$
2: **for each** M_s **in** M **do**
3: $\quad D \leftarrow \texttt{JOIN}(D, M_s)$
4: **end for**
5: $G \leftarrow \texttt{BNL}(D)$ { – Step 1}
6: $L \leftarrow \texttt{HLD}(G, Q, \emptyset, \emptyset)$ { – Step 2}
7: $U, C \leftarrow \texttt{CIR}(L)$ { – Step 3}
8: **return** U, C

Diffusion of High-level SLOs

The diffusion requires the BN (G) from the previous step and a list of high-level SLOs (Q) – the shape of individual SLOs is as introduced in Section 5.2.2. In the following, we start traversing G from nodes that represent high-level SLOs and then gradually visit their ancestors (i.e., nodes with an edge pointing to them). To fulfill high-level SLOs, each node is extended with a threshold it must ensure; if it is a root node, it is called a "parameter", otherwise a "low-level SLO". This is shown in Fig. 5.8, where high-level SLO variables are located on the right (purple); by traversing its parents, the middle column is constrained to certain thresholds, which are reached through the parameter assignments on the left (i.e., grandparents). Visiting variables more than one time can lead to conflicts – this is discussed further in Section 5.2.3.

The diffusion's abstract implementation is shown in Algo. 5.6, which accepts two additional input parameters: a parent node (p) and a list of target states (S_{hl}). However, these are only set in subsequent recursions. The start case (Lines 2-5) is simple: for every high-level SLO (q), find all states $hl \in \texttt{STATES}(G, q)$ that satisfy q's target condition (Line 4). For example, given an SLO $latency \leq 10$, S_{hl} summarizes all known states of

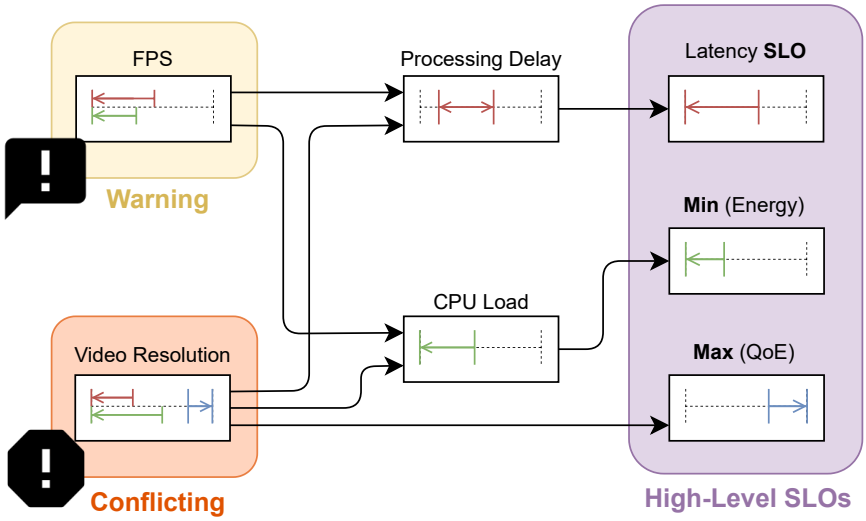


Figure 5.8: Diffusing high-level SLOs into lower-level SLOs and assignments

latency that meet this threshold. Next, in Lines 18-20, find q's parents and constrain each parent node (p_p) by inferring respective low-level states that cause S_{hl}.

Traversing q's parents instantiates multiple recursions (Lines 6-14): for every parent variable (p), find p's states that (likely) fulfill q; in other words, this is the low-level SLO or parameter assignment. This can be inferred from a BN through variable elimination [ZP94] (VE) – an instance of exact Bayesian inference. For every low-level state $ll \in \text{STATES}(G, p)$, we call $\text{VE}(G, q, p = ll)$, which returns the probability of different outcomes (i.e., that a high-level state $q = hl$ occurs) when assigning $p = ll$; in Algo. 5.6, we abbreviate this $P(q = hl \mid p = ll)$ as z. However, if observing $q = hl$ is actually desirable, is determined by hl's occurrence in the list of target states (Line 9). For every state ll, the probability of $p = ll$ causing a desired outcome (i.e., fulfilling q) is summarized (ll_q) and appended to X – a temporary storage to collect these probabilities.

Whether a state ll is included in the low-level SLO, is determined by ll_q – its probability of causing q to be fulfilled. In particular, ll_q must meet the acceptance threshold (t), which is calculated relative to the state with the highest probability (Line 15). The acceptance range can be customized through the hyperparameter $\lambda \in (0, 1]$ – higher λ raises t proportionally; hence, the acceptance range becomes more narrow, meaning fewer states can satisfy it. The accepted states constitute either a low-level SLO or a parameter assignment of p; what follows, is that these constraints are appended to L, as done for recursively visited nodes.

After all high-level SLO variables and their ancestors were visited, L contains all low-level SLOs and parameters that were inferred from G; however, it potentially includes duplicate entries for variables that were visited multiple times. This methodology step addressed **(RQ-1)** by presenting a diffusion mechanism for high-level SLOs; **(RQ-2)** was equally addressed by specifying the acceptance threshold λ. Nevertheless, for both of them, the

Algorithm 5.6: High-level SLO diffusion (HLD)

Require: $G, Q, p, S_{hl}; \lambda$ (global)
Ensure: L {List of low-level SLOs and parameters}
1: **for each** q **in** Q **do**
2: **if** $p = \emptyset \vee S_{hl} = \emptyset$ **then**
3: $p \leftarrow q$
4: $S_{hl} \leftarrow \{hl \mid hl \in \texttt{STATES}(G, q), q(hl) = \texttt{True}\}$
5: **else**
6: **for each** ll **in** $\texttt{STATES}(G, p)$ **do**
7: $ll_q \leftarrow 0$
8: **for each** (hl, z) **in** $\texttt{VE}(G, q, p = ll)$ **do**
9: **if** $hl \in S_{hl}$ **then**
10: $ll_q \leftarrow ll_q + z$
11: **end if**
12: **end for**
13: $X[ll] \leftarrow (ll, ll_q)$
14: **end for**
15: $t \leftarrow max(X) \times \lambda$
16: $S_{hl} \leftarrow \{ll \mid (ll, ll_q) \in X, ll_q \geq t\}$
17: **end if**
18: **for each** p_p **in** $\texttt{PARENTS}(G, p)$ **do**
19: $L \leftarrow \texttt{HLD}(G, q, p_p, S_{hl}) \cup L$
20: **end for**
21: **end for**
22: **return** $L \cup (p, S_{hl})$

evaluation must provide further details on their influence on high-level SLO fulfillment.

Conflict Management

After inferring low-level SLOs and parameter assignments, the entire collection (L) is post-processed to identify and resolve conflicts. Generally, if a variable $v \in G$ was visited n times, then L contains n constraints for v – what differs are the imposed thresholds, each according to another high-level SLO. Recall Fig. 5.8, where the grandparent on the left (i.e., *fps* and *video_res*) were visited two, and respectively, three times; the colored arrows in the variable range indicate from which high-level SLO the constraint originated. The central difference between the two cases is the following: for *fps* there exists a satisfying intersection of its constraints (red $\cap$ green), whereas the constraints of *video_res* are disjoint and not satisfiable. In the following, we resolve the former case as "minor conflict", and indicate the latter as "major conflict".

This behavior is expressed in more detail in Algo. 5.7; in particular, all entries in L are traversed to determine if there are conflicts, and whether they can be resolved. In

Algorithm 5.7: Conflict identification and resolution (CIR)

Require: L {List of low-level SLOs and parameters}
Ensure: U, C {Unique constraints and conflicts}
1: $A, B, C \leftarrow \emptyset$
2: **for each** (v, S_{hl}) **in** L **do**
3: **if** $\texttt{COUNT}(L, v) = 1$ **then**
4: $A \leftarrow (v, S_{hl}) \cup A$
5: **else**
6: $k \leftarrow \texttt{INTER}(\texttt{DUPL}(L, v))$
7: **if** $k \neq \emptyset$ **then**
8: $B \leftarrow (v, k) \cup B$
9: **else**
10: $C \leftarrow v \cup C$
11: **end if**
12: **end if**
13: **end for**
14: **return** $A \cup B, C$

the simplest case, a variable (v) is only present once in L; all variables that fulfill this condition are collected in A (Line 4). Otherwise, for a list of duplicate constraints (L_v), the intersection between all the variables' constraints is calculated according to Eq (5.6).

$$\texttt{INTER}(L_v) = \bigcap_{i,j=1;i\neq j}^{n} L_i \cap L_j \neq \emptyset \tag{5.6}$$

If there exists an intersection (k), this resolves the conflict and k is appended to B (Line 8) – the list of minor conflicts. Otherwise, if the constraints are disjoint, v is appended to C (Line 10) – the list of major conflicts. Finally, Algo. 5.7 returns a list of unique constraints (U), which combines $A \cup B$; C is maintained separately so that these conflicts can be indicated to application developers. Algo. 5.5 returns the same lists.

With the presented methodology, conflicts occur independently of the order in which high-level SLOs are traversed; it is not the case, for example, that the first high-level SLO visiting a variable is prioritized. However, resolving major conflicts would inevitably require some sort of hierarchy among the high-level SLOs, otherwise there cannot be any satisfying variable assignment. Hence, we answered what kinds of conflicts can be resolved **(RQ-3)**; the evaluation will provide further details on where conflicts actually occur. This concluded the presented methodology, which started by training a BN from metrics, inferring low-level SLOs, and finally, in this subsection, resolving conflicts as far as possible, or at least indicating them to the application developer.

Table 5.7: Microsevices available for evaluation

ID	type	param / var	host
TrafficSensors [Wam23]	Producer	1 / 1	*Xavier*
HistoricDB	Producer	2 / 2	*Server*
CameraWrapper[SMDD23]	Producer	2 / 2	*Nano*
WeatherSensors [MAM+24]	Producer	1 / 1	*Xavier*
AnomalyDetection [Wam23]	Worker	0 / 5	*Fog*
HistoricProvision	Worker	2 / 7	*Server*
StreetAnalysis [SPDD24c]	Worker	0 / 4	*Orin*
PrivacyTransform [SMDD23]	Worker	0 / 6	*Orin*
IsentropicPrint [MAM+24]	Worker	2 / 6	*Fog*
TrafficPrediction	Consumer	0 / 2	*Fog*
VehicleRouting	Consumer	0 / 3	*Orin*
LiveMonitoring	Consumer	0 / 3	*Server*

5.2.4 Evaluation

To evaluate the presented ideas, we focus on the individual steps of the methodology and highlight whether the outcome fulfills the research questions. For this, we first outline how the evaluation scenarios were set up and how the methodology was implemented; then we present the results of our experiments and discuss their implications.

Evaluation Scenarios

We evaluate our methodology multiple times under different scenarios; each scenario consists of a microservices application, where individual services are chained together to form a composite pipeline. Please refer to Table 5.7 for a list of all microservices. The presented services are categorized into three types: (1) *producer* services provide sensor data, (2) *worker* services run data processing, and (3) *consumer* services face clients and determine how the pipeline is perceived; hence, stakeholders would place high-level SLOs for *consumers*.

The internal state of each service is described by a set of variables, which can be collected and analyzed through metrics. Some variables can actively be set by the stakeholder to change the resulting service; we call those variables “parameters". Each service in Table 5.7 features a column that specifies the ratio between parameters and variables. For *IsentropicPrint*, the param/var ratio 2/5 indicates that it features 5 variables, of which 2 are parameters. The last table column specifies at which host[7]the services are executed; please refer to [SPDD23] for additional information on device capabilities.

Hosting devices have direct implications on SLO fulfillment, for example, due to heterogeneous hardware capabilities [SPDD24b]. Among the specified hosts, some devices are more restricted than others; as an example, *Server* dwarfs all Jetson devices (i.e., *Nano*, *Orin*, and *Xavier*). Nevertheless, in this subchapter, we assume that deployments of individual services are predetermined. Consequently, this also defines the networking

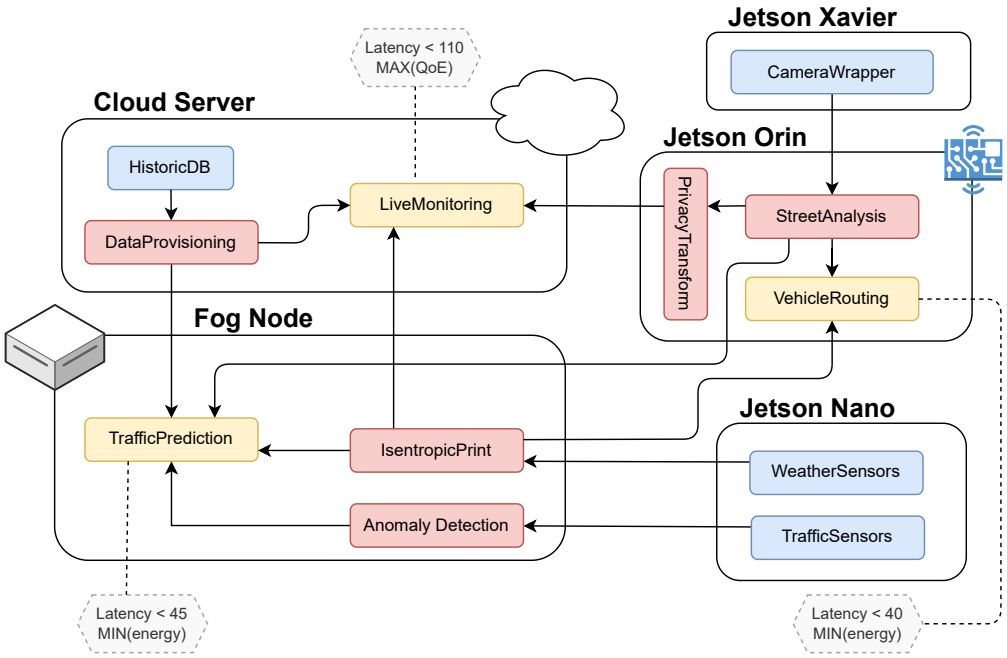


Figure 5.9: Logical microservice architecture with the respective hosting devices

delay between services, which in turn, affects the overall execution time for service pipelines distributed over multiple hosts.

In particular, Figure 5.9 shows the logical distribution between services and hosts, i.e., where individual services are deployed. Microservices are connected alongside the arrows, where data flows in the pointing direction. This creates pipelines from the producers (blue), over the workers (red), to the consumers (yellow). For instance, using *VehicleRouting* and all its parent nodes, it is possible to assemble a smart city application that consumes road conditions to reroute traffic.

In the following, the objective is to diffuse the high-level SLOs for each consumer application and its dependent services. The grey hexagons in Figure 5.9 represent high-level SLOs that stakeholders specified for every consumer service. For instance, to evaluate *LiveMonitoring*, its high-level SLOs are diffused over parent services it depends upon: *IsentropicPrint*, *WeatherSensors*, *PrivacyTransform*, *StreetAnalysis*, and *CameraWrapper*. Thus, we provide evidence of how these services contribute to high-level SLO.

Implementation

We provide a Python-based prototype[8]that implements all aspects of our methodology; apart from that, the repository contains all microservices used to generate BNL training data. Notice that, as depicted in Table 5.7, microservices were adopted from existing research as far as possible. As discussed in Section 5.2.2, the applied BNL algorithm also originates from previous work; noteworthy, for this subchapter, we implemented the

Table 5.8: SLOs and parameter thresholds inferred for *VehicleRouting*

Microservice	Variable	States	SLO / Param
VehicleRouting	cumm_delay	$\leq$ 45 ms	High-level
	energy	$\leq$ 19 W	
	viewer_sat	————	
StreetAnalysis	delta	$\leq$ 35 ms	Low-level
StreetAnalysis	cpu (Orin)	$\leq$ 21 %	
StreetAnalysis	gpu (Orin)	$\leq$ 40 %	
IsentropicPrint	delta	$\leq$ 37 ms	
IsentropicPrint	cpu (Fog)	$\leq$ 17 %	
CameraWrapper	pixel	= 480 p	Parameter
CameraWrapper	fps	= 15 f	
IsentropicPrint	fig_size	$\leq$ 50 p	
IsentropicPrint	isent_level	$\leq$ 200 k	
WeatherSensors	data_size	$\leq$ 30 pi	

algorithm with pgmpy [AT23] – a python framework. To ensure that the trained BN models all applied microservices precisely, the services were configured to evaluate all possible parameter permutations during runtime. While this presents a limitation to the applicability of our methodology, it can be circumvented with alternative approaches, e.g., interpolating between empirically visited configurations [SPDD24b].

Metrics created by each service are collected in CSV files: 80% are used for BNL, whereas the remaining 20% are retained for evaluation purposes. For each application, we first use the training set to train a BN, which is then used to execute our methodology. Afterward, any resulting SLO fulfillment was measured for the test data set; the scripts to create results, images, or tables are all contained in the repository.

Results

In the following, we address the three research questions posed in Section 5.2.3. For each question, we explain how it was evaluated, and then discuss the respective results.

SLO Diffusion (RQ-1) A consequence of successfully translating high-level SLOs to low-level ones would be to find system configurations (i.e., parameter assignments) that fulfill high-level SLOs; hence, we will analyze the resulting SLO fulfillment as an indicator for a correct diffusion. To that extent, we diffuse the respective high-level SLOs over each consumer service and infer low-level SLOs and parameter assignments. We configure the system according to the inferred constraints and analyze whether this could control SLO fulfillment.

[8]Prototype artifact available at GitHub, accessed Apr 10th 2024

Table 5.9: High-level SLO fulfillment for all three microservice applications

Microservice	High-level SLO	% Min	% Fulfill	% Max
VehicleRouting	cumm_delay ≤ 45	0.00	0.94	1.00
	min(energy)	0.53	0.99	1.00
TrafficPrediction	cumm_delay ≤ 40	0.00	0.83	0.90
LiveMonitoring	cumm_delay ≤ 110	0.13	0.93	1.00
	max(viewer_sat.)	0.00	1.00	1.00

The first step is to train a BN for each application; as an example, Figure 5.6 shows the BN for *VehicleRouting* and all microservices it depends on. Grey nodes reflect high-level SLOs, green ones low-level SLOs, and purple nodes parameters; each service also features a unique symbol. For example, the fulfillment of the central *energy* SLO is dependent on the variables that have an edge directed to it, i.e., the *gpu* of *StreetAnalysis*, and the *cpu* from both *StreetAnalysis* and *IsentropicPrint*. Apart from them, there exist nodes that do not impact high-level SLOs (i.e., they have no directed path to grey nodes), which will not be traversed by Algo 5.6.

The resulting constraints are shown in Table 5.8, which contains all low-level SLOs and parameter values that were diffused from the high-level SLO; notice how *viewer_satisfaction* was not constrained in this scenario. Given the high-level SLOs (i.e., first two rows), the low-level SLOs (i.e., second part) present indicators for preferable variable distributions, which are best assured by assigning parameters as specified. Parameters such as *pixel* and *fps* are assigned to one value, whereas the latter three can assume arbitrary values in a range – each value supposedly fulfills low-level SLOs to a degree $> \lambda$. For instance, any $data\_size \leq 30$ causes *cpu* and *delta* (check dependencies from Figure 5.6) to stay in bounds, while keeping *energy* at 19W – the lowest possible assignment.

For all three applications, we configured the parameters according to the inferred thresholds and evaluated the SLO fulfillment; the respective results are contained in Table 5.9. The maximum (or minimum) values reflect possible values from alternative parameter combinations (i.e., permutations); orange cells indicate cases where our methodology could not infer assignments that maximize high-level SLO fulfillment. These discrepancies occur either due to (1) flexible boundaries, e.g., $fig\_size = 50$ is an acceptable assignment $> \lambda$, although *cumm_delay* would be more likely fulfilled with $fig\_size = 10$, or (2) conflicts within high-level SLOs, e.g., *LiveMonitoring* cannot ensure both maximum *viewer_satisfaction* and $cumm\_delay \leq 110$ for 100 % of observations. Nevertheless, we showed that our approach can reach SLO fulfillment of up to 100 %; the lower bound here was given by the *cumm_delay* SLO of *TrafficPrediction*, which reached 83 %.

Acceptance Range (RQ-2) The acceptance range (λ) determines the degree of freedom for low-level SLOs and parameter assignments. A narrow margin promises less tolerance for SLO violations but at the same time risks SLO conflicts due to disjoint

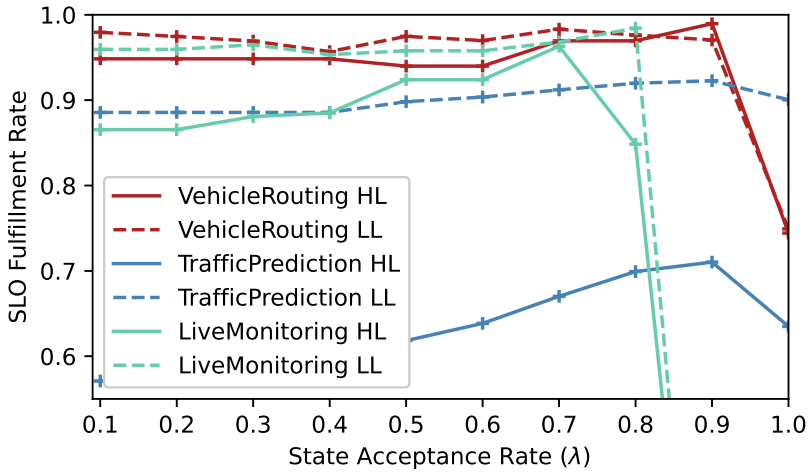


Figure 5.10: SLO fulfillment (high/low-level) of different applications and λs

inference results. To answer what λ is optimal for each application, we vary λ and highlight its effect on high-level and low-level SLO fulfillment.

We apply our methodology with $\lambda \in \{0.1, 0.2, ..., 1.0\}$ and collect the resulting parameter assignments and low-level SLOs. Then, we configure the system according to these constraints and evaluate *all* SLOs: Figure 5.10 visualizes both the high-level and the low-level SLO fulfillment (dashed or solid lines) for different λ values. The SLO fulfillment (y-axis) is calculated as the average of all microservices included per application, e.g., *VehicleRouting* and all its parents.

We observe, that low-level SLOs are always fulfilled to a higher degree than high-level SLOs, which supports the claim that high-level fulfillment is a consequence. Further, increasing λ had a positive effect on the SLO fulfillment (transition from 0.1 to 0.7); however, as the acceptance range becomes too narrow, *LiveMonitoring* runs into SLO conflicts, indicated by fulfillment $= 0$. The optimal λ was different for each application; hence, it needs a dynamic mechanism that maximizes λ without risking conflicts.

Conflicts (and Resolution) (RQ-3) The missing piece for this RQ is to answer where in the BN conflicts actually occur. To that extent, we prepare different combinations of high-level SLOs, provide them to the diffusion algorithm, and analyze for each application whether conflicts occur, and where they occur. Still, whenever possible, conflicts should be resolved automatically, otherwise indicated to stakeholders.

The DAG for *LiveMonitor* equals in large parts the one of *VehicleRouting* in Figure 5.6, which is why we reuse it for the following explanations. To show how and when conflicts occur, we focus our evaluation on *LiveMonitor*: we provide three high-level SLOs, various thresholds for each of them, and combine them as depicted in Table 5.10. The

Table 5.10: Conflicts among high-level SLOs for *LiveMonitor*

cumm_delay	min(*energy*)	max(*customer_sat*)	both
$\leq$ 120 ms	✓	✓	↯{*fps*}
$\leq$ 100 ms	↯{*pixel*}	✓	↯{$\wedge \cup$ *pixel*}
$\leq$ 50 ms	↯{$\wedge \cup$ *fps*}	↯{*gpu, pixel, fps*}	↯{$\wedge \cup$ *gpu*}
$\leq$ 40 ms	↯{$\wedge \cup$ *batch*}	↯{$\wedge$}	↯{$\wedge \cup$ *fig_size*}
$\leq$ 25 ms	↯{$\wedge$}	↯{$\wedge \cup$ *cache_db*}	↯{$\wedge \cup$ *cache_db*}

cumm_delay is always included in the diffusion but combined either with min(*energy*), max(*customer_sat*), or both of them.

Depending on the combinations of high-level SLOs and the threshold of *cumm_delay* $\leq$ $\{120, 100, 50, 40, 25\}$, different variables start to show conflicts. In particular, Table 5.10 also shows for each combination of high-level SLOs whether it creates any major conflict, which is indicated by a ↯ symbol. While *cumm_delay* ≤ 120 did not produce conflicts with either min(*energy*) or max(*customer_sat*), applying both immediately causes a conflict for *fps*. In the rows below, smaller *cumm_delay* gradually leads to more conflicts; $\wedge$ indicates that all conflicts from the above line are propagated, hence, ≤ 25 and min(*energy*) led to three conflicting variables. Thus, conflicts can be identified prior to runtime, which is useful to indicate what type of high-level SLOs can be combined.

Limitations

When diffusing high-level SLOs, the complexity of Algo. 5.6 is dominated by the number of `STATES` of high-level SLO variables (h) and their ancestors (l), leading to a complexity of $\mathcal{O}((h \times l \times Q)^m)$. Hence, this approach works well for variables that have few discrete states, or continuous variables that are binned into a low number of bins; how much precision this sacrifices depends on the use case. Apart from that, the complexity is determined by the depth of ancestors (m).

Various optimizations could be applied to Algo. 5.6, one of them would be to “fold up" longer subtrees in the BN that are single-parented, i.e., do not have other parent nodes. This means, that none of them would have to be extended with a low-level SLO, except for the root node and its direct children (not grandchildren); thus, the list of SLOs could be simplified. However, this condition did not occur in the evaluation, which shows that we must also aim for more complex use cases to improve our methodology further.

Lastly, the algorithm puts a lot of emphasis on the quality of the BN: if G does not accurately reflect reality, e.g., edges are missing or pointing in the wrong direction, the outcome of the algorithm will deviate. Although BN quality was not the focus of this subchapter, the evaluation indicated that BNs are a bottleneck for the methodology. In particular, the applied techniques often fluctuate regarding the direction of edges. These minor issues can prove fatal for the results of the algorithm, hence, we tried to pin respective edges according to expert knowledge to create a stable evaluation environment.

Nevertheless, the results of different BNL techniques and their impact on high-level SLO fulfillment must be the focus of future work.

5.2.5 Summary

This subchapter presented a diffusion mechanism that translates stakeholders' high-level SLOs into lower-level constraints. For a composition of microservices, it becomes thus clear how individual sub-processes contribute to high-level objectives, and how these must be configured to ensure SLO fulfillment. In particular, we presented a 3-step methodology that infers this knowledge from a Bayesian network, while resolving potential conflicts among competing SLOs as far as possible. The evaluation showed how multiple high-level SLOs, each targeting different QoS or QoE aspects, can be diffused over four different microservice pipelines. For each application, the inferred constraints could exert direct control over high-level SLO fulfillment, which was consequently satisfied between 83 % to 100 % of observations. Further, we could show the impact that the "restrictiveness" of low-level SLO assignments has on higher-level SLOs and how conflicts that occur can endanger these values. In that regard, future work will use these insights to improve the methodology further.

5.3 SLO-Aware Task Offloading

In the context of autonomous vehicles (AVs), offloading is essential for guaranteeing the execution of perception tasks, e.g., mobile mapping or object detection. While existing work on offloading focused extensively on minimizing inter-vehicle networking latency, vehicle platoons (e.g., heavy-duty transport) present numerous other objectives, such as energy efficiency or data quality. To optimize these Service Level Objectives (SLOs) during operation, this subchapter presents a purely Vehicle-to-Vehicle approach (V2V) for collaborative services offloading within a vehicle platoon. By training and using a Bayesian Network (BN), services can proactively decide to offload whenever this promises to improve platoon-wide SLO fulfillment; therefore, vehicles estimate how both sides would be impacted by offloading a service. In particular, this considers resource heterogeneity within the platoon to avoid overloading more restricted devices. We evaluate our approach in a physical setup, where vehicles in a platoon continuously (i.e., every 500 ms) interpret the SLOs of three perception services. Our probabilistic, predictive method shows promising results in handling large AV platoons; within seconds, it detects and resolves SLO violations through offloading.

The remainder of this subchapter (see footnote 1) is organized as follows: Section 5.3.2 provides an illustrative scenario, Section 5.3.3 describes our framework for SLO-aware offloading, which is evaluated in Section 5.3.4; Section 5.3.5 provides related work. Finally, Section 5.3.6 summarizes the subchapter with an outlook on future work.

5.3.1 Introduction

The swift evolution of Autonomous Vehicles (AVs) promises a disruptive impact [MWYY20] for future transportation. Despite AV solutions claim considerable benefits, such as rapid green transition and traffic flow improvement [KBJ+20], the execution of AV-enabling services, such as perception, path planning, and control [LMYDM22] pose ambitious processing requirements. Here, optimal allocation and execution of workloads highly depend on AVs' constrained computation capabilities and the supporting infrastructure's network bandwidth. A lack of these guarantees can cause delays in real-time perception and decision-making, leading to potentially harmful consequences.

Services offloading [FSL+23] aims at mitigating these risks, for example, by minimizing computation latency between neighboring vehicles through Vehicle-to-Vehicle (V2V) or Vehicle-to-Infrastructure (V2I) transmission. However, collaborative AV scenarios commonly have higher-level objectives besides latency. For instance, consider AV platoons for public or heavy-duty transport, where the system providers want to minimize costs or energy consumption. We define these requirements as Service Level Objectives (SLOs) – a term from software engineering. The concept of SLOs is wide enough to define any high- or low-level objective that a management framework can enforce [MSRD23, SPDD24b] by elastically adapting hardware or software. SLO-awareness also offers promising scenarios [QLZH18] for V2V offloading; however, its adoption remains limited, highlighting the gap for more intelligent offloading mechanisms [GLL20].

This subchapter, therefore, aims to ensure SLOs by incorporating them into the offloading mechanism – we call this "SLO-aware task offloading". Our motivation stems from two central objectives: (1) we want to ensure that vehicles fulfill the SLOs of their local services; if SLOs are violated, this might be resolved by offloading services, and simultaneously, (2) offloaded tasks must not jeopardize the SLO fulfillment of existing services at the target host. This goal implies solving a combinatorial problem, i.e., the optimal assignment of n services to m vehicles; this problem is NP-hard, hence practically intractable. A solution could be to decompose the problem so that AVs make decentralized offloading decisions. However, training an offloading model for every AV separately would introduce a considerable overhead. Furthermore, we would miss the chance to combine knowledge from multiple AVs, which promises a more profound understanding. For these reasons, we envision a method that trains a decision model within an AV but simultaneously integrates knowledge from other AVs.

In this subchapter, we present a modular, collaborative framework for autonomous SLO interpretation and service offloading. We consider collaborative offloading approaches using "decentralized" sensory data [GLZ+21]. Individual services continuously observe their processing to understand the extent to which SLOs can be fulfilled on different processing hardware; this knowledge is encoded in an SLO interpretation (SLO-I) model. These models are updated by a mutable platoon leader according to AVs' observations and then broadcast to other AVs. Given the SLO-I model, individual services predict how offloading impacts global SLO fulfillment. The contributions of this subchapter are:

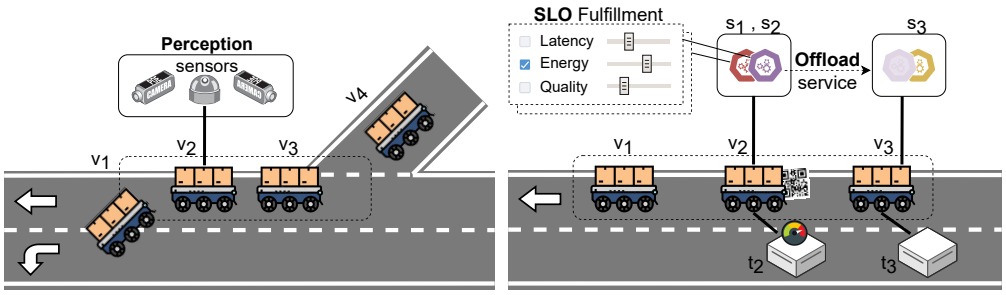
1. An SLO-aware offloading mechanism based on Bayesian networks that dynamically estimates the hardware implications of multiple competing services to find a satisfying assignment. Thus, it is possible to optimize the SLO fulfillment by shifting computation within a composable vehicle platoon.

2. A collaborative training strategy that continuously exchanges model updates between edge devices while adjusting the training frequency according to agents' local SLO prediction errors. Thus, service agents improve their SLO interpretation whenever the system does not behave as predicted.

3. A modular framework for collaborative service offloading that can be extended with custom processing services and respective SLOs. Thus, other service managers can plug their own service implementation into the framework, which itself can be installed on arbitrary edge device types.

5.3.2 Illustrative Scenario

Here, we consider a platoon of vehicles for heavy-duty transportation. Depending on the trajectories of platoon members, individual vehicles can join or leave the platoon at specific intersections, such as ramps. One of the platoon members is elected as the leader, either apriori or dynamically. In this subchapter, we focus on V2V offloading, as V2I infrastructures could be impractical [FSL+23] or add delays [CCL+20].



(a) Maneuvers of the platoon (b) Offloading within the platoon

Figure 5.11: Composite vehicle platoons offload computations according to SLO fulfillment; if service s_2's SLOs are violated at host v_2, it searches for alternatives, such as v_3

As shown in Fig. 5.11a, n vehicles are clustered into a platoon $P = \{v_1, ..., v_n\}$. We represent each vehicle through the pair $v = \langle id, t \rangle$, where $v.t$ specifies the type of processing device embedded. Additionally, each vehicle is equipped with numerous sensors and perception services, for instance, in Fig. 5.11b, vehicle v_2 runs two services, i.e., mapping its surroundings through Lidar (s_1) and detecting objects on the road through computer vision (s_2). Given that v_2 has a QR code attached to its rear, v_3 follows its predecessor by scanning for QR codes (s_3). We define a service through $s = \langle type, Q, C \rangle$, which reflects the *type* of perception service, e.g., Lidar or CV; Q specifies a set of processing SLOs, and C a list of service constraints, e.g., CV should operate at $fps = 15$. These specifications ensure safe operations when vehicles must respond to dynamic conditions.

Depending on services' resource demand, vehicles may lack the processing capabilities to fulfill their SLOs, which impacts how (i.e., latency and quality) a vehicle perceives its environment. For instance, v_2 might employ a weaker processing device ($v_2.t$); however, v_3's resources are less utilized, so v_2 might offload one of its services to v_3. Therefore, v_1 must now decide (1) which service, i.e., s_1 or s_2, should best be offloaded to v_3, (2) whether this improves SLO fulfillment of remaining services at v_2, and (3) if offloading could impact s_3 negatively. In the context of this subchapter, we focus on higher-level requirements, i.e., leaving out networking latencies for transferring input data and results under the assumption of high network throughput between nearby vehicles.

5.3.3 Methodology

In the following, we present our modular framework for SLO-aware task offloading in composable vehicle platoons. This means, continuously observing service executions to collect insights, interpreting these insights through collaborative training, and making offloading decisions. Fig. 5.12 provides a high-level overview of these processes, which are explained in more detail in subsections 5.3.3 to 5.3.3.

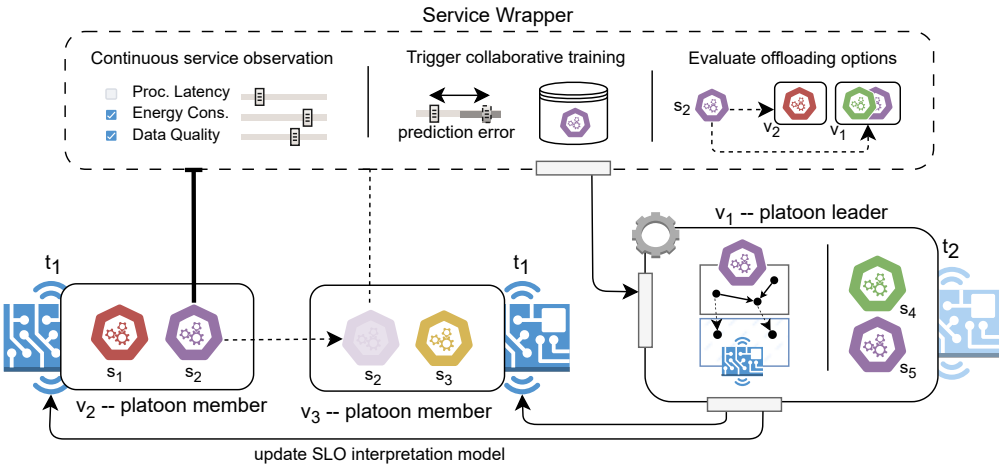


Figure 5.12: Framework for collaborative offloading: inaccurate SLO predictions trigger retraining of SLO interpretation models; services use these models to evaluate alternative hosts according to their expected hardware utilization and SLO fulfillment

Service Observation

The first building block of our approach is observing a service, i.e., continuously monitoring and interpreting its SLO fulfillment. Observation requires interpreting service metrics parallel to service execution, as part of the service wrapper in Fig. 5.12. Perception tasks, such as those executed by autonomous vehicles, usually work iteratively; hence, service metrics are also interpreted step by step. In Algo. 5.8, it is depicted how metrics ($D_{s,v}$) from executing a service (s) on a vehicle (v) are interpreted: for a set of SLOs (Q), the percentage of metrics (ϕ)[9]that fulfill these conditions is determined as shown in Eq. (5.7); then, ϕ is appended to the sliding window W_ϕ. To avoid overhasty decisions based on sporadic SLO violations, the length of the sliding window ($|W_\phi|$) can be customized.

$$\phi(Q) = \frac{\sum_{i=1}^{|Q|} \phi(q_i)}{|Q|} \tag{5.7a}$$

$$\phi(q_i) = \phi(q_i, m, v | \forall m \in D_{q_i}^v, v \in V) = \sum_{j=1}^{|D_{q_i}^v|} \frac{\phi(m_j, q_i)}{|D^v|} \tag{5.7b}$$

$$\text{where } \phi(q_i, m_j) = \begin{cases} 1, & \text{if } m_{j\min}^{q_i} \leq m_j \leq m_{j\max}^{q_i} \\ 0, & \text{otherwise} \end{cases} \tag{5.7c}$$

To understand if a service should be loaded off, we consider both its current SLO fulfillment as well as predictions according to historical observations; for this, we infer the predicted SLO fulfillment (Line 3) using a Bayesian Network (BN). BNs, as presented

[9]We choose the symbol ϕ due to the sound of the letter, i.e., SLO ful-phi-llment

Algorithm 5.8: Continuous SLO Interpretation

Require: D, B, W_ϕ; s, $m_{s,t}$, ρ, ω, γ (global)

1: $\phi_s \leftarrow \phi(s.Q)$
2: $W_\phi \leftarrow W_\phi \cup \phi$
3: $p_\phi \leftarrow \texttt{INFER}(m_{s,t}, s.Q, s.C)$
4: $B \leftarrow B \cup D$
5: $e_r \leftarrow \textbf{abs}(W_\phi - p_\phi) + \texttt{FULL}(B)$
6: **if** $e_r > \rho$ **then**
7: $\quad m_{s,t} \leftarrow \texttt{RETRAIN}(B); B \leftarrow \emptyset$
8: **end if**
9: $e_o \leftarrow \textbf{abs}(W_\phi - p_\phi)) + (1 - W_\phi)$
10: **if** $e_o > \omega$ **then**
11: $\quad v' \leftarrow \texttt{FIND\_OFFLOAD}(s, v)$
12: $\quad$ **if** $v' \neq \emptyset$ **then** $\texttt{OFFLOAD}(s, v')$
13: **end if**

in Section 5.1.2, can answer how likely it is to observe a specific (i.e., SLO fulfilling) state at runtime; hence, we call them SLO interpretation (SLO-I) models. For an SLO-I model m and service s, agents predict SLO fulfillment through $\texttt{INFER}(m, s.Q, s.C)$.

To ensure that predictions remain accurate regardless of variable drifts, increasing prediction errors trigger retraining. As more training data is collected (Line 4), the utilization of the metrics buffer, as shown in Eq. (5.8), indicates that the model becomes outdated, putting additional weight on retraining.

$$\texttt{FULL}(B) = \frac{\sum_{i=1}^{n} 1}{|B|} \tag{5.8}$$

Next, in Line 5, we calculate the evidence to retrain (e_r) as the sum of absolute prediction error and metric buffer utilization. If e_r surpasses the retraining rate (ρ), the metrics buffer is sent to the platoon leader to update the SLO-I model; this is further elaborated in Section 5.3.3. Notice that the buffer size ($|B|$) and ρ can both be customized; for instance, $\rho = 1.0$ would be exceeded if $\texttt{FULL}(B) = 0.8$ and the prediction is off by 0.3.

Model retraining assures that offloading decisions are taken based on accurate assumptions; to that extent, the evidence to load off (e_o) is computed (Line 9) as the sum of absolute SLO violation and prediction error. When e_o surpasses a custom rate ω, and only in this case, does the agent look for a suitable host within the vehicle platoon (Line 11); if there is one, the service will then be offloaded there; this is explained in Section 5.3.3.

Collaborative Training

Retraining of SLO-I models is carried out by the platoon leader, i.e., a distinguished member elected; however, training data is provided by all platoon members. For instance,

recall Fig. 5.12, where s_2 and s_5 are two CV service instances executed on different hosts. Each service collects evidence to retrain (e_r) independently of other instances; once its $e_r > \omega$, the service requests a model update from the platoon leader, providing its local training buffer. Technically, our architecture allows platoon members to update SLO-I models locally; however, limiting the training to the leader improves model consistency over the platoon, plus it isolates the training overhead. Also, to avoid a platoon leader becoming a single point of failure, new leaders can be reelected at any point; for the context of this subchapter, we exclude leader election strategies from the analysis.

Each combination of service and device type is encoded in a unique SLO-I model. Therefore, as soon as the platoon leader (v_1) receives a metric buffer (B_{s,v_2}) from a member (v_2), it first checks v_2's type of processing device ($v_2.t$), e.g., Jetson Orin NX. Next, the leader updates its local SLO-I model ($m_{s,t}$) for service s and device type $v_2.t$; in our example, this means updating the SLO-I model of service $s = CV$ executed on device type $t = NX$. Finally, a new model version $m' = \texttt{PARL}(m, B)$ is created by updating the BN parameters according to recent observations ($B_{s,v}$). Retraining through PARL is limited to updating the conditional probabilities of BN variables; the structure (i.e., variable relations) is left untouched and only supplied through expert knowledge.

After retraining, the updated model (m') is shared within the platoon. For this, the platoon leader broadcasts $m'_{s,t}$ to all members in $\{v \in P \mid v.t = v_2.t\}$, i.e., to all platoon members with the matching device type. Vehicles that received an updated model now substitute the SLO-I models of locally running services. For Fig. 5.12, this would mean that s_2 gets updated, but s_5 not, since v_1 has a device type $v_2.t \neq v_1.t$. Thus, all instances of service s at vehicles with type $v.t = v_2$ interpret their SLO fulfillment according to the new model version.

Service Offloading

Once a service collected sufficient evidence to load off (e_o), like s_2 in Fig. 5.11 & 5.12, the service looks for the best alternative host, which means comparing for each of the other platoon members if global SLO fulfillment would be improved by offloading there. Formally, this is described in Algo. 5.9, which uses the list of platoon members (P), the assignments (A) of which vehicle currently executes which service, and the shared collection (M) of all SLO-I models. In case the platoon does not contain other vehicles (Line 1), the search stops immediately; otherwise, the service predicts (1) the combined SLO fulfillment (ϕ_S) for all services (S_v) executed at vehicle v (Line 4), and (2) how offloading s would change local SLO fulfillment ($\phi_{S'}$) (Line 5). For this, we first estimate the combined hardware demand (CONV_HW) that would emerge from co-locating the services on a target device and then estimate per service if the increased hardware load has an impact on its SLO fulfillment.

Before continuing Algo. 5.9, we briefly explain $\texttt{CONV\_HW}(S, t)$, which predicts the hardware utilization that would result from executing all $s \in S$ at a device of type t. For each service $s \in S$, we use the respective model $m_{s,t} \in M$ to infer its expected hardware

Algorithm 5.9: Evaluating Alternative Host (`FIND_OFFLOAD`)

Require: s, v; P, A, M (global)
Ensure: v' {Optimal vehicle for offloading s from v}
1: **if** $|P| = 1$ **then return** $\emptyset$
2: $S_v \leftarrow \{s_a \mid (s_a, v_a) \in A \mid v_a = v\}$
3: $S'_v \leftarrow S_v \setminus \{s\}$; $\Gamma \leftarrow \emptyset$
4: $\phi_S \leftarrow \texttt{INFER}(M[S_v], S_v.Q, \texttt{CONV\_HW}(S_v, v.t))$
5: $\phi_{S'} \leftarrow \texttt{INFER}(M[S'_v], S'_v.Q, \texttt{CONV\_HW}(S'_v, v.t))$
6: **for each** w **in** $P \setminus \{v\}$ **do**
7: $\quad \Sigma_w \leftarrow \{s_a \mid (s_a, v_a) \in A \mid v_a = w\}$
8: $\quad \Sigma'_w \leftarrow \Sigma_w \cup \{s\}$
9: $\quad \phi_\Sigma \leftarrow \texttt{INFER}(M[\Sigma_w], \Sigma_w.Q, \texttt{CONV\_HW}(\Sigma_w, w.t))$
10: $\quad \phi_{\Sigma'} \leftarrow \texttt{INFER}(M[\Sigma'_w], \Sigma'_w.Q, \texttt{CONV\_HW}(\Sigma'_w, w.t))$
11: $\quad \gamma \leftarrow (\phi_{S'} + \phi_{\Sigma'}) - (\phi_S + \phi_\Sigma)$
12: $\quad \Gamma \leftarrow \Gamma \cup (\gamma, w)$
13: **end for**
14: $\gamma, v' \leftarrow \{(\gamma, w) \in \Gamma, \mathbf{max}(\gamma)\}$
15: **return** v' **if** $\gamma > 0$ **else** $\emptyset$

utilization; in our case, we consider the hardware variables $hw = \{cpu, gpu, memory\}$, but the list can be extended arbitrarily with other monitor variables included in the SLO-I model. This returns a probability distribution (e.g., p_{cpu}) for each variable $\in hw$; afterward, the combined hardware load is calculated as the convolution of the individual loads. Formally, the convolution of two or more random variables (X, Y) with probability density functions $f_X(x)$ and $f_Y(y)$, i.e., the probabilities for each hw variable, is the sum $(Z = X + Y)$ of their individual distributions [Bac89], as shown in Eq. (5.9).

$$f_Z(z) = (f_X * f_Y)(z) = \int_{-\infty}^{\infty} f_X(t) f_Y(z - t)\, dt \tag{5.9}$$

Thus, we obtain the combined hardware utilization, which is supplied as a constraint to `INFER`; this allows estimating how the respective hardware load would impact SLO fulfillment (ϕ_S and $\phi_{S'}$). Alternative approaches to estimating combined load and resulting SLO fulfillment might need to empirically test the service deployment, which is infeasible when decisions must be made quickly.

In the next step, we estimate for each of the other platoon members (w) the SLO fulfillment (ϕ_Σ) of its local services (Σ_w) and how this would be affected ($\phi_{\Sigma'}$) if we would offload s there. This follows the same pattern applied for the source vehicle v: we use the list of services executed at w (Line 7) and their respective SLO-I models to estimate their SLO fulfillment according to the combined hardware load (Lines 9 & 10). The last step is calculating the offloading gain (γ) for each platoon member (w), i.e., whether global SLO fulfillment would be improved by offloading s to w, and then return the best possible vehicle. For this, it first calculates γ (Line 11), which is appended to the collection Γ. In

the final step, it selected the best alternative host among the platoon members (Line 14); however, if not even the best host would improve overall SLO fulfillment, it prefers to keep the current host (Line 15). The outcome is returned to Algo. 5.8, which offloads the service accordingly.

5.3.4 Use Case: Collaborative Vehicle Platoon

Here, we evaluate our methodology for a set of heterogeneous perception services and a composable vehicle platoon. Specifically, we implement a prototype of our framework that addresses the illustrated scenario; afterward, we document the experimental setup, including service implementations and applied processing hardware, then present the experimental results, and critically discuss them.

Implementation

To implement our methodology, we provide a Python-based prototype[10]that follows a clear modular structure for services, their SLOs, and device types. Hence, the framework can be extended with new services as long as they are supported by the underlying edge device. Once the framework is installed[11], services can be started or stopped remotely through HTTP; for running the experiments, we send the respective instructions to different platoon members using Postman flows[12]. To isolate resource consumption, services are executed in individual Python threads. During that time, each service observes its SLO fulfillment as part of its service wrapper (i.e., Algo. 5.8); in the present state, this is done every 500ms, though it can be customized for service types or instances. To avoid interfering with regular service execution, model training and evaluation of alternative service hosts run detached from the main service thread.

Vehicles communicate exclusively over HTTP; the respective connection is established either through a local access point managed by the platoon leader, or through IBSS, i.e., a peer-to-peer network. Training and updating of SLO-I models, or rather their underlying BNs, uses pgmpy [AT23], a Python library for Bayesian Network Learning (BNL). In pgmpy, BNs can be encoded in XML, which each had a size of roughly 10kB in our evaluation; hence, a feasible size to be transmitted and shared within the platoon.

Experimental Setup

To evaluate our prototype in a realistic environment, we implement the scenario illustrated in Section 5.3.2, i.e., perception services are offloaded within a vehicle platoon according to their local SLO fulfillment. We provide three perception services that can be executed on edge devices; Tab. 5.11 provides essential information on these services: *CV* uses Yolov8 to detect objects in a video stream, *LI* processes point clouds from a Lidar sensor

[10]The framework prototype is available at GitHub, accessed on July 14th 2024

[11]Instructions are provided in the following README, accessed on July 14th 2024

[12]Postman is a common tool for sending HTTP requests; Postman flows is a UI extension that allows to specify sequences of requests, e.g., start/stop services

Table 5.11: List of all predefined services that were added to the framework

ID	Service Description	CUDA	Parameters	SLOs
CV	Object Detection with Yolov8 [VM24]	Yes	*pixel*, *fps*	**time**, **energy**, **rate**
LI	Lidar Point Cloud Processing [Dzu20]	Yes	*mode*, *fps*	**time**, **energy**
QR	Detect QR Code w/ OpenCV [ope24]	No	*pixel*, *fps*	**time**, **energy**

Table 5.12: List of all edge devices that were involved in the evaluation

Full Device Name	ID	Price[14]	CPU	RAM	GPU	CUDA
Jetson Orin NX (3)	*NX*	450 €	ARM Cortex 8C	8 GB	Volta 1k	11.4
Jetson Orin AGX	*AGX*	800 €	ARM Cortex 12C	64 GB	Volta 2k	12.2

to map the environment, and *QR* uses OpenCV to detects QR codes in a video. Each service has specific tuning parameters, such as the resolution (*pixel*) and *fps* for *CV* and *QR*; *LI* accepts an additional parameter *mode* to define the point cloud radius.

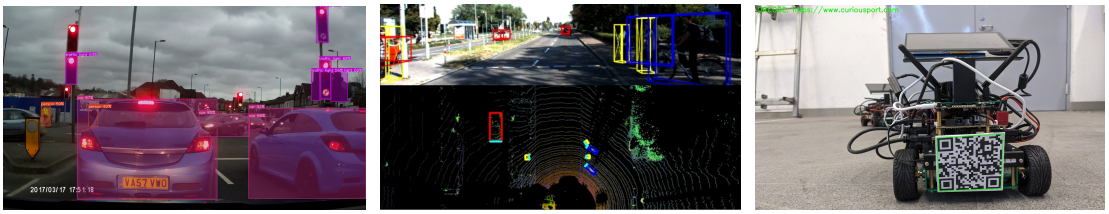
According to our expert knowledge, each service's expected QoS level is specified through a list of SLOs; through heuristic trial and error, the following ones proved useful: we constrain the processing **time** $\leq 1000/fps$, i.e., frames must be processed faster than they come in; the maximum **energy** consumption can be adjusted for individual devices: we put a limit of $\leq 15W$ for regular platoon members and $\leq 25W$ for the platoon leader. Notice, that this considers the vehicle-wide energy consumption over all executed services. According to the video resolution (*pixel*) provided to *CV*, the service uses the respective Yolov8 model size (i.e., v8n, v8s, v8m); however, this affects the number of objects that are detected, which is ensured through the **rate** SLO.

The presented framework is evaluated on two different instances of Nvidia Jetson boards, namely Jetson Orin *NX* and Orin *AGX*, which are described in more detail in Tab. 5.12: the *AGX* is superior in terms of memory and GPU and has a slightly better CPU. While the specific Nvidia CUDA version has minor importance, CUDA itself is crucial to accelerate the *CV* and *LI* services. Each Jetson *NX* is embedded in a Rosmaster R2[13]car – a battery-powered multi-sensory vehicle used for development. To ensure a stable evaluation environment, the service processed either prerecorded videos (*CV* & *QR*) or binary-encoded point clouds (*LI*); Fig. 5.13 shows a demo output for each service.

Results

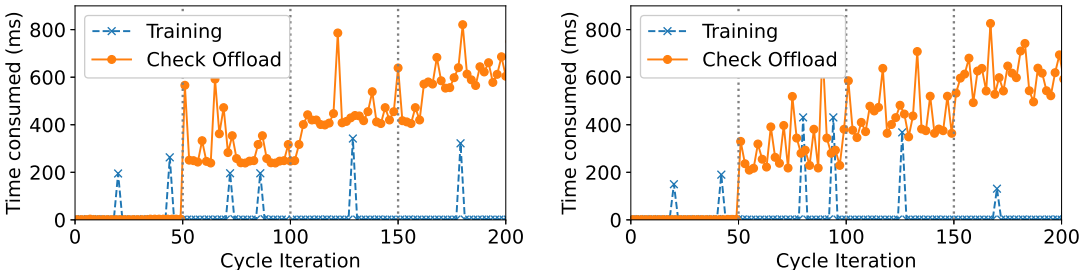
We evaluate the prototype by observing: (1) what is the overhead of continuously interpreting services, and what limitations arise from the platoon size; (2) if the SLO-aware retraining ensure prediction accuracy regardless of unexpected runtime behavior; and (3)

[13]More information about the Rosmaster R2 here, accessed Jul 14th 2024
[14]Prices adopted from sparkfun, accessed Jul 14th 2024



(a) CV (Yolov8) (b) LIdar (SFA3D) (c) QR (OpenCV)

Figure 5.13: Demo output for each service according to the prerecorded input data



(a) Orin *NX* (b) Orin *AGX*

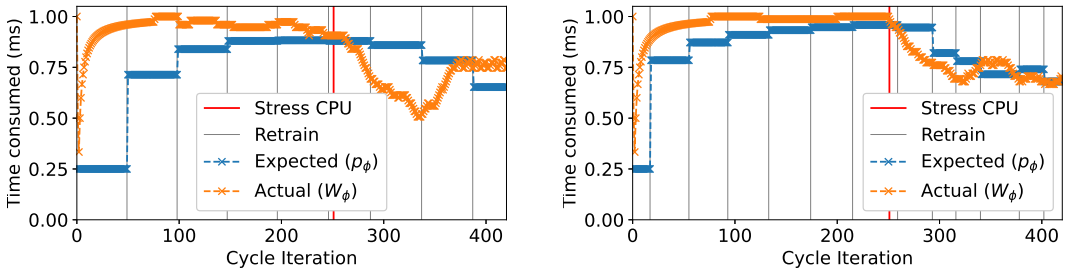
Figure 5.14: Time required to train the SLO-I model and evaluate alternative hosts

if the framework fulfills high-level SLOs within the platoon by offloading computations. We assess these aspects using two base cases and one advanced scenario, all of which involve real workloads and devices:

Scenario 1A An individual vehicle (i.e., *NX* or *AGX*) executes the *QR* service; every 25s, we add a vehicle to its platoon, up to a maximum size of 4 vehicles. Given this, we track the time to execute the service wrapper, i.e., how long it takes to retrain the SLO-I model and evaluate alternative hosts for *QR*.

Fig. 5.14 visualizes the times required to train the SLO-I model or evaluate alternative hosts for offloading; both processes are executed as part of the service wrapper. The wrapper runs every 500ms for a total of 100 seconds, hence, the plot contains 200 wrapper iterations. Vertical grey lines indicate when an additional device is introduced to the platoon, i.e., at 50, 100, and 150 iterations.

Given this, we conclude that the platoon size has a linear impact on the time required to evaluate alternative hosts; the exception is $|P| = 1$, when evaluating other vehicles for offloading is obsolete. For a platoon with $|P| \leq 3$, the entire service wrapper finished mostly in $\leq 500ms$; however, $|P| \geq 4$ starts exceeding $500ms$, which indicates that it would not be possible to interpret the SLO fulfillment every $500ms$. This could be overcome by either structuring the platoon into smaller subgroups or adjusting the evaluation interval.



(a) Without SLO-dependent retraining (b) With SLO-dependent retraining

Figure 5.15: Improved prediction accuracy through SLO-dependent retraining

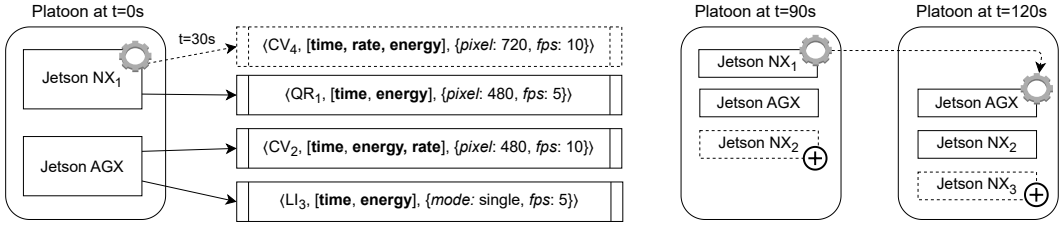


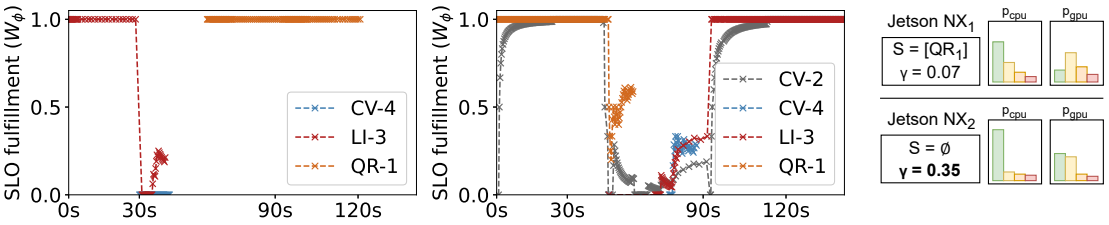
Figure 5.16: Sequential description of Scenario 2: starting services and adjusting platoon

Scenario 1B An individual vehicle (i.e., AGX) runs CV locally; however, the respective SLO-I model was not yet fine-tuned and initial predictions are likely inaccurate. Additionally, variable drifts occur, which we simulate through stress-ng: after $125s$ the CPU load of AGX is stressed 40%. We measure p_ϕ and W_ϕ, and compare our presented training strategy with a static service wrapper.

Fig. 5.15 visualizes for both runs the predicted (p_ϕ) and actual SLO fulfillment (W_ϕ); vertical grey lines indicate when retraining happened, and the red line when the perturbation occurred. Not only does the left side perform fewer retraining, i.e., 8 instead of 12, but more importantly, the right side presents shorter training intervals when the SLO fulfillment is unstable, such as during the period between $x = [250, 350]$. Consequentially, the Mean Squared Error (MSE) was 0.07 on the left and 0.01 on the right side; given that, we conclude that SLO-dependent retaining helped to increase the prediction accuracy for initially inaccurate models or at runtime when perturbations occur.

Scenario 2 Fig. 5.16 provides a sequential description of this scenario: at time $t = 0s$ the platoon $P = \{NX_1, AGX\}$ starts 3 services (i.e., QR_1, CV_2, LI_3); at $t = 30s$ NX_1 starts CV_4; at $t = 90s$ NX_2 joins the platoon, and at $t = 120s$ NX_3 joins, NX_1 leaves the platoon, and leadership is transferred to AGX.

Fig. 5.17 visualizes the SLO fulfillment of all services executed at NX_1 and AGX; at first, all three services (i.e., QR_1, CV_2, LI_3) achieve maximum SLO fulfillment, i.e., $W_\phi = 1.0$. However, as soon as CV_4 is started at $t = 30s$, NX_1 fails to ensure the SLOs for both LI_3 and CV_4. Due to that, NX_1 decides to load off both services to AGX, which in turn,



(a) Orin NX_1 (b) Orin AGX (c) CV_4 at $t = 90s$

Figure 5.17: SLO fulfillment and decision making for constrained services in the platoon

causes AGX to fail most of its services' SLOs. This changes at $t = 55s$, when AGX decides to move one of its services (i.e., *QR-1*) to NX_1, which slightly recovers the SLO fulfillment of the remaining three services. Next, at $t = 90s$, NX_2 joins the platoon, which encourages AGX to offload another service (i.e., CV_4) to NX_2. Here, Fig. 5.17c shows the decision-making of AGX: since NX_1 already executes QR_1, it estimates how adding CV_4 would have a negative impact on QR_1 due to predicted resource shortage; hence, it chooses NX_2, which promises global SLO improvement of $\gamma = 0.35$.

Given this, we conclude that services can react in $\leq 10s$ to local SLO violations, which appears practical for real-time systems. This highlights the impact of co-locating too many services at one edge device and how this can be resolved by adding new vehicles to the platoon. Furthermore, changing the platoon leader at $t = 120$ showed no negative impact on the remaining vehicles – its ongoing computations were shifted to an idle vehicle (i.e., NX_4) that just had joined.

5.3.5 Related Work

We classify existing literature on task offloading for IoV and related scenarios in two main categories: offloading in V2I / V2V scenarios and offloading through Markovian or Bayesian methods. To set the foundation for our contribution, we highlight the strengths and limitations of these approaches.

IoV offloading mechanisms

In the context of V2I task offloading, Xu et al. [XDL+23] provide a neighborhood search algorithm that minimizes costs of task outsourcing, estimated on simulated network traffic. Similarly, Dong et al. [DXK23] provide a multi-task and multi-user offloading mechanism for Mobile Edge Computing (MEC), optimized through a particle swarm. Ant colony optimization (ACO) is another explorative algorithm for optimal pathfinding: Mousa and Hussein [MH22] apply ACO to cluster IoT devices accessed by UAVs; Ma et al. [MSXY22] model the same scenario, but with Mixed-Integer Linear Programming (MILP), closely to Zhang et al. [ZJXZ23]. Related to our use case, Lu et al.[LLSY22] provide a latency-aware V2V/V2I offloading mechanism based on Deep Reinforcement

Learning (DRL). Fan et al. [FSL+23] propose a V2V/V2I offloading tool that decomposes optimization problems with Generalized Benders Decomposition (GBD).

Other authors model offloading scenarios as shortest path [FCC+19] or stochastic optimization problem [HCHC19]; some methodologies focus on **solely V2V** offloading: Du et al. [DLZZ20] provide a collaborative offloading mechanism for sensing tasks in autonomous vehicle platoons, making use of idle resources. Guo et al. [GRG22] combine LSTM-based trajectory prediction and optimization strategy for V2V offloading. However, all these methods, while solid, rely on simulations rather than real-world data, assume static and homogeneous infrastructures, which are unrealistic, and frequently neglect SLO measures like energy consumption.

Offloading through Markovian and Bayesian methods

To the best of our knowledge, there are no solutions based on Bayesian Networks for V2V task offloading in platoons. Still, Markov models and Bayesian approaches are found in Edge-to-Cloud scenarios for task offloading [SPDD24b, SPDD24d]. Hazra et al. [HDAD23a] use MILP to find offloading locations in hierarchical computing environments under latency and energy constraints. Wu et al. [WCB+23] offload streaming tasks from edge nodes to fog or cloud resources through a Markov decision process, improved through Reinforcement Learning (RL). Tasoulas et al. [THB12] provide a prediction mechanism that uses historical observations to forecast VMs' resource demand through Bayesian Networks. However, these papers offer little variety for SLOs and do not incorporate dynamic or real-time adaptations.

Takeaways

Existing research focused extensively on MEC offloading mechanisms to RSUs or UAVs for optimizing network latency; however other objectives, as energy efficiency or QoS are often overlooked. In addition, most approaches were only evaluated in simulations; however, to establish reliable offloading mechanisms, it is paramount to consider dynamic runtime behavior. Conversely, we propose an SLO-aware mechanism for V2V offloading that optimizes various SLOs in heterogeneous vehicle platoons. Centralized approaches suffer from the combinatorial complexity of finding a global optimum and the risk of becoming a single point of failure; in our approach, however, services have decentralized authority to interpret their runtime behavior and make offloading decisions.

5.3.6 Summary

This subchapter introduced a novel V2V offloading mechanism that ensures high-level requirements during runtime. By leveraging probabilistic models, individual services estimate the resource demand over multiple services and the consequential SLO fulfillment at alternative hosts. We evaluated the proposed framework in a physical setup, in which platoon members feature heterogeneous processing devices. Noteworthy, we showed how the framework could handle an increasing number of platoon members and a

series of perception services; hence, it improves platoon-wide SLO fulfillment through decentralized decision-making. While this subchapter showed promising results, there remain limitations and areas of improvement: First, although baselines are scarce, the work must be contrasted with comparable approaches. While we ruled network latency negligible in our case, future work could also include this for more detailed analyses. Furthermore, our implementation executes services in Python threads; we plan a more effective and elegant solution, containerizing each service instance. Another interesting direction would be to explore more complex architectures in which a single platoon has multiple swarms or when multiple platoons need to coordinate with each other.

5.4 Takeaways

The results presented in this chapter support the claim that our contributions improve the state-of-the-art for orchestrating applications in DCCS. Nevertheless, there remain further challenges in this domain, most noteworthy the resilience and fault tolerance in large scalable architectures. Providing solutions to these problems and evaluating these solutions require considerable effort to build and maintain adequate evaluation environment. For this thesis, we did not want to apply any simulations but develop smaller, encapsulated software components that could be empirically evaluated. Hence, evaluating the presented ideas within larger environments remains for future work.

CHAPTER 6

Equilibrium through Active Inference

Computing Continuum (CC) systems are challenged to ensure the intricate requirements of each computational tier. Given the system's scale, the Service Level Objectives (SLOs), which are expressed as these requirements, must be disaggregated into smaller parts that can be decentralized. We present our framework for collaborative edge intelligence, enabling individual edge devices to (1) develop a causal understanding of how to enforce their SLOs and (2) transfer knowledge to speed up the onboarding of heterogeneous devices. Through collaboration, they (3) increase the scope of SLO fulfillment. We implemented the framework and evaluated a use case in which a CC system is responsible for ensuring Quality of Service (QoS) and Quality of Experience (QoE) during video streaming. Our results showed that edge devices required only ten training rounds to ensure four SLOs; furthermore, the underlying causal structures were also rationally explainable. The addition of new types of devices can be done a posteriori; the framework allowed them to reuse existing models, even though the device type had been unknown. Finally, rebalancing the load within a device cluster allowed individual edge devices to recover their SLO compliance after a network failure from 22% to 89%.

The remainder of this chapter is organized as follows: Section 6.2 introduces background knowledge and related work as a prerequisite. Section 6.3 presents our framework for collaborative edge intelligence. Section 6.4 contains the prototypical implementation of the framework and the evaluation methodology; the respective results are presented in Section 6.5. Section 6.6 provides an overview of existing research in this field. Finally, we summarize this chapter in Section 6.7.

6.1 Introduction

Computing Continuum (CC) systems, as envisioned in [B+20, DPD23, T+22], are large-scale distributed systems composed of multiple computational tiers. Each tier serves a unique purpose, e.g., providing latency-sensitive services (i.e., Edge), or an abundance of virtual, scalable resources (i.e., Cloud). However, the requirements that each tier must fulfill are equally diverse, as they span a wide variety of edge devices and fog nodes. Assume that requirements would be ensured in the cloud, e.g., by analyzing metrics and reconfiguring individual devices, massive amounts of data would have to be transferred. Also, if edge devices fail to provide their service to a satisfying degree, the latency for detecting and resolving this would be high.

Given the scale of the CC, requirements must be decentralized; this means that the logic to evaluate requirements must be transferred to the component that they concern. Cloud-level requirements, i.e., Service Level Objectives (SLOs), may thus be disaggregated into smaller parts that are ensured by the respective components. To contribute to high-level goals, each device optimizes its service according to its scope. This allows SLOs to span the entire CC, also called Deep SLOs [CPMM+23]. While it is one challenge to segregate and disseminate SLOs, ensuring them is another. Requirements are versatile and may change over time, every component must itself discover how its SLOs are related to its actions. For this to happen, the device could use Machine Learning (ML) techniques to discover causal relations between its environment and SLO fulfillment [SPDD23]. This promotes the usage of Active Inference (AIF) [FDCS+23], an emerging concept from neuroscience that describes how the brain continuously predicts and evaluates sensory information to model real-world processes. By extending individual CC components with AIF, they could develop a causal understanding of how to adjust their environment to ensure preferences (i.e., SLOs).

Ensuring SLOs autonomously (i.e., evaluating the environment to infer adaptations) makes components intelligent [KLM+23]; any system composed entirely of such intelligent, self-contained components becomes more resilient and reliable. No central logic must be employed to ensure SLOs; thus, higher-level components can rely on the SLO fulfillment of underlying components. Ascending from intelligent edge devices, the next level would be intelligent fog nodes; those we see in the ideal position to orchestrate the service of edge devices. Thereby, edge devices in proximity are bundled into a device cluster, administered by a fog node; whenever the Edge is scaled up with new devices (or device types), existing SLO-compliance models can be exchanged within the cluster. While each tier has its own SLOs, their tools for adaptation can have a different scale, e.g., fog nodes would be able to shift computations within clusters from devices that fail their SLOs. Such operations can consider environmental impacts (e.g., network issues) as well as heterogeneous device characteristics. The Cloud, as the next layer, would even have sweeping tools to ensure global SLOs.

To realize this vision, we present our framework for collaborative edge intelligence. Guided by AIF, individual edge devices gradually develop a causal understanding of how to

ensure their SLO. This knowledge is federated through a device cluster; edge devices of arbitrary types reuse existing models to ensure their SLOs. Thus, the entire Edge becomes spanned with SLO-compliant devices, which allows other CC tiers (i.e., up to the Cloud) to construct their service on top of that. By the same method, cluster leaders infer how to adjust their environment; thus, each tier may achieve an equilibrium for the compound service offered. Hence, the contributions of this chapter are:

- An AIF-based ML technique that allows CC components to gradually identify causal relations between environmental metrics and SLO fulfillment. Components can thus evaluate SLOs decentralized and update their beliefs according to new observations.
- The transfer and combination of ML models between heterogeneous devices to accelerate their convergence towards SLO-fulfilling configurations. This simplifies the onboarding of new device types (i.e., horizontal scaling) on the Edge.
- An offloading mechanism that redistributes load in an edge-fog cluster according to devices' capabilities to fulfill high-level SLOs. Thus, it counters environmental factors and improves the cluster-wide level of QoS and QoE.

6.2 From Neuroscience to Computer Science

The framework presented in this chapter builds heavily on two existing concepts that we adapt for our usage, namely causality and AIF. Although these topics might be known to some readers, we provide this section to ensure a solid understanding of their core aspects and terminology. Furthermore, since both concepts are not native to computer science (or distributed systems), we highlight existing intersections as far as possible.

Causality and Causal Network Graphs

Causality allows modeling causal relations between events or variables. While spurious correlations are misleading and hide the true causes, causality answers *why* an event happened. However, to identify causal relations, specific experiments and consideration of expert knowledge are required. To define a general theory of causality, Pearl [Pea09] proposed Structural Causal Models (SCMs). Such a mathematical model can be expressed through causal graphs, e.g., as Directed Acyclic Graph (DAG). Thus, variables can be arranged from cause to consequence.

Causality is a hot topic in research because of its ability to provide explanations for phenomena through interpretable graphical models. This is why many works link causality and machine learning; see [GFB+23] for a comprehensive review. Thereby, causality can also be embedded into distributed systems, e.g., for root cause detection [CQH19]. As another instance, *Lin et al.* [LCZ18] use causal graphs in Cloud computing to detect dependencies within a microservices-based architecture. For such use cases, DAGs are an

ideal modeling tool. Interestingly, they monitor SLOs to trigger causal inference over their causal graphs, being able to detect the source of the SLO violation.

Another crucial concept for this chapter – or generally for scalability in the CC – is the Markov Blanket (MB). Consider a Bayesian network (BN) represented as a DAG (e.g., Fig. 6.2): a random variable is conditionally independent of all other variables, given its MB. In other words, the MB of a variable *shields* it from external variables. In a DAG, the MB of a variable consists of its parents, children, and co-parents. Discovering the structure of BNs and extracting MBs through data is not a simple task, and many works are devoted to that; see [TAS03] or [NMC07] for specific techniques, and [VCB21] for a thorough survey. Regardless of the system size, MBs can achieve modularity; thus, the system can be managed and controlled on a convenient scale.

Graph-based causal models promise to make systems explainable. Inspired by that, our work stems from [DPD23, PRD21] to build MBs around SLO-governed components. Thus, it becomes possible to isolate the system variables that affect SLO fulfillment. On the one hand, this drastically reduces the number of variables required for analysis thanks to conditional independence; the system can thus be managed at scale. On the other hand, it is possible to leverage the BN to explain causal effects between variables in the MB and the SLOs' behavior (e.g., failure).

Active Inference

In this chapter, we use AIF to extend devices with causal knowledge on how to fulfill their SLOs. However, we consider AIF an unknown concept for most readers outside of neuroscience; therefore, we use this section to summarize core concepts of AIF according to *Friston et al.* [Fri13, KPP$^+$18, FDK09, SFW22, SBPF21, PPF22].

Core Concepts To interpret observable processes, agents generate models that resemble these processes, e.g., humans reason that it rains due to water drops falling from the sky. However, if this generative model and the real-world process diverge, the agent will eventually be "surprised", e.g., because water drops were actually caused by a neighbor watering her plants. The discrepancy (or uncertainty) between the agent's understanding of the process and the reality is called Free Energy (FE). In simple terms: the lower the FE, the higher the prediction accuracy.

Internally, agents organize generative models in hierarchical structures; each level interprets lower-level causes and, based on that, provides predictions to higher levels. For example, suppose (1) it rains with a certain probability, (2) I bring an umbrella. This is commonly known as Bayesian inference and allows agents to use priors (i.e., existing beliefs) to calculate the probability of related events. Thus, decision processes can be segregated into self-contained causal structures (i.e., MBs) that share only a limited number of interface variables. For example, only the weather state (*rainy* or *sunny*) is considered for picking the umbrella; any lower-level observations that determined the agent's perception of the weather (e.g., humidity or illumination) are disregarded.

To decrease FE, AIF agents repeatedly engage in action-perception cycles by (1) predicting outcomes, (2) awaiting (or seeking) the outcome, and (3) updating beliefs. Afterward, they can actively adjust the environment to their beliefs. As generative models become more accurate, causal relations between their preferences (e.g., SLOs) and the environment are revealed. However, the ability of agents to discover causal relationships is highly dependent on the number and accuracy of observations [CVGN+23]. Fortunately, the CC provides large amounts of operational metrics.

Some aspects of AIF, in particular decision-making, intersect with reinforcement learning. Notably, the two approaches are not mutually exclusive, on the contrary, they are complementary as shown by existing works [FDK09, MKBC21, TMSB20]. Important differences of AIF are that agents are biased when they try to adapt the exterior towards their beliefs and that they are specialized in minimizing surprise for an empirically verifiable model.

Intersection with Distributed Systems Considering presented works, most research on AIF has not been embedded and evaluated in operative distributed systems (e.g., [SFW22, HMD+22]). To the best of our knowledge, our latest research [SPDD24a] is thus among the few works that embedded AIF into distributed systems; another work that we want to highlight is *Levchuk et al.* [LPS+19], which created a decentralized mechanism for team adaptation. For the remaining chapter, our work in [SPDD24a] serves as a reference on how AIF agents can infer SLO-compliant device configurations: agents operate parallel to continuous processing and adapt their generative models according to prediction errors. We call such a model – at its core a BN – an Equilibrium-Oriented SLO-Compliance (EOSC) model. In this chapter, we will extend the EOSC model to achieve equilibrium in the CC.

6.3 Collaborative Edge Intelligence

To ensure SLOs throughout computational tiers, we propose our framework for collaborative edge intelligence that encompasses three main contributions: (1) The continuous model optimization based on AIF, which ensures SLOs (locally) on a device basis; (2) the federation and combination of EOSC model between edge devices, which decreases the overhead of training models for different device types from scratch; and (3) the evaluation of SLOs on a cluster-level, which can rebalance load within the cluster according to environmental factors.

These three contributions are described in the respective subsections (6.3.1 to 6.3.3); Figure 6.1 contains a high-level overview of the framework's capabilities. On the left, it is depicted how SLOs are evaluated to continuously train an ML model and adapt the service accordingly; this model is then federated and combined at a fog node, which provides the model to an unknown device type (marked as red). The fog node analyzes the overall SLO fulfillment in the cluster; if it appears beneficial to offload computation from one device to another one (e.g., from the blue to the red one), this is orchestrated by

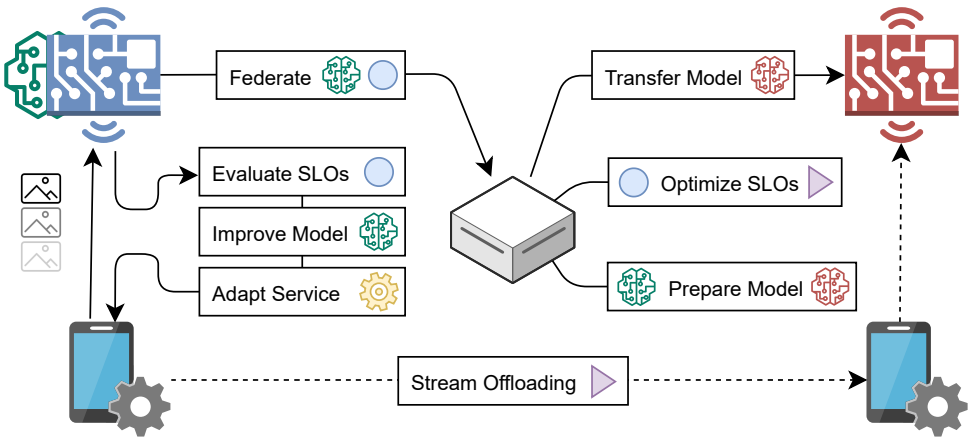


Figure 6.1: High-level overview of the collaborative edge intelligence framework that continuously improves model evidence, shares this knowledge between edge devices, and optimizes SLO fulfillment within this cluster.

the fog node. Logically, the model transfer and load balancing rely on the SLO fulfillment in the Edge; this is why all three contributions are required to ensure SLOs on multiple tiers (or the entire CC).

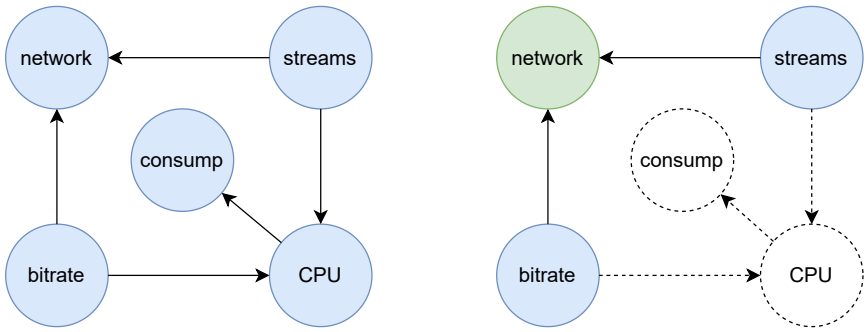
6.3.1 Continuous Model Optimization

An accurate generative model allows to explain a system's behavior (e.g., why SLOs were violated), infer how to adapt the system to ensure SLOs, and predict how changes will affect this. Further, prediction errors are propagated back to the agent so that the model can be improved according to the experienced deviations. In the following, we will first present the representation of the EOSC model and the applied training method. Afterward, this process is integrated into an AIF agent, which uses this process to continuously improve the model accuracy.

Static Model Training and Inference

To bootstrap from evaluated concepts, we use the Bayesian Network Learning (BNL) methodology presented in Section 4.2. This allows us to: (#1) train generative models of processing tasks, (#2) filter the Markov blanket around SLOs, and (#3) extract knowledge from Bayesian networks. The following paragraphs explain how this methodology was incorporated into the AIF framework presented in this chapter.

Bayesian Network Learning As presented, BNL is an efficient way to generate an accurate structure from data; its two main parts are `STRL` – structural learning of causal dependencies (i.e., DAG), and `PARL` – parameter learning as quantification of variable dependencies. For a data set with 5 columns, the resulting DAG could look like Figure 6.2a. The AIF agent uses these methods for constructing (and later updating)



(a) Entire DAG (b) MB for *network*

Figure 6.2: Causal variable relations in the DAG of a trained BN

the EOSC model: for a data set D, it trains a BN as $model = \texttt{PARL}(\texttt{STRL}(D), D)$; this first creates the BN structure and then the its conditional dependencies.

Markov Blanket Selection As presented, the Markov blanket shield a variable from all nodes that are conditionally independent of it. Suppose we specify an SLO according to device capabilities (e.g., network throughput $< t$) and evaluate it using a single variable (e.g., *network*), we want to identify metrics related to SLO fulfillment. Namely, these are all variables contained in the MB of *network*; the function $\texttt{MB}(model, network)$ thus returns all blue nodes in Figure 6.2b. In this context, we distinguish between metrics that statically reflect the system state (e.g., *CPU*), and those that represent a parameterizable variable (e.g., *bitrate*). However, we summarize both using the term "metrics" from a BNL perspective. While static metrics are essential to explain why an SLO is in its current state, only parameterizable ones can be dynamically reconfigured, i.e., they are the possible action states of the AIF agent. Overall, the sum of metrics in the MB provides a clear understanding of why an SLO is in its current state.

Knowledge Extraction Consider the DAGs from Figure 6.2: We construct a QoS SLO that is fulfilled if *network* is below t and infer the probability of SLO violations for different variable assignments. As discussed, VE accepts a list of target variables (T), variable assignments (A), and an elimination order (O); by iterating over O, the graph eventually contains only T. To decrease the complexity of VE, we execute it on $mb = \texttt{MB}(model, network)$, the node list thus equals $\{network, streams, bitrate\}$. Later, we call VE through $\texttt{INFERENCE}(m_x, T, A)$, where m_x can be any subset of the BN. If we execute INFERENCE with mb, $T = \{network\}$, $A = [(streams : 2), (bitrate : 720)]$, and arbitrary O, the result contains all conditional probabilities of *network* given the variable assignment; from this we can extract $P(network > t)$. This is our central mechanism for identifying probabilities of SLO violations given a system state.

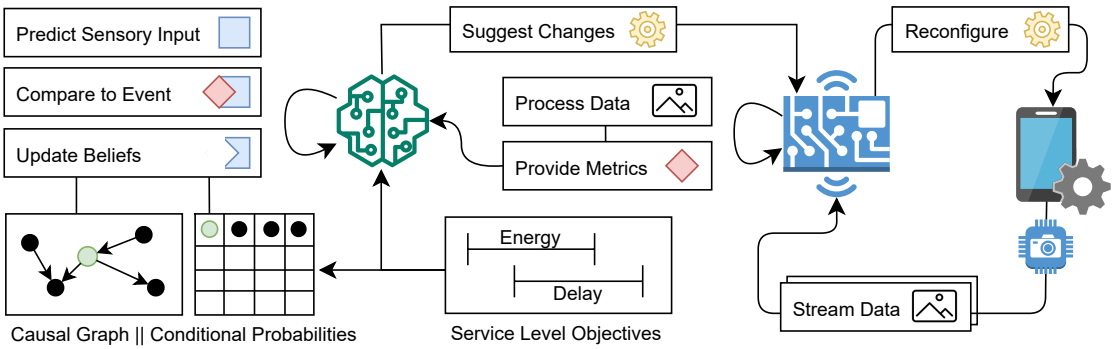


Figure 6.3: Overview of Active Inference cycle: learning how to fulfill SLOs by continuously training a generative model from metrics and inferring device configurations

Active Inference Cycle

The tools presented in the last section created a BN from processing metrics, extracted an MB, and inferred system configurations that fulfill given SLOs. Supposed there is sufficient data available, BNL can be a one-time process; however, there are two fundamental issues: (1) data shifts, which likely occur after some time, will inevitably distort the accuracy of the ML model, and (2) it is impractical to empirically evaluate how an exponential number of system configuration impacts SLO fulfillment. Large and complex systems, such as the CC, require a different approach: creating and updating a model incrementally according to new observations while drawing conclusions for unknown parameter combinations from existing data.

To evaluate this parameter space of configurations, we extend the AIF agents from [SPDD23] to interpolate between empirically evaluated combinations; to maintain the model's FE low, agents continuously update conditional probabilities of variable relation according to new observations. By design, our AIF agents can be employed at any CC tier; nevertheless, this chapter is focused on intelligent edge devices, which collaborate under the supervision of a fog node. Thus, we raise the granularity of intelligence from the Edge to the Fog. In the following, we present the different tasks executed by an AIF agent; this includes training and updating the BN, as well as evaluating its scope of actions according to a set of behavioral factors. Based on that, agents decide how to modify the system; each of these changes is again reflected by system metrics.

Agent and Operation The AIF agent operates parallel to regular device tasks, e.g., serving clients. Although regular operation, model training, and inference are logically separated, they take place on the same physical device; Figure 6.3 contains a visual representation: assume an edge device that continuously performs a workload, e.g., processing client data. The agent observes the device state and the environment through metrics; thus, it can evaluate whether processing complies with SLOs, e.g., if a request was finished with $delay < t$. From that data, the agent creates a BN, where conditional

probabilities reflect the SLO fulfillment under a discrete environmental state. Then, the agent starts with predictive coding, i.e., forecasting whether future events will fulfill SLOs, comparing the expectation with actual observations, and updating the BN accordingly.

After each iteration, the agent infers how to modify the system configuration to optimize local SLO fulfillment. Following that approach, the AIF agent can create a generative model from scratch or update a BN according to new observations by following its sensing-acting loop. Thus, it is possible to cancel out data shifts, e.g., the result of a model transfer from one edge device type to another. AIF can therefore perform the fine-tuning that is required after such an operation.

Free Energy Minimization To create an accurate model, the AIF agent operates in cycles; each cycle processes a *batch* of observations that reflects the environmental state, including the latest system configuration. The agent continuously evaluates the *batch*, updates its *model*, and chooses which system configuration (c_{next}) to choose for the next iteration. Throughout cycles, the AIF agent has one central goal: decreasing the FE, or in other words, minimizing surprise of predictions. Therefore, we will first present how we calculate surprise and then embed it into the high-level loop executed by the agent.

For calculating the surprise for *batch* and *model* we present Algorithm 6.1. To decrease the complexity, we limit the calculation to variables that directly reflect SLO fulfillment (V_{SLO}), and execute `INFERENCE` only on the MB of V_{SLO} (Line 2). This node set is further filtered (Line 5) to contain only the evidence variables (*ev*) that impact the outcome of *var*; afterward, in Line 7, each *row* in the *batch* is filtered to contain only these variables. In Lines 8 & 9, the probability of observing *var*, i.e., the state of the SLO, given the environment (*evidence*) is first inferred and then appended as *log_likelihood*. For each *var*, the *cpt* from *model* is considered, from which k – the number of states – can be extracted as a representation of model complexity. `CPT` is as a helper function to get the Conditional Probability Table (CPT) for a *var* in *model*. Together with n – the number of observations – the BIC is calculated (Line 14). After calculating the surprise for each $var \times row$, this overall sum is returned.

The surprise has a special role within our AIF cycle, as it determines when and how BNL takes place; consider therefore Algorithm 6.2, which shows the high-level loop executed by the AIF agent. At the beginning of each iteration, the agent ensures that there exists a model, otherwise, it creates an initial structure from *batch* (Lines 1 & 2). Notice, that `STRL` and `PARL` accept now another parameter – *model* – which allows to update the DAG and CPTs of *model* according to *batch*. Whether `STRL` or `PARL` is executed (Lines 7-11) is determined by the surprise magnitude (s). If s exceeds the median surprise of the last 10 rounds (m_{10}) by a custom factor h, `STRL` is applied; otherwise, if s exceeds m_{10}, `PARL` is applied. This distinction is necessary because `STRL` and `PARL` have quite different runtimes, as we will reveal in Section 6.5. Finally, in Lines 12 & 13, the agent evaluates possible system configurations and determines which one it will use for the following iteration. We will explain these two functions in the next two paragraphs.

Algorithm 6.1: `SURPRISE` for model and batch

Require: $model$, $batch$, V_{SLO}
Ensure: $\Im$ // surprise over all observations
1: $\Im \leftarrow 0$
2: $mb \leftarrow \texttt{MB}(model, V_{SLO})$
3: **for each** var **in** V_{SLO} **do**
4: $\quad log\_likelihood \leftarrow 0$
5: $\quad ev \leftarrow \texttt{MB}(model, var)$
6: $\quad$ **for each** row **in** $batch$ **do**
7: $\quad\quad evidence \leftarrow row \cap ev$
8: $\quad\quad p \leftarrow \texttt{INFERENCE}(mb, var, evidence)$
9: $\quad\quad log\_likelihood \leftarrow log\_likelihood + \log(p)$
10: $\quad$ **end for**
11: $\quad cpt \leftarrow \texttt{CPT}(model, var)$
12: $\quad k \leftarrow |cpt|$ // number of states in the CPT
13: $\quad n \leftarrow |batch|$
14: $\quad bic \leftarrow (-2) \times log\_likelihood + k \times \log(n)$
15: $\quad \Im \leftarrow \Im + bic$
16: **end for**
17: **return** $\Im$

Behavioral Factors The behavior of an AIF agent, i.e., how it selects between possible actions, is determined by three major factors: The pragmatic value (pv) defines how well the device fulfills client expectations, e.g., if video *resolution* is satisfactory. The risk assigned (ra) determines how likely the system will fail its service, e.g., if stream packets are delivered on time. Lastly, the information gain (ig) represents the agent's expectation of how much it can improve model accuracy. The ig is directly related to surprise minimization, whereas pv and ra reflect the agent's capability to fulfill SLOs. To separate concerns, we divide SLOs according to their characteristics: pv represents QoE requirements, while ra contains QoS requirements. Combined, these three factors determine the agent's behavior; in the following, we will calculate each of them.

To infer the optimal device configuration (i.e., highest SLO fulfillment),the agent limits itself to finding the Bayes-optimal configuration [GABZ23], i.e., optimal under current knowledge. Therefore, the AIF agent first infers the assignment for known parameter combinations (c_k) that were empirically evaluated and then interpolates between these values to span the entire parameter space. Calculating pv and ra is similar to Algorithm 6.1 (Lines 5-8): It requires a subset $V_Q \subseteq V_{SLO}$ – either QoS or QoE SLOs – which is used as $ev \leftarrow \texttt{MB}(model, V_Q)$. For each row in c_k, $evidence$ is constructed equally, so that $\texttt{INFERENCE}(mb, V_Q, evidence)$ provides the joint probability of QoS or QoE violations.

$$ig(c) = e + \left(\frac{\tilde{\Im}_c}{\bar{\Im}} \right) \times 100 \tag{6.1}$$

Algorithm 6.2: An Iteration in the AIF Cycle

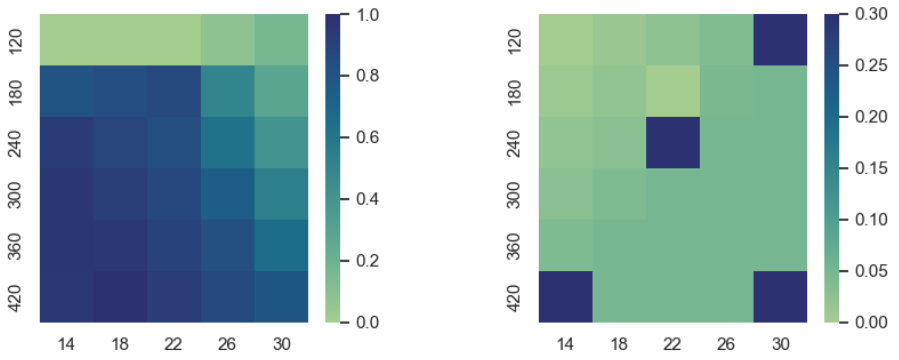
Require: $model$, $batch$, $\Im$, h, V_{SLO}
Ensure: c_{next} // Next configuration
1: **if** $model = \emptyset$ **then**
2: $model \leftarrow \texttt{PARL}(\texttt{STRL}(\emptyset, batch), batch)$
3: **end if**
4: $s \leftarrow \texttt{SURPRISE}(model, batch, V_{SLO})$
5: $\Im \leftarrow \Im \cup \{s\}$
6: $m_{10} \leftarrow median(\Im_{10})$ // over the last 10 values
7: **if** $s > (m_{10} \times h)$ **then**
8: $model \leftarrow \texttt{STRL}(model, batch)$
9: **else if** $s > m_{10}$ **then**
10: $model \leftarrow \texttt{PARL}(model, batch)$
11: **end if**
12: $K \leftarrow \texttt{CALCULATE\_FACTORS}(model)$
13: $c_{\text{next}} \leftarrow \texttt{BEST\_CONFIGURATION}(K)$
14: **return** c_{next}

In accordance with [SPDD24a], high surprise indicates high information insight and, hence, possible improvement of the model precision. However, from an agent's perspective, is it worth abandoning a supposedly satisfactory configuration (in terms of pv and ra) to search for a global optimal one? This presents a tradeoff between exploration of unknown areas and the tendency to stick to exploited areas; multi-agent systems commonly model this through hyperparameters (e.g., [LPS+19]). In our case, we calculate the ig of a configuration $c \in c_k$ as presented in Eq. (6.1) [SPDD24a]: it compares the median surprise ($\tilde{\Im}_c$) for c with the overall mean surprise ($\bar{\Im}$). Configurations with high $\tilde{\Im}_c$ will thus be preferred by the AIF agent.

Parameter Space The AIF agent calculates the behavioral factors for all entries in c_k and summarizes them as K (Line 12 of Algorithm 6.2). For the next step, imagine two configuration parameters $\{fps, pixel\}$ with their combinations arranged in a 2D $[fps \times pixel]$ matrix. After calculating K, the blank spaces in the parameter matrix are filled by performing linear interpolation[1]. As a potential result, consider the matrix depicted in Figure 6.4a. Later, in Section 6.4.2, the agent will interpolate in a 3D parameter space.

Contrarily to pv and ra, the agent does not apply interpolation to estimate the ig of an unknown parameter configuration. Instead, in the absence of observations for c, it assumes that $ig(c) = \max(\Im)$. Further, it remains to introduce a hyperparameter from Eq. 6.1, namely e. To improve the interpolation of pv and ra, the agent initially focuses on key

[1] In fact, this is done using Python scypy, which triangulates data through a convex hull to perform linear barycentric interpolation on each triangle.



(a) Interpolation for pv (b) ig after 1 round

Figure 6.4: Matrices of behavioral factors used by the AIF agent

positions of the possible configurations. Figure 6.4b illustrates that tendency; the visually highlighted blocks are increased by $e = 0.3$. When calculating the behavioral factors, the AIF agent thus initially focuses on these cornerstones to set up the interpolation; after visiting c, it subtracts e from $ig(c)$.

To summarize possible risks but also benefits that emerge from a configuration c, we combine the three factors under a common one (u) that we calculate as $u_c = pv_c + ra_c + ig_c$. The AIF agent compares common factors of all possible configurations and selects the highest-scoring (Line 13 of Algorithm 6.2). By repeating this cycle, the agent gradually develops an understanding of which areas in the parameter space are more likely to fulfill SLOs, e.g., the left-bottom area in Figure 6.4a.

Final AIF Agent This concludes the agent's continuous model optimization, which maintains an up-to-date model of a processing task (i.e., the generative process). The high accuracy in the EOSC model allows the AIF agent to infer (Bayes-)optimal device configurations, which ensures QoS and QoE of ongoing operation. In the following, we will now focus on the collaboration between the Edge-based agents.

6.3.2 Knowledge Transfer within the Cluster

By now, we presented AIF agents that can create generative models from scratch or update a model according to new observations. However, if we assume a cluster of nearby devices that process similar workloads, training EOSC models for every device seems redundant. Also, if we aim to extend the cluster with more devices (i.e., scaling up horizontally), model training delays the time until devices operate according to requirements. Instead, we envision the federation of knowledge between edge devices by exchanging EOSC models within the device cluster. Such a transfer learning approach appears to be a straightforward process if the models were trained in the exact same environment [WWZ+17]. However, the Edge is composed of multiple heterogeneous

device types; the resulting models thus reflect the characteristics of the device it was trained on, i.e., its capability to cope with SLOs depends on the processing hardware. For example, a multi-core device is certainly capable of processing multiple video streams, while a single-core one is not. Furthermore, the behavior of AIF agents (i.e., which action it takes) and environmental dynamics (e.g., demand) determine which parameter combinations get more or less exploited.

Whenever a new device (type) joins a cluster, the question is whether there exists a device within the cluster whose environment and characteristics match the newly-joint device's. Meanwhile, devices present in the cluster share their EOSC models and device characteristics (e.g., hardware specs or environmental factors) with the cluster leader (i.e., standing hierarchically above the device cluster). As a new device joins the cluster, its characteristics are compared with the present ones to select a fitting model. In cases where the characteristics of multiple devices are similar, their models are merged and provided to the newly-joint device. Thus, the newly-joint device builds its EOSC model on top of existing knowledge in the federation.

In the following, we dive deeper into this transfer-learning process by answering (1) how models are federated between devices, (2) how hardware characteristics are compared to select a model, and (3) how models are combined to fit a target device.

Cluster-wide Model Exchange

The EOSC model exchange knows two roles: (1) consumer – when joining a device cluster it might be preferable to adopt an existing model rather than training one, and (2) provider – any device might itself share its model with devices that join the cluster. The selection of a fitting model, however, can happen on any trusted device; we assume for this task either a cluster leader (i.e., an outstanding device elected due to its capabilities) or a powerful fog node. To provide an estimation, these models are supposedly smaller than 2 MB, as measured in [SPDD23, SPDD24a].

When making the architectural decision (i.e., cluster leader or fog node), various factors can be considered, among them: network scale, cost, geographic location, and availability. In cases where the cluster would be small (e.g., 10 devices), an edge device (e.g., from Table 6.3) could cope with collecting and preparing EOSC models; however, for larger clusters (e.g., 1000 devices), regular edge devices might fail to do so. In any case, a strong factor for using fog nodes is their high availability – fog nodes can reliably cache a high number of models from various devices. Either choice, they assume equal responsibilities, thus we call them simply *leader node*. This leader node periodically collects EOSC models of devices registered in the cluster, as well as their hardware characteristics. Based on this information, models will be provided for new device types.

Model Comparison and Selection

Transferring a EOSC model to a newly-joint device raises two questions: First, is the transfer of an existing model more efficient than learning the model from scratch? And

second, how to choose the most convenient model for the new device? Of course, the second question assumes that the device type is unknown and the cluster does not contain the respective trained models so far. The first question will be answered and discussed as a result of this article; the second question, however, requires building a hypothesis around how to choose a model.

The dynamism within the training environment has a decisive impact on the resulting model: applications with a stable number of user requests do not suffer many dynamics, while applications that are linked to specific events (i.e., disaster management) can experience extremely different requirements. However, we assume that environmental factors are out of our hands – we are unaware of the dynamics of the environment in which the device is set. Due to that, we focus on the device characteristics when transferring models between edge devices. To that extent, we get inspiration from the work of *Casamayor et al.* [PMN23], which allows classification of heterogeneous characteristics of the devices found in a cluster, namely their CPU and GPU capacity. This means that we relatively classify the CPU capacity (p) of the devices in the cluster in a range $[p_{min}, p_{max}]$, and their GPU capacity (g) from $[g_{min}, g_{max}]$. Given that there are numerous edge devices without GPU, it is possible to set $g_{min} = 0$. To make this more tangible, in Section 6.4.2, we present a list of edge devices whose hardware is classified accordingly. Finally, we define each device's capacities as $dc = p + g$. To estimate the similarity of device characteristics and to identify a device with a matching model, the leader node selects the device(s) with the closest integer dc.

Combination and Preparation of Models

Heterogeneous edge devices differ in terms of hardware characteristics. Using the presented mechanism, there would frequently occur situations in which there is not exactly one device that trumps all others. For example, consider a device with type t_x that joins a device cluster; there are already numerous device types present, among them t_a and t_b. The leader node classifies their capabilities as $dc_a = 3, dc_b = 5,$ and $dc_x = 4$. Which model should now be provided to t_x, the one trained on t_a or on t_b? And in case $dc_a = 2$ and $dc_b = 7$; is choosing dc_a really the smartest choice?

For both cases we merge the models from t_a and t_b, thus creating a new model m_{ab} that presents the intersection. In the second case, where dc_x does not fall exactly between dc_a and dc_b, this is done proportionately. Therefore, we require a mechanism to combine EOSC models – still BNs at their cores. To date, merging BNs is an ongoing research field that still presents various limitations [VRP22, VLS23]; in most cases, it is coupled to conditions that models must fulfill. Due to this, we limit our work to merging CPTs. As long as two models m_a and m_b contain the same structure (i.e., their DAGs are identical) and their CPTs have the same cardinality (i.e., variable states), this is done as follows: For a random variable r and its $\texttt{CPT}(m, r)$, each table cell's expected value (P) is calculated as shown in Eq. (6.2); P_a and P_b represent probabilities of m_a and m_b, the coefficients w_a and w_b reflect the distribution of dc_x between dc_a and dc_b. For example, if dc_x is aligned centrally between them, they take the value $w_a = w_b = 0.5$; otherwise, it

is shifted proportionally, but $w_a + w_b = 1$ must remain true.

$$P_x = (w_a \times P_a) + (w_b \times P_b) \tag{6.2}$$

If m_a and m_b do not fulfill these requirements, they would have to undergo a transformation process. Nevertheless, in Section 6.4.2, we apply a workaround to merge BNs whose CPTs have different cardinalities. After merging the EOSC models, the leader node provides m_{ab} to the newly-joint device; once received, transfer learning is completed. Thus, it decreases the time for model training or even skips it entirely.

6.3.3 Stream Offloading in the Edge-Fog Cluster

Regardless of whether trained by an AIF agent or transferred from another device, a EOSC model is a decisive step toward SLO fulfillment. Thus, edge devices are continuously reconfigured to achieve maximum SLO compliance. However, despite our efforts, edge devices are still vulnerable to environmental factors that cannot be controlled, e.g., irregular peaks in client traffic. While a EOSC model can have a hard time finding an SLO-compromising device configuration, idle edge devices in close proximity might be available for offloading computation. Again, to match our desired level of intelligence, this can be achieved through collaboration between the agents. Given that the struggling edge device is part of a device cluster, it is possible to (1) compare the device's capabilities to fulfill their SLOs within their environment, and (2) balance the load accordingly. Notice, that shifting the load within the cluster is a (local) reconfiguration that follows the same rules as in Section 6.3.1; this time, however, on a higher level.

In the following, we describe how to evaluate, analyze, and optimize the cluster-wide SLO compliance; the overall process is visible in Figure 6.5: The edge devices in the cluster (red & blue) serve their respective clients, e.g., by processing data, which is subject to dynamic reconfiguration according to the EOSC model. Throughout processing, the edge devices supply their SLO fulfillment to the leader node. Among that, they provide other factors (i.e., as metrics) that potentially impact the fulfillment. Environmental factors (e.g., insufficient hardware, power shortage, or client demand) can thus be contrasted with the devices' capacity to fulfill SLOs. Based on that analysis, the leader reconfigures the cluster (e.g., by redistributing the load) so that QoS and QoE SLOs are optimized within the cluster.

Cluster-wide Evaluation of SLOs

To analyze SLO fulfillment on a cluster level, the leader node does not reevaluate lower-level SLOs – this was already covered within the Edge. Instead, the leader node merely collects SLO compliance rates per device as a combined factor $f = pv \times ra$. These metrics are collected at the leader; depending on the desired amount of historical data, the high availability of the Fog would again be beneficial for collecting data. The question is now how to transfer metrics: Considering the potential size of a device cluster, we opt for a push-based approach, where devices periodically supply their data to the leader.

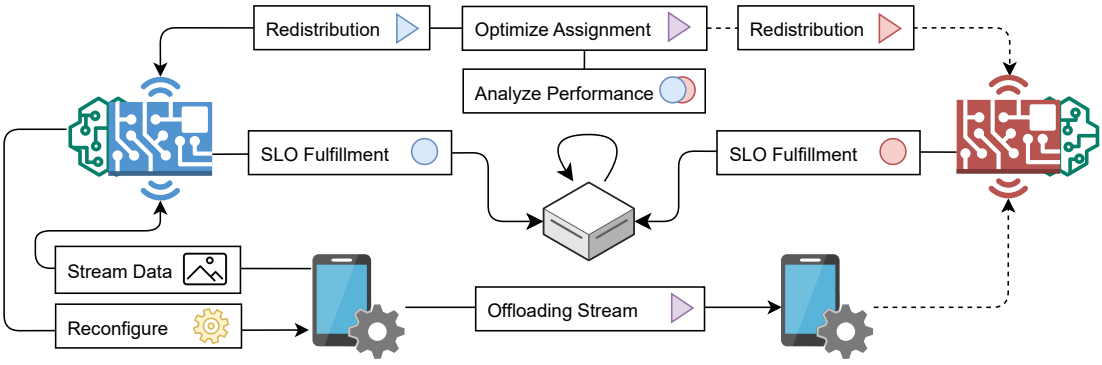


Figure 6.5: Evaluating SLOs within a device cluster and reassigning tasks

Apart from the SLO fulfillment, edge devices provide metrics that reflect their current environmental state. This includes any factors that the leader node should consider. If a battery-equipped device suffers occasional power shortages, it can report this conditional to the leader node, which adapts the network, e.g., by offloading computations to other devices to decrease its power drain. However, in the event of an entire network outage, devices can be incapable of reporting their state, and another node (e.g., leader) would have to detect this. Other frequent conditions can be general network congestion, including poor latency, jitter, or packet loss, but also devices' geographic location, user density, and peak usage times. Given their impact on the devices' capacity to fulfill SLOs, the leader node will rebalance the environment.

Analysis & Optimization per Device

Optimizing the devices' environments requires methods to draw conclusions between discrete environmental states and their consequential SLO fulfillment. To that extent, we aim – again – to identify causal relations between metrics; however, this time on a cluster level. Given a metric set (i.e., reflecting the environmental state) and the respective SLO rates per device, the leader node can construct a BN and infer how environmental changes impact the SLO fulfillment. To accelerate the construction of such a model, the leader node can combine metrics from devices of the same type, or even those that have comparable hardware characteristics (as done in Section 6.3.2). Although we ascended from an Edge to a cluster level, we still use the same tool for analyzing and adapting the environment – the EOSC model. However, to make a distinction, we call this new instance a EOSC-F (Fog) model.

Given a trained EOSC-F model (or rather, its BN), it is evident which environmental factors (σ_{env}) have a causal impact on SLO fulfillment. This can also help to improve the QoS in the long run, e.g., by pinpointing issues within the infrastructure. However, we aim to ensure SLO fulfillment the moment the QoS or QoE drops; the EOSC-F model can therefore consider the devices' environment and redistribute client load to ensure maximum SLO fulfillment within the cluster. To that extent, we present Algorithm 6.3,

Algorithm 6.3: Client reassignment algorithm

Require: $model$, n_{client}, σ_{env}
Ensure: ass // assignment according to env. state

1: $i \leftarrow 0$
2: $ev \leftarrow \texttt{MB}(model, f)$
3: **for** each $\lambda \in \Lambda$ **do**
4: $\quad ass[\lambda] = 0$
5: **end for**
6: **while** $i < n_{clients}$ **do**
7: $\quad \delta_{best} = -\infty$
8: $\quad$ **for** each $\lambda \in \Lambda$ **do**
9: $\quad\quad evidence \leftarrow ev \cap (\sigma_{env}[\lambda] \cup ass[\lambda] + 1)$
10: $\quad\quad \delta \leftarrow \texttt{INFERENCE}(ev, f, evidence)$
11: $\quad\quad$ **if** $\delta > \delta_{best}$ **then**
12: $\quad\quad\quad \delta_{best} = \lambda$
13: $\quad\quad$ **end if**
14: $\quad$ **end for**
15: $\quad ass[\delta_{best}] \leftarrow ass[\delta_{best}] + 1$
16: $\quad i \leftarrow i + 1$
17: **end while**
18: **return** ass

which distributes a number of streams (n_{client}) between the devices (Λ) in the cluster. `Inference` is again executed only on the variables that relate to SLO fulfillment, i.e., $\texttt{MB}(model, f)$, by filtering the *model* (Line 2 & 10). In Lines 6-18, the agent then iteratively assigns clients to the device, whose SLO fulfillment is the least impacted by receiving another stream ($ass[\lambda] + 1$). This assumes, that both ass and σ_{env} are part of ev, i.e., have an impact on SLO fulfillment. To that extent, $\sigma_{env}[\lambda]$ can contain factors like device characteristics. After assigning all streams within the cluster, the assignment can be orchestrated to the clients.

Orchestration and Redistribution

As a last step, the new cluster configuration must be enforced; in this case, by informing pertinent devices of the new assignment. The leader node pushes this information to all edge devices that must alter their configuration. In accordance with Figure 6.5, this includes all devices that offload or receive clients (red & blue); thus, the red device redirects clients to the blue device. To improve the SLO fulfillment within the cluster, the assignment considered each device's environment to provide an adequate configuration on a cluster level. Regardless of whether the QoS was impacted by poor network conditions or by poor hardware, if these conditions are packed as stateful information, the leader node optimizes the cluster accordingly. Thus, covering heterogeneities between edge devices, which otherwise fail to scale their service given the environmental stress.

This concluded the client load redistribution, which optimized overall SLO fulfillment in the cluster according to the EOSC-F model. To transfer intelligence to the network edge, or even to the level of cluster or fog nodes, this section provided various concepts that all had the same goal: ensure SLOs in the respective system. It remains to provide a prototypical implementation of presented ideas, evaluate it according to key aspects, and argue to what extent it is ready for wider adoption.

6.4 Use Case: Distributed Video Processing

In the following, we describe a CC scenario that requires edge devices to continuously transform video streams; this use case poses various requirements that must be ensured throughout processing. Afterward, we outline our prototype that ensures SLOs through collaborative edge intelligence. Essentially, this is the implementation of the presented framework. Lastly, we explain the methodology according to which the prototype will be evaluated. Section 6.5 will contain the respective results.

6.4.1 Use Case Description

The CC as a distributed system provides unprecedented opportunities for service providers and clients, e.g., in terms of processing or requirements assurance. As an example, consider a region with frequent natural disasters where the humanitarian situation should be documented. Therefore, reporters provide video streams in which vulnerable groups, e.g., minors of age, are detected. In the same step, individuals can be counted or visually highlighted; their identities, however, must be preserved. The region suffers from occasional network breakdowns (i.e., this affects access to global resources like the cloud but not internal connectivity); the reporting team thus provides ad hoc networking infrastructure in the form of edge devices, which are installed in close proximity to the operation area. Reporters equipped with IoT cameras are now capturing their surroundings; the video streams are transformed on edge devices, where they can be cached as long as global internet services are unavailable. Once resumed, videos are streamed to a cloud platform that provides the content to worldwide consumers.

Envisioned Solution Due to the nature of how disasters happen, it is impossible to fine-tune the complete streaming architecture beforehand. Therefore, the system is unaware of how to ensure its service (i.e., characterized by SLOs) within this highly dynamic environment. To that extent, we advertise our framework for collaborative edge intelligence as the missing piece: Edge devices are supervised by AIF agents, which ensure QoS and QoE through their EOSC model. Whenever the computing architecture is extended with new devices (i.e., scaled horizontally), existing models can be transferred to new devices, regardless of their device types being known. Apart from that, the leader continuously analyzes edge devices' capacity to comply with SLOs; in case some devices are excessively loaded or suffer from short-term network issues, IoT clients are reassigned to edge devices to optimize the cluster-wide SLO fulfillment.

6.4.2 Implementation

While the last part of the use case outlined the envisioned solution, not all of these aspects are implemented and evaluated; in this regard, we focus on the ideas presented in this chapter. This especially concerns the three contributions of the presented framework, i.e., the AIF-based model training, knowledge transfer between heterogeneous devices, and rebalancing of load according to environmental factors. Aspects such as bootstrapping of IoT and edge devices and leader node election (e.g., fog or edge) were already covered, e.g., by *Murturi et al.* [MD22, DM20]. The same applies to cloud-based distribution of video streams. An exception, however, are privacy-preserving stream transformations; for this, we make use of previously evaluated work [SMDD23]. To give our evaluation more rigor, we chose this over simulating a workload and its impact on SLOs.

Prototype

We share the Python-based prototype of our framework in a GitHub repository[2]; it contains all source code for implementing the three contributions, as well as the EOSC models for each device type. The core logic is separated into two classes: `Agent` and `FogNode`. These are the high-level loops executed in the main thread; all other processes (e.g., `AIF` or `VideoProcessor`) run in detached threads. The central library that is applied for training and updating BNs, as well as running inference queries, is again *pgmpy* [AT23]. *pgmpy* offers ample support of BNL techniques; however our choice is also motivated by personal preference – the framework's performance must be analyzed under different libraries (e.g., as done by [ZCC+23]). To improve the portability of our framework and simplify distribution, we provide a docker image[3]. The image exposes multiple env variables for configuring the solution, e.g., forcing the `Agent` to create a EOSC model from scratch or disabling `AIF` entirely.

The source code also contains the framework for privacy-preserving stream transformation and the ML models for face [Lin22] and age detection [RTG15]. To improve the reproducibility of results, we cancel out irregularities in the video streams by processing prerecorded videos; these are contained in the same repository. To simulate redirecting IoT devices within the cluster, it thus suffices to open/close processing threads on the edge devices; this simplifies networking. The `Agent` can thus reconfigure the stream assignment immediately, at the end of every `AIF` iteration. Because the use case is focused on video streaming and the number of frames per second (*fps*) that are transferred, each iteration lasts up to 1000ms.

Practical Limitations

Merging BN, as presented in Section 6.3.2, is only possible under the specified conditions, which are not always given during the AIF cycles. The number of states in a CPT, for example, is highly dynamic and extended as new data is received. To merge the EOSC

[2] github.com/borissedlak/FGCS, Last accessed: April 30, 2025
[3] hub.docker.com/basta55/workload, Last accessed: April 30, 2025

models under such circumstances, we provided a workaround: Instead of merging two BNs (m_a and m_b), we extend one of them (e.g., m_a). The device that trained m_b maintains a backup of the training data (d_b); this we use to update the CPTs of m_a through PARL[4], i.e., $m_{ab} = \texttt{PARL}(m_a, d_b)$. Notice, that this merges the conditional probabilities of the models, but not the structure; this remains an open question. While the resulting models are valid, we cannot assume that the original training data is always maintained.

Another limitation is that the DAG of the *model* cannot be updated frivolously through STRL; this triggers numerous updates within the CPTs of the BN, which are not supported by default in *pgmpy*. Although *bnlearn* [Scu10] promises these features, we require a package that can be embedded into our Python environment. Therefore, we make use of the following workaround: Instead of updating the DAG of model m_a according to new observations *batch*, we train a new BN with $data = batch \cup d_a$, where d_a reflects again the backup data. So internally, the AIF agent executes $\texttt{STRL}(model, batch)$ as $\texttt{PARL}(\texttt{STRL}(data), data)$, which likewise updates the CPTs with every execution. Solving this limitation will be a far-reaching achievement that requires dedicated future work.

Variables and SLOs

For the given use case, the agents consider device and application (i.e., video processing) metrics to construct EOSC models. Internally, BNL transforms metrics into model variables, which are used to evaluate conditional probabilities. Table 6.1 contains an overview of all captured metrics; each row contains a description, measuring unit, and if it can be set as parameter. Notice, that only parameterizable variables can be adjusted by AIF agents to optimize SLO fulfillment. For example, *pixel* and *fps* are video stream properties of the IoT device, which are reconfigured by edge devices according to agents' behavior. The leader node, on the other hand, can adjust the number of *streams* per device, which is out of scope for individual devices.

The EOSC (or EOSC-F) models can be applied in different computational tiers to ensure each tier's unique requirements; thus, their model variables might not overlap. The edge-based EOSC model contains the upper part of the variables, i.e., from *pixel* to *success*, whereas the cluster-based EOSC-F model treats the lower part. Notice that the metric's origin, i.e., if it was measured from system stats or the application, does not determine where it is used as a variable. From these variables, we construct SLOs that reflect the system state in terms of QoS and QoE. The AIF agent considers this classification when calculating *pv* and *ra* (recall Section 6.3.1). In Table 6.2, we present four SLOs that must be ensured during edge-based processing and one that is ensured by the cluster's leader node. To simplify the EOSC models, we include the SLO into BNL and remove the source variable, i.e., **distance** instead of *distance*.

[4]This functionality is natively offered by *pgmpy*; by default, the models are merged proportionally to the number of samples that m_a and d_b contain. This can be fine-tuned by adjusting the *n_prev_samples* parameter; we use this to prioritize new observations *batch* over existing conditional probabilities.

Table 6.1: Device metrics captured, which are turned into model variables by AIF

Name	Origin	Unit	Description	Param
pixel	IoT	num	number of pixels contained in a frame	Edge
fps	IoT	num	number of frames received per second	Edge
bitrate	IoT	num	number of pixels transferred per second	No
cpu	Edge	%	utilization of the device CPU	No
memory	Edge	%	utilization of the system memory	No
streams	Edge	num	number of IoT devices providing data	Fog
consumption	Edge	W	energy pulled by the device	No
network	Edge	num	network throughput per application	No
delay	App.	ms	processing time per video frame	No
success	App.	T/F	if a pattern (i.e., face) was detected	No
distance	App.	num	relative object movement between frames	No
slo_rate	Edge	%	combined SLO Fulfillment rate ($pv \times ra$)	No
device_type	Edge	enum	physical device type	No
congestion	Edge	num	network congestion that increases latency	No

Table 6.2: Extracted SLOs and their classification.

SLO	Condition	Tier	Type
network	$throughput < 1.6$ MB/s	Edge	QoS
in_time	$delay < 1/fps$	Edge	QoS
success	$success = True$	Edge	QoE
distance	$distance < 50$	Edge	QoE
slo_rate	$\max(slo\_rate)$	Fog	Both

We consider the presented SLOs relevant because (1) **network** ensures that the actual throughput does not exceed the bandwidth allocated to this application, (2) **in_time** makes sure that frames are computed within the available time frame, (3) **success** guarantees maximal privacy preservation, and (4) **distance** ascertains a smooth trajectory for tracked objects. The **slo_rate** reflects the cluster-wide SLO fulfillment. Notice that in the supplied video stream, there was always a face present, which means **success** can be compared against a ground truth.

Device Classification

Video processing is very dependent on the availability of GPU acceleration [SMDD23]; therefore, we apply multiple edge devices – with and without GPUs. All devices applied for this chapter are listed in Table 6.3; in the following, we call them by their ID. The other columns contain hardware characteristics and – complementarily – the original price of the device. A special instance is $Xavier_{CPU}$: while its physical hardware is equal to $Xavier_{GPU}$, we disabled the GPU acceleration (i.e., NVIDIA CUDA) to create another device type. Overall, our devices differ greatly in terms of computing capabilities (e.g.,

Table 6.3: List of devices used for implementing and evaluating the presented methodology

Full Device Name	ID	Price[5]	CPU [1,4]	GPU [0,2]	Σ
ThinkPad X1 Gen 10	*Laptop*	1800 €	Very High (4)	None (0)	4
Jetson Orin Nano	*Orin*	500 €	High (3)	High (2)	5
Nvidia Jetson Nano	*Nano*	150 €	Low (1)	None (0)	1
Jetson Xavier NX	$Xavier_{CPU}$	300 €	Medium (2)	None (0)	2
Jetson Xavier NX	$Xavier_{GPU}$	300 €	Medium (2)	Low (1)	3

missing GPU support or a highly superior CPU with 16 cores); nevertheless, as a whole, these devices compose the heterogeneous edge layer of the CC architecture.

As a prerequisite for transfer learning, we classify devices in a cluster according to their hardware characteristics. Although this process is dynamic, i.e., done repeatedly as devices join or leave or leave the cluster, we focus our evaluation on a scenario where the cluster contains all devices from Table 6.3, excluding Xavier_{GPU}; the latter will be the device joining the cluster. As discussed in Section 6.3.2, we classify these devices relative to each other according to their CPU and GPU capabilities; the results are contained in Table 6.3. To achieve the desired distance between the scalars, the CPU is aligned between $[1 \leq p \leq 4]$ and the GPU between $[0 \leq g \leq 2]$.

6.4.3 Evaluation Methodology

The implementation of the use case is thus set up for evaluation. To ensure a solid foundation for our framework, we will target each of the three pillars (i.e., the contributions) individually. The order in which they are evaluated resembles the one used throughout the chapter; this makes sense also from a logical point of view because transfer learning and stream offloading rely on the underlying AIF mechanism. In the three paragraphs below, we outline the evaluated aspects and motivate each question. Combined, this represents our evaluation methodology.

Active Inference Our main interest includes the executability of the AIF agent on edge devices and the extent to which the EOSC model improves the SLO fulfillment within the Edge. Because structure and parameter learning are recurrent factors in the evaluation, we will put emphasis on when they happen. Namely, our questions include:

A-1: Do MBs reduce the complexity of inference?

Increasingly large BNs require mechanisms to limit the complexity of a system; otherwise, resource-restricted edge devices may fail to execute the AIF cycle within an induced time frame. The MB, as a potential remedy, could achieve this.

[5]Prices adopted from sparkfun, accessed Jul 14th 2024

A-2: What is AIF's operational overhead?

Training and updating EOSC models directly on edge devices allows them to adapt quickly to system dynamics. However, any overhead introduced by AIF must not disrupt regular device operation, e.g., data processing.

A-3: How long require AIF agents to ensure SLOs?

To optimize SLO fulfillment, the agent must be able to infer adequate system configuration. However, there is no guarantee after how many AIF iterations the model will converge to the desired accuracy. Hence, we must provide an estimate for this.

A-4: Are the produced Bayesian networks interpretable?

Large-scale distributed systems, e.g., the CC, require trusted and reliable components as a solid foundation. Given that AIF can provide structures that are empirically verifiable, this promises to increase trust.

A-4-2: Is the behavior of AIF agents explainable?

Being able to understand an agent's decisions allows to justify (or empirically debug) its behavior, e.g., why the agent chose a certain device configuration at a specific time. If agents follow patterns, this also simplifies the configuration of hyperparameters.

A-5: What is the operational impact of including BNL in the AIF cycle?

BNL was identified as the dominant factor for the complexity of the AIF cycle; therefore, we must ascertain whether edge devices can perform BNL without limitations. Depending on the results, the two processes could be broken up into a federated learning approach, e.g., to execute sub-steps in the Fog.

A-6: Can changes in variable distribution be handled?

Real-world generative processes are not guaranteed to stay stable, small environmental changes (e.g., a new client) might suffice to change the SLO result. Nevertheless, these changes should be detected and resolved through AIF-based model training.

A-7: Can SLOs be modified during runtime?

In the CC, devices can be administered by entities that stand hierarchically above them; these can change their role in the architecture, or more simply, their SLOs. If a device could not adapt its existing EOSC model, it would have to train from scratch.

Knowledge Transfer After focusing on the training of EOSC models, we are mainly interested in how well the created models can be exchanged with other edge devices, and if this promises to improve the training time. Ideally, we would thus reuse existing knowledge instead of "rediscovering" it.

K-1: What is the SLO fulfillment rate of transferred models?

Transfer learning can provide ML models (i.e., specific for one device) to other devices. However, it is not guaranteed that a transferred model performs equally to a model specifically trained for a device. For example, the transferred model might be more likely to violate SLOs.

K-2: Can knowledge transfer achieve any speedup?

Transferring a trained model removes computational overhead (*A-2*) from the recipient; thus, it could decrease the overall energy dedicated to model training, most beneficial for resource-restricted edge devices. Furthermore, this could decrease the time required to ensure SLOs (*A-3*).

K-3: Can merged models decrease the FE compared to choosing a single one?

Models with low FE can infer SLO-fulfilling system configurations with higher accuracy. Exchanging knowledge within the cluster can include the combination of multiple eligible models. However, can such combined models interpret observations with less surprise compared to a single transferred model?

Stream Offloading To optimize their SLO fulfillment, intelligent edge device continuously adapt their environment. However, for environmental factors that are out of their scope (e.g., network failures or hardware limitations), the device cluster can be the remedy to compensate for these issues. In this context, we want to determine whether the SLO fulfillment of individual devices can be recovered through collaboration.

S-1: How is load distributed among resource-constrained devices?

The Edge, as one CC tier, allows clients to request services from nearby edge devices; however, this fosters situations where load is highly unbalanced within the system. This might cause resource-restricted devices to fail their service; once this is detected, the load must be rebalanced within the system.

S-2: Can the CC hierarchy optimize local SLO fulfillment?

Depending on the scale of SLO failure, individual devices may be incapable of recovering their service through local reconfiguration. Nevertheless, higher entities in the CC (e.g., cluster) can evaluate and resolve this by employing their own SLOs.

6.5 Results and Discussion

In the following, we evaluate the prototype according to the presented methodology. We structure our results according to the three contributions and the evaluation order in Section 6.4.3; based on the results, we pose derivative questions for future work. At the end of this section, we take a bird's-eye view to look at the results as one coherent framework and discuss the applicability of our approach.

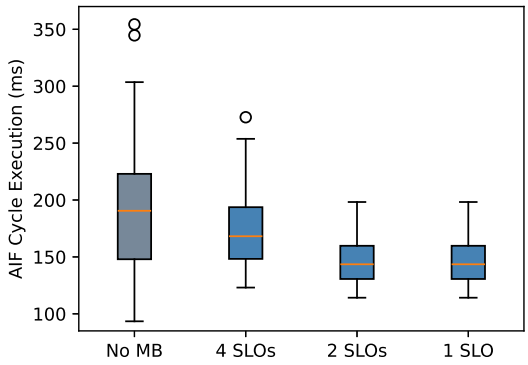


Figure 6.6: Duration of AIF cycle depending on the application of an MB and the number of SLOs (*A-1*)

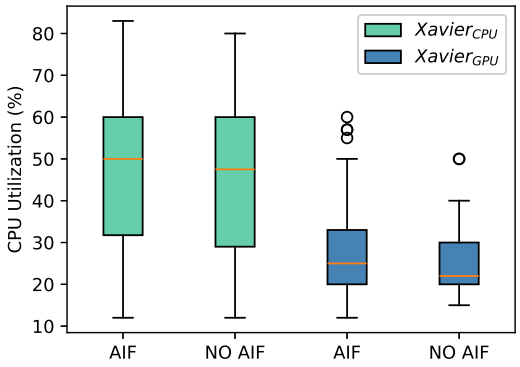


Figure 6.7: Overhead introduced by AIF when operating on $Xavier_{CPU}$ or $Xavier_{GPU}$ (*A-2*)

6.5.1 Active Inference

A-1: Do MBs reduce the complexity of inference?

To show whether an MB can decrease the AIF cycle duration, we focus on one of its subparts – the inference. We modify the implementation of Algorithm 6.1 (Lines 2 & 8) to execute `INFERENCE` either (1) on the entire BN including all 4 SLOs, (2) the MB including 4 SLOs, (3) the MB with 2 SLOs, or (4) the MB with 1 SLO. Then, we execute the AIF cycle on *Laptop* and capture the running time of each configuration over a duration of 10 min; this produces 600 observations for each experiment. Figure 6.6 visualizes the time that *Laptop* requires for performing `INFERENCE`, given the different MB sizes.

We observe: (1) applying an MB reduces the median execution type significantly, i.e., from 191 ms (grey) to 159 ms (blue) for 4 SLOs, and (2) decreasing the number of SLOs gradually reduces the execution time further. We thus conclude that MBs can reduce the complexity of VE (*A-1*).

A-2: What is AIF's operational overhead?

To evaluate AIF's overhead, we use pre-trained models for $Xavier_{CPU}$ and $Xavier_{GPU}$. Each device processes 6 video streams. We measure the CPU load (%) of the two devices with one of these two configurations: (1) AIF enabled, and (2) AIF disabled. We capture the load over 10 min; this produces 600 observations for each experiment. In Figure 6.7, we show the CPU load of $Xavier_{CPU}$ and $Xavier_{GPU}$. The left bar of each device shows the load when operating with AIF and the right one without AIF.

We observe: (1) the CPU load is clearly decreased by videos processing on GPU, $Xavier_{GPU}$ with AIF enabled presented a 24% lower load than $Xavier_{CPU}$, and (2) the AIF background process introduced a computational overhead of 3% for both devices

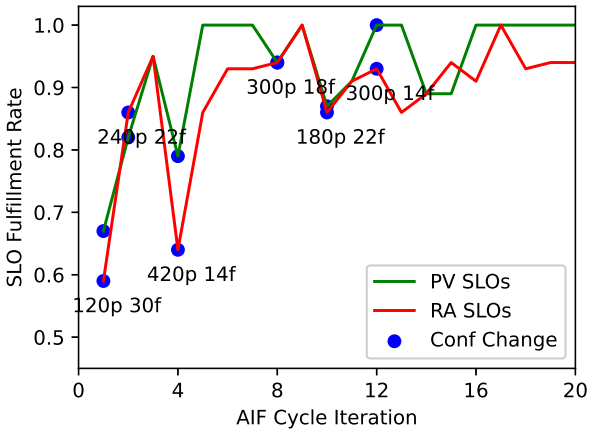


Figure 6.8: SLO fulfillment (*pv* & *ra*) when operating on a blank *Laptop* client (*A-3*)

(left vs. right bar). Overall, this provides an estimate of the general overhead (*A-2*); however, whether this is acceptable depends on the use case.

A-3: How long require AIF agents to ensure SLOs?

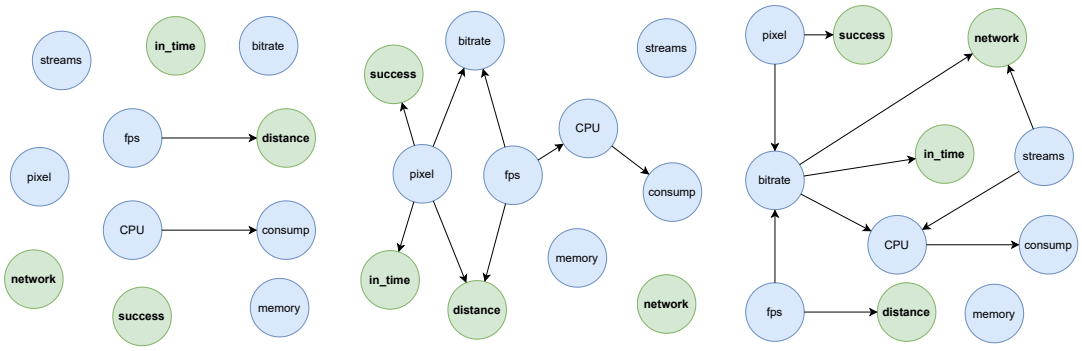
To evaluate the time to train a EOSC model, we count (1) the number of AIF iterations that the agent requires to arrive at a (nearly) optimal device configuration, and (2) how often the agent changes the configuration. The model is trained from scratch; therefore, the AIF agent (i.e., executed on *Laptop*) trains the model over 20 cycles and reports after each cycle (3) the SLO fulfillment according to the selected device configuration. We present the results in Figure 6.8: The green and red lines represent the SLO fulfillment (*pv* & *ra*); whenever the agent reconfigures the edge device, we print a blue dot for both lines in the graph.

We observe: (1) the agent requires roughly 7 cycles to converge to a configuration that satisfied SLOs with more than 90%, which is maintained in later rounds; (2) this state is reached after 3 reconfigurations; and (3) *pv* and *ra* showed similar trends in this example. Thus, we answered how long an AIF agent requires to provide an acceptable configuration (*A-3*), both in terms of AIF cycles and the number of reconfigurations.

A-4: Are the produced causal graphs interpretable?

To discuss the interpretability of created causal structures, we compare the DAGs produced by `STRL` and highlight at which stage the graph can be empirically explained. We will not consider specific metrics here but interpret the DAGs according to our expert knowledge. On *Laptop*, we train a EOSC model from scratch and extract the DAGs after {1,3,5,10} rounds of BNL. Thus, we want to show how the AIF agent discovers (ideally) causal relations between model variables. The results are visible in Figure 6.9: SLO variables (see Table 6.2) are colored in green; regular variables in blue.

We observe: (1) all SLO variables are influenced by variables that the AIF agent can



(a) DAG after 1 round (b) DAG after 3 round (c) DAG after 10 rounds

Figure 6.9: Progress of the DAG after {1,3,10} rounds of parameter training when creating a model with AIF on *Laptop* (*A-4*)

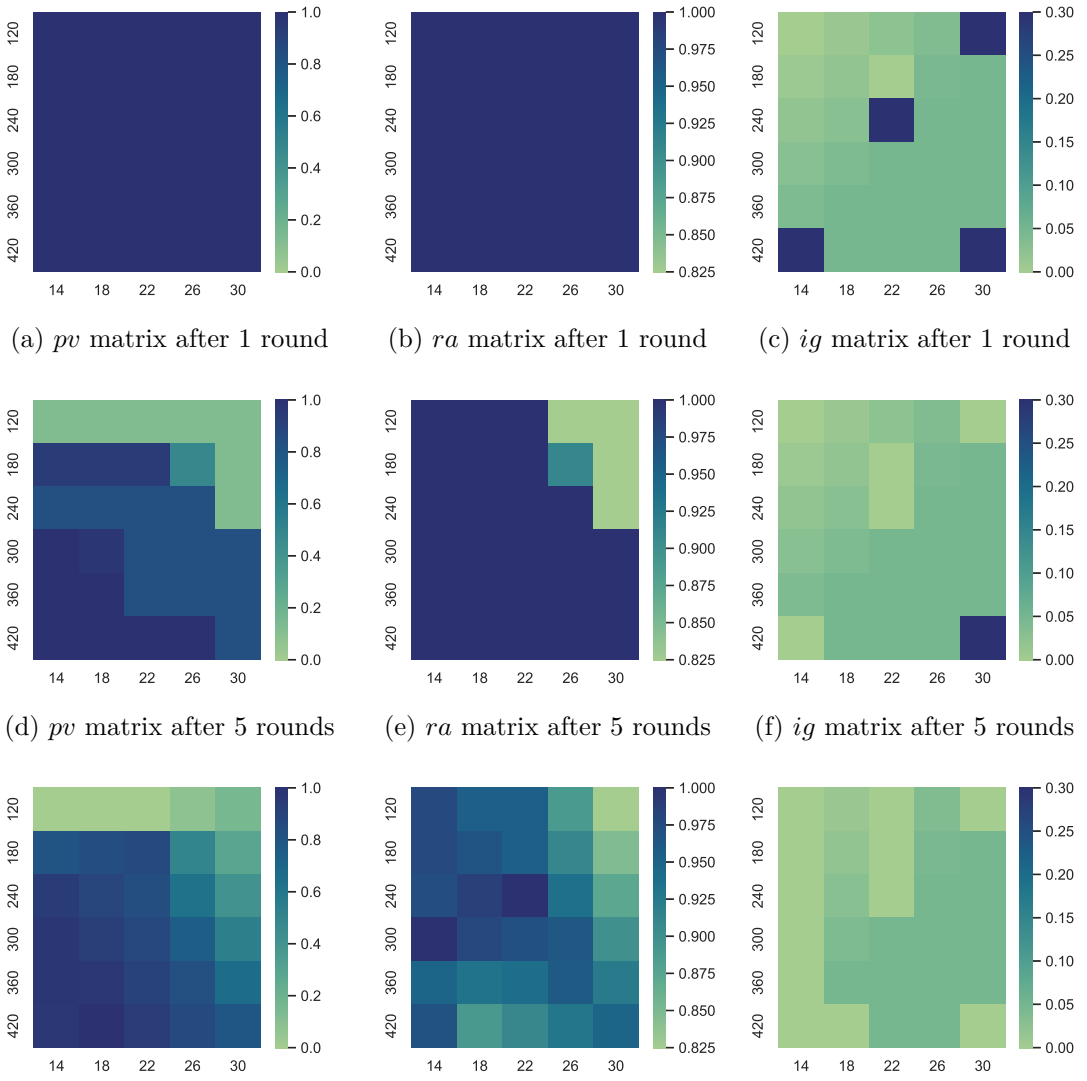
control, and (2) memory was the only variable that could not be related to others. After studying the graphs carefully, we could not detect any edge that appears counterintuitive to us; however, this does not prove that they are indeed causal. In total, we claim that the created graph is coherent and the links are understandable (*A-4*), but it requires sophisticated experiments to prove causality for each edge.

A-4-2: Is the behavior of AIF agents interpretable?

Complementarily, we were interested in how the behavior of the AIF agent could be interpreted. In Figure 6.10 we present three matrices for each behavioral factor (i.e., pv, ra, and ig). We executed the AIF agent on *Laptop* and extracted the matrices after {1,5,50} iterations. The first row presents the agent's initial assumptions on how the parameters are related to SLO fulfillment (pv & ra) and which rows provide the most insight (ig).

We observe: (1) the ig is initially high at corner points in the parameter space (as discussed in Section 6.3.1), which are visited in the first AIF iterations – this is evident because at round 5 only one cell with $e = 0.3$ remains; (2) the interpolation improves as transitions in the heatmap become smoother (from top to bottom); (3) the highest SLO fulfillment is at $pixel = 300, fps = 14$; and (4) the agent develops clear preferences in terms of pv (i.e., bottom-left corner), while the optimal ra is located in the center of the parameter space. Areas to avoid would be, e.g., $pixel = 120$, because image detection requires more detail, or $fps > 22$ because the processing time frame shrinks. Overall, we argue that the visualizations allow understanding the agent's behavior (*A-4-2*).

A-5: What is the operational impact of including BNL in the AIF cycle? To answer whether BNL can be applied on regular edge devices, we train a EOSC model on $Xavier_{GPU}$ and measure the execution time of `STRL` and `PARL`, i.e., the BNL sub-steps from Algorithm 6.2. In Figure 6.12 we visualize the execution time of `STRL` and `PARL` over 100 AIF iterations, respectively 1.5 min of operation. We observe: (1) `PARL` requires



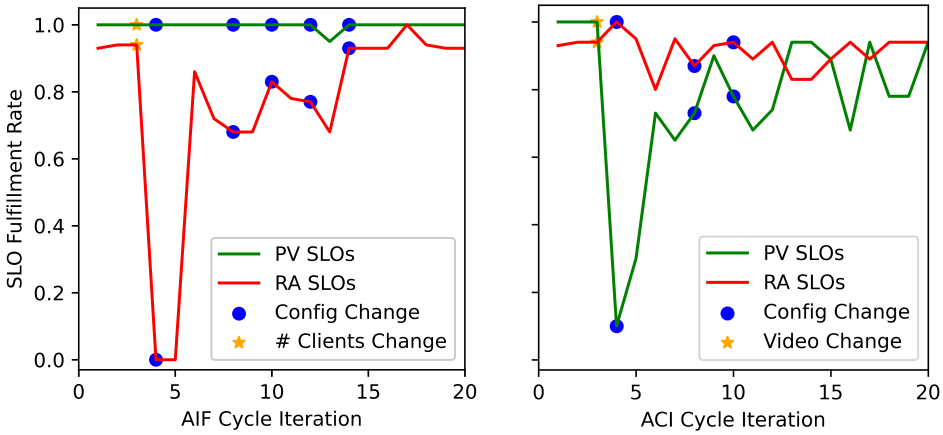
(a) *pv* matrix after 1 round (b) *ra* matrix after 1 round (c) *ig* matrix after 1 round

(d) *pv* matrix after 5 rounds (e) *ra* matrix after 5 rounds (f) *ig* matrix after 5 rounds

(g) *pv* matrix after 50 rounds (h) *ra* matrix after 50 rounds (i) *ig* matrix after 50 rounds

Figure 6.10: Behavioral factors (i.e., *pv*, *ra*, and *ig*) interpolated by the AIF agent to evaluate possible device configurations (*A-4-2*)

a stable runtime of around 250ms, (2) the runtime of STRL increases as more training data becomes available, and (3) running STRL after 100 AIF iterations took more than 20s. We conclude that PARL might be run on the employed edge device because it can be completed within less than 1000ms (i.e., the time frame for concluding the AIF cycle from Section 6.4.2). However, the runtime of STRL presents an obstacle because the AIF agent might thus have to skip iterations until the ongoing execution of STRL finishes. Hence, it would be advisable to perform STRL on another device (*A-5*) or find a way to



(a) Stream changes

(b) Video changes

Figure 6.11: Changes in the variable distribution caused (a) by higher number of video streams or (b) lower video quality (*A-6*)

decrease the runtime, e.g., by updating the DAG regardless of existing CPTs.

A-6: Can changes in variable distribution be handled?

Variable distributions can change due to various external factors; to evaluate how well the system can handle this, we either (1) simulate a peek usage time by increasing the number of processed video streams from 1 to 6, or (2) distort the video content with a Gaussian blur of 5px, which could resemble a foggy video setting. We measure the impact on the SLO fulfillment (*pv* & *ra*) over 20 AIF cycles and visualize to what extent the EOSC model is capable of restoring satisfactory (i.e., close to original) SLO rates. Figure 6.11 shows in both subfigures the SLO fulfillment rate of *Laptop*, when the disruptive factor was introduced (i.e., after 3 iterations), and at which points the AIF agent reconfigured the system (blue dots).

We observe: (1) after the stream change, *Laptop* took 11 AIF cycles (incl. 4 reconfigurations) to recover the SLO fulfillment, and (2) the information loss introduced by the video manipulation could not be recovered, although SLO fulfillment was improved as far as possible. Hence, we conclude that the system was able to adapt to changes in the variable distribution (*A-6*); however, only as long as the device can compensate for this factor. In fact, the **success** SLO could not be fulfilled after the video change took place because the agent could not increase the resolution sufficiently to recognize the faces.

A-7: Can SLOs be modified during runtime?

To simulate changing requirements, we modify the **distance** SLO from 50 to 20 (i.e., clearly stricter) and measure the SLO fulfillment rate before and after the modification. Additionally, we capture the surprise (Algorithm 6.1) to show if SLO outcomes reflected the expectations of the agent. Figure 6.13 shows in the upper part the SLO fulfillment

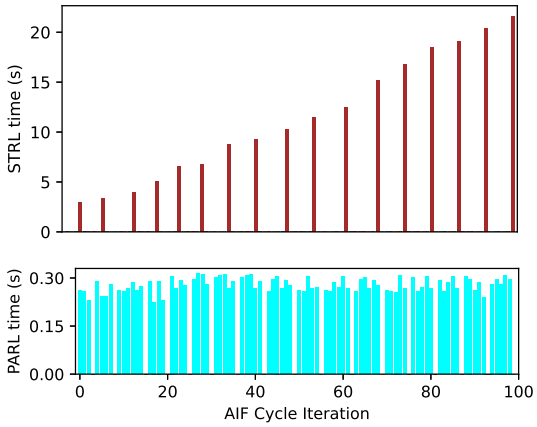


Figure 6.12: Duration of structure and parameter learning on $Xavier_{GPU}$ when training a BN from scratch (*A-5*)

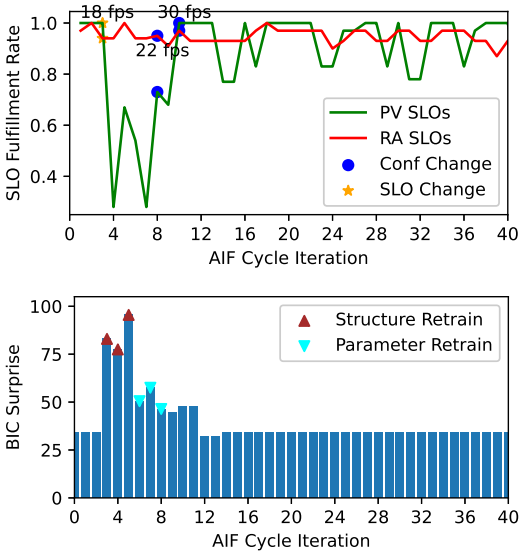


Figure 6.13: Impact of changing the **distance** SLO during runtime, combined with the surprise measured (*A-7*)

rate over 40 AIF cycles; the SLO changes after 3 iterations. The lower part shows the agent's surprise at each round and when `STRL` or `PARL` happen.

We observe: (1) after the SLO change, the agent experienced 9 rounds of high surprise, i.e., >> 35, (2) after 2 reconfigurations, the state prior to the SLO change was recovered, although final SLO rates (mean 0.91) are slightly below previous (mean 0.94), (3) to satisfy lower **distance**, the answer was to increase *fps*, and (4) the magnitude of the surprise was decisive for the decision between `STRL` and `PARL` (as envisioned in Algorithm 6.2). However, as known from Figure 6.12, `STRL` can exceed the AIF time frame multiple times; hence, the AIF agent is forced to wait for this process to finish. This could be solved, e.g., by offloading `STRL`. Hence, we conclude that the system was able to handle SLO changes during runtime (*A-7*).

6.5.2 Knowledge Transfer

K-1: What is the SLO fulfillment rate of transferred models?

Transfer learning promises to accelerate model training, but we must ensure that transferred models perform similarly to trained ones. For this, we assume $Xavier_{GPU}$ wants to join the cluster. According to Table 6.3, *Laptop* and $Xavier_{CPU}$ are eligible for providing their model, i.e., their *dc* (2 & 4) are the closest to $Xavier_{GPU}$ (3). Hence, we merge their EOSC models and transfer the result to $Xavier_{GPU}$. We compare the SLO fulfillment of the merged model with a separate run, where a model is trained from scratch. We place both runs into Figure 6.14; the blue line represents the combined model, and the grey

one was trained from scratch. Additionally, we indicate each time the agents changed the configuration.

We observe: (1) the merged model does not face substantial improvements of its initially high SLO fulfillment; (2) the agent required 14 rounds to arrive at a comparable SLO rate – this also matches Figure 6.8, where *Laptop* required 7 to 16 AIF rounds for training; and (3) the final rates are within the range [0.85,0.95]. From that, we conclude that results produced by the trained model are comparable to the merged model (*K-1*), and that KT could achieve a speedup of 14 rounds (*K-2*), assuming that the transferred model was available. Nevertheless, this is only valid for the given setup (i.e., these two devices); it is not possible yet to derive general implications of our approach.

K-3: Can merged models decrease the FE compared to choosing a single one? As discussed in Section 6.2, it is hard to estimate the FE of a model, but we consider the fact that surprise is bounded by FE. Although low surprise does not imply low FE, we use it as an indicator: We transfer a model to $Xavier_{GPU}$ (merged from *Laptop* and $Xavier_{CPU}$ as above) and calculate the surprise throughout multiple AIF cycles. This we compare against alternative runs, in which $Xavier_{GPU}$ uses one of the EOSC models of the other devices (from Table Table 6.3). Furthermore, we count the usage of `PARL`. The results are presented in Figure 6.15; each of the colored lines represents one of the respective models, which were copied to $Xavier_{GPU}$. The blue line, however, describes the combined model. The lower figure shows for each run when `PARL` was executed.

We observe: (1) the models trained on *Orin* and *Nano* produced initially very high surprise ($>> 50$), indicating that these models fit $Xavier_{GPU}$ the least; (2) nevertheless, the agent was able to improve these models and converge to an area where all 5 models provide similar surprise after 25 iterations; (3) the combined model provided initially the best values and only performed `PARL` twice; and (4) interestingly, although close to each other, the combined model produces after 25 rounds the highest surprise (33), while $Xavier_{CPU}$ reached 17. This shows, that the frequent retraining performed by the other devices (colored triangles in the lower graph) allowed the other models to surpass $Xavier_{GPU}$. This raises the question if it would be advisable to always run `PARL`, regardless of the surprise magnitude Combined, we can answer that the merged model had initially less surprising values (*K-3*); however, frequent retraining may achieve even better results.

6.5.3 Stream Offloading

S-1: How is the load distributed among resource-constrained devices?

To offload computations within the cluster, we aim to show how low-resource devices are relieved from excessive load. For this, we assume 25 IoT devices that are either assigned *Equal* to the edge devices or *Random*. As an indicator for maximum SLO fulfillment, we added *Single*, where each device processes one stream; Table 6.4 shows an overview of each scenario's assignment. After operating with *Equal* or *Random*, the leader node starts to optimize the environment, i.e., using the EOSC-F model to distribute the 25 streams

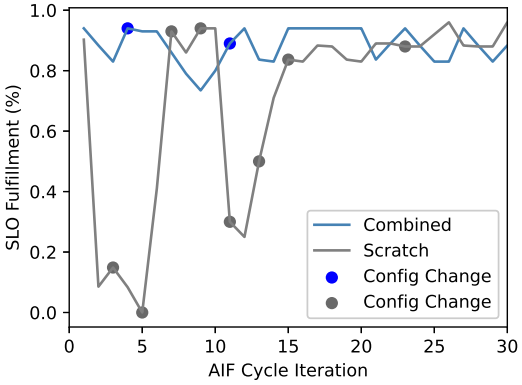


Figure 6.14: Difference in SLO fulfillment between an agent using a transferred model or training from scratch (*K-1* & *K-2*)

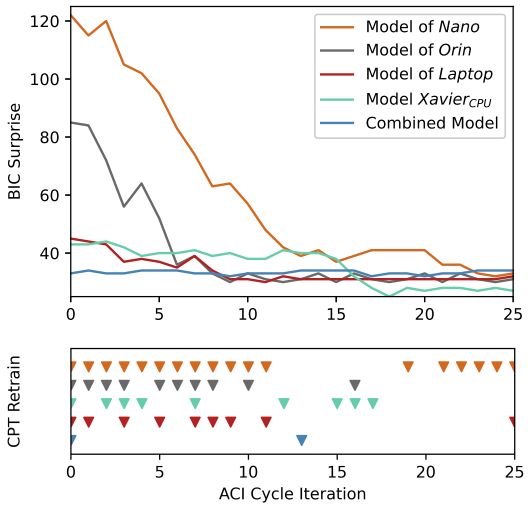


Figure 6.15: Surprise per batch on $Xavier_{GPU}$ with combined model or existing one. Paired with PARL frequency (*K-3*)

Device ID	*Single*	*Equal*	*Rand*	*Infer*
Laptop	1	5	4	9
$Xavier_{GPU}$	1	5	8	5
$Xavier_{CPU}$	1	5	5	1
Orin	1	5	4	9
Nano	1	5	3	1
Sum Σ	5	25	25	25

Table 6.4: Streams for scenarios

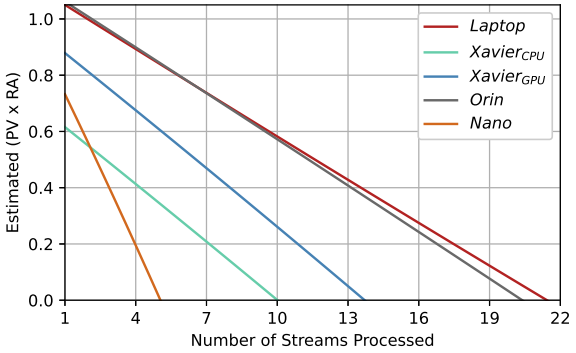
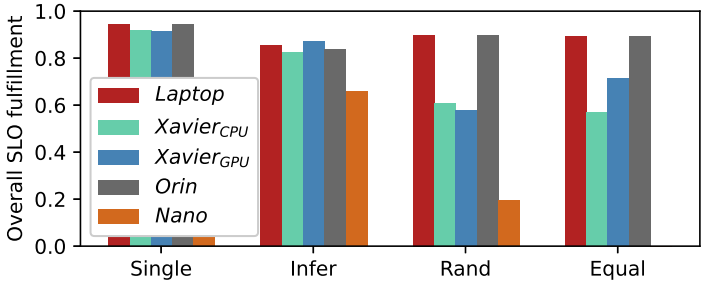


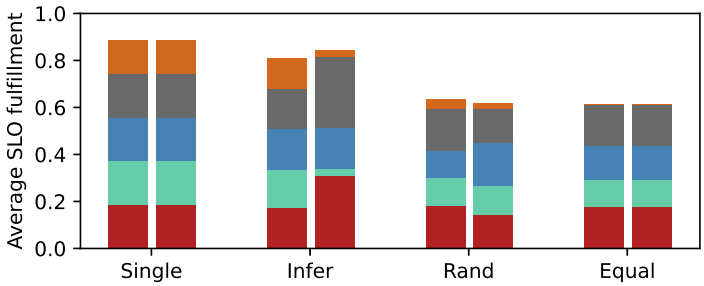
Figure 6.16: Regression between *streams* assigned to edge devices and respective SLO fulfillment rate ($pv \times ra$)

depending on the device capabilities (*Infer*). This new assignment is then provided to the edge devices. We thus perform offloading, e.g., *Nano* drops from 5 (or 3) to 1 stream. In Figure 6.17, we show each device's SLO fulfillment rate per scenario. The left bars of Figure 6.17b show the cluster-wide average of the SLO fulfillment and the right bar the weighted average according to the number of streams ($slo\_rate \times stream$). To get a feeling of the heterogeneous device capabilities, Figure 6.16 provides a regression function that shows how SLO fulfillment per device is impacted by the number of *streams*.

We observe: (1) the average SLO fulfillment clearly improved by using *Infer* (0.81) instead of *Random* (0.64) or *Equal* (0.60); (2) this is also reflected by the weighted average (right



(a) SLO fulfillment per device

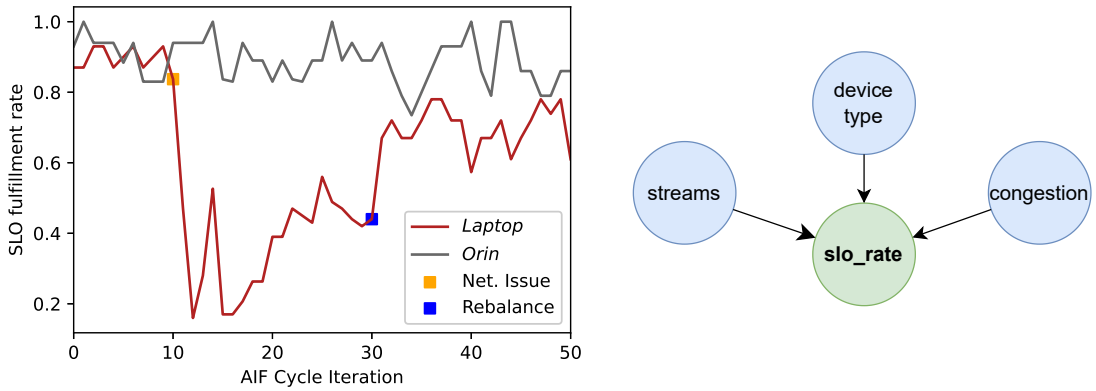


(b) Average and weighted average per batch

Figure 6.17: SLO fulfillment within the edge-fog cluster when distributing load according to *Infer*, *Random*, or *Equal*. *Single* is an upper bar for this device constellation (*S-1*)

bars of Figure 6.17b), which puts *Laptop* and *Orin* in focus that processed 9 streams each; (3) the weighted average of *Infer* comes close to *Single* (0.89), even though the cluster processed 25 instead of only 5 streams. From that, we conclude that the intelligent cluster was able to incorporate restricted edge devices (e.g., *Nano*) into the architecture (*S-1*), and that the overall SLO compliance improved by following our approach.

S-2: Can the CC hierarchy optimize local SLO fulfillment? To improve the SLO fulfillment whenever individual devices lack the required scope, we will resolve such SLO failures within the cluster. Therefore, we consider a condensed device cluster consisting of *Laptop* and *Orin*. *S-1* showed that they have comparable processing capabilities; therefore, it is fair to split 10 streams equally between them. Figure 6.18b provides the DAG internal to the EOSC-F model: Blue nodes are environmental factors, from which only *stream* can be configured (recall Section 6.4.2); *slo_rate* represents the common factor $f = pv \times ra$. We simulate network congestion[6]for *Orin* – which the leader node can evaluate through *congestion* – and redistribute the load according to the EOSC-F model, i.e., $Orin = 8, Laptop = 2$. Then, we compare the overall SLO fulfillment before and after offloading; the results are shown in Figure 6.18a. The two lines show the SLO fulfillment (f) of *Laptop* (red) and *Orin* (blue) over 50 AIF iterations; after 10 rounds, the network gets congested. In round 30, the cluster leader rebalanced the load according to its EOSC-F model; although it is possible to rebalance earlier, we decided to observe



(a) Impact of rebalancing

(b) Internal variable relations

Figure 6.18: Recovering network congestion by rebalancing the load within the device cluster according to the EOSC-F model; both devices initially processed 5 streams, 3 are offloaded to *Orin* (*S-2*)

the system behavior until manually rebalancing in round 30.

We observe: (1) the network issue crushed the SLO fulfillment of *Laptop* from around 0.9 to a minimum of 0.2 at round 15; (2) the edge device was able to improve the rate in the following 20 iterations by reconfiguration, until reaching a local optimum at 0.43. Further, (3) the cluster-wide SLO compliance was clearly improved through rebalancing, i.e., at round 15 the sum of $f_{Laptop} + f_{Orin}$ was 1.03, at round 30 it was 1.33, while at round 45 it rose to 1.54. We conclude that the intelligent cluster was able to resolve the introduced network issue (*S-2*) by redistributing the load according to the EOSC-F model. However, to draw general conclusions, we aim to consider a larger range of potential issues.

6.5.4 Result Implications

As a summary, we can report that (1) edge devices were gradually able to ensure local SLO compliance without prior knowledge; it took them 16 rounds to identify factors that impact SLO fulfillment and adapt the environment accordingly; the resulting SLO fulfillment aligns close to existing work [ZZL23], (2) the underlying causal structures and the transitions between device configuration were empirically explainable; this increases traceability and trust of ML models, and (3) shifted variable distributions were canceled out through continuous model retraining; edge devices took 9 rounds to interpret an unprecedented increase in demand, while SLO failures introduced by poor video quality could not be fully recovered. Further, (4) the causality filter based on MBs decreased the complexity of inference and sped up SLO evaluation by 17%, and (5) our

[6]Internally, we increase the processing delay according to *congestion*; this increases the overall latency and causes **in_time** to fail more likely. The EOSC-F model considers *congestion* as an environmental factor for Algorithm 6.3.

framework introduced a negligible CPU overhead of 3%, which makes it a suitable choice for resource-restricted devices.

It turned out that (6) BNL, or in particular structure learning, surpassed the given time frame for continuous model adaptation; nevertheless, parameter learning took only less than 250 ms and the overall training time appears promising compared to [KPS+20]. Thus, (7) models transferred between nearby devices could be continuously improved, even in cases where they fit poorly; this improves the reusability of models in the heterogeneous Edge, (8) the SLO fulfillment of devices with transferred models equaled the one of self-trained models; this accelerated the distribution of SLO-compliance models within one computational tier by up to 16 rounds, (9) rebalancing the load after a network error increased the overall SLO fulfillment from 1.03 to 1.54; this showed that collaboration within this tier increased the scope of SLO failures that could be covered. A closing observation is that (10) variable shifts showed the same effects on SLO fulfillment as low accuracy after transferring a model to an unknown device type. To our framework, they did not provide any fundamental differences, which is why they could both be resolved through continuous model training.

6.6 Related Work

This section provides recently published related works that discuss (1) the training and application of causal ML models on the Edge, (2) transfer learning approaches in the CC, and (3) methods of load balancing and computation offloading that are popular across the CC. Following that, we highlight for each of these fields the research gap that our work aims to fill.

Causal ML Training on the Edge *Sudharsan et al.* [SBA20] developed an `Edge2Train` model to analyze real-time data on the fly. With Edge2Train, Support Vector Machine (SVM) models are trained offline in edge nodes using real-time IoT. Adopting causality to Edge2Train can help converge the most efficient training models quickly. Diagnosing the root cause of performance degradation in the CC is a challenging issue, and *Chen et al* [CQH19] use causal inference (`CauseInfer`) mechanisms to pinpoint the root causes within the system. CauseInfer determines fault propagation paths that can be determined explicitly, without production systems being instrumented. A similar approach (called `Nazar`) is designed by *Hao et al.* in [HWH+23], where they apply mobile devices to diagnose root causes in distributed systems. Further, this approach enhances its training models through cause-specific adaptive mechanisms. Through experiments, Nazar confirmed that training models can be improved due to cause-specific adaptation while monitoring a large number of devices.

Lin et al. introduced `Microscope` in [LCZ18], a micro-service environment to diagnose the possible root causes of abnormal services in distributed systems through causal graphs. *Lin et al.* demonstrate that Microscope can construct a service causal graph in real time and infer the root cause of abnormal services. *Tariq et al.* present the What-If Scenario

Evaluator (WISE) tool in [TZV+08], which predicts the effect of potential configuration and deployment changes on content delivery networks (CDN). WISE initially learns causal relations among existing response time distributions. Based on the available datasets, it estimates possible future response time distributions. Finally, it allows network designers to express possible deployment scenarios without knowing how variables will affect response time.

There evidently exists work that identifies and applies causal understanding to ensure system requirements; however, with the exception of Nazar [HWH+23], they treat model training as a one-time process. Hence, drifts (or shifts) in the variable distribution stay undetected. Further, it is impractical to assume that sufficient training data is available to arrive at this causal understanding; this is also the shortcoming of Nazar. Contrarily, our approach, which focuses on ACI, is able to gradually create causal models over multiple iterations (i.e., as new training data becomes available), and continuously ensures model accuracy by updating beliefs according to prediction errors.

Transfer Learning in the CC *Goyal et al.* present MyML [GDB22], a hardware-friendly model transfer for edge nodes. MyML uses transfer learning to create small, lightweight, custom ML models based on user preferences. This approach is hardware-friendly, bottom-up pruning, which can be utilized on any mobile edge platform because of its ability to handle large, compute-intensive ML models. In addition, systolic array-based edge accelerators are introduced to prevent cloud interactions. *Wu et al.* present a novel approach to online transfer learning for both heterogeneous and homogeneous labels of multi-source domains [WWZ+17]. This approach is very efficient in online classification, and the weights are dynamically adjusted depending on the source domain. The work fits well into the CC due to the complex heterogeneity of devices within the system. *Hsu et al.* provide a clustering mechanism that considers the similarity of domains and tasks for transfer learning [HLK18]. They provided a similarity function for cross-task transfer learning that is based on similarities between domains.

Xing et al. introduced a model called RecycleML in [XSB+18] that enables multi-modality among edge devices, where knowledge is shared by transforming common latent features into their lower layers. Further, it provides task-specific knowledge transfer between models through the retraining of higher layers beyond the latent space shared by both models, thus reducing the need for labeled data. *Sharma et al.* proposed a knowledge transfer technique between edge devices to lower computational intensity without losing accuracy and convergence speed [SBZ18]. In this, the student network takes the knowledge from the teacher network to achieve this goal. Using an IoT testbed, *Kolcun et al.* [KPS+20] evaluated various machine learning classifiers' convergence speed and accuracy. These testbeds considered both data- and resource-specific constraints. The results of each local testbed's training models are transmitted to the gateway to minimize global training model overhead.

Transferring ML models is an important measure for relieving resource-restricted devices from training; teacher devices can therefore consider the context of the student to provide

a tailored model. This is an important feature since edge devices have heterogeneous characteristics; however, none of the presented works considered low-level hardware characteristics to identify potential teachers among nearby devices. Further, while it is possible to combine models, the presented techniques are not applicable to the causal structures that we require for decentralized SLO assurance. To that extent, our framework uses hardware classification to find adequate models within a device cluster and creates a tailored model by merging the conditional probabilities of BNs.

SLO-Induced Load Balancing and Offloading Elasticity is one of the most effective ways to ensure requirements of dynamic workloads by automatically provisioning or de-provisioning resources based on demand [DGST11]. `SLOC` is a novel elastic framework developed by *Nastic et al.* in [NMP+20], that allows users to provide and consume cloud resources in an SLO-native manner while guaranteeing performance. Its primary goal is to provide better support for SLOs by exploiting and advancing current elasticity management solutions. Further, *Furst et al.* bring elastic service principles from the cloud to edge computing [FFACP18]. They evaluated elastic and non-elastic services at the edge while processing images to latency SLOs, and noticed improved service provisioning through elasticity.

Tran and Kim introduce an edge serverless auto-scaling method based on traffic prediction that can be used against a Kubernetes cluster [TK24]. In their work, system resource usage is optimized to ensure latency SLOs. No additional resources are required to perform this operation; this optimizes the amount of available resources. *Hazra et al.* [HDAD23b] proposed efficient heuristic-based transmission scheduling and graph-based computational offloading (`TSCO`) through mixed linear programming to achieve energy efficiency and minimize latency. A single- and multi-task load balancing with a prioritization approach to computing Deep Neural Networks (DNNs) at the edge has been presented by *Karjee et al.* in [KPNS21]. In these approaches, prioritized tasks are distributed among IoT and edge nodes to balance energy, lower latency, and continue task execution without restarting the system. *Lim and Lee* proposed a load-balancing approach for distributing mobile devices tasks within a cloud-edge continuum using graph coloring [LL20]. Through this process, computing resources are scaled with increased edge resource utilization.

A trilayer mobile hybrid hierarchical peer-to-peer (MHP2P) model was proposed by *Duan et al.* in [DTZ+22] as a cloudlet for efficient load balancing strategy through mobile edge computing (MEC). MHP2P promises high reliability, scalability, and efficiency in service lookups. Moreover, there is a load-balancing scheme to ensure that MHP2P loads are evenly distributed between MEC servers and queries. In [Men21], *Menino* proposed efficient failure detection mechanisms for unstructured overlay networks. This approach aims to identify efficient neighborhood overlays, which dynamically identify and maintain each node in P2P networks.

SLOs are an efficient way for modeling and enforcing requirements; thus, high-level SLOs can be segregated and enforced at the respective CC component. Nevertheless,

the remaining question is whether the component has the required scope to recover SLO failures (e.g., by offloading computation), but it is impractical to evaluate SLOs in the cloud (e.g., MHP2P). Hence, ad-hoc hierarchical structures could provide a remedy, which *Menino* [Men21] are the only ones to use among the related work. However, they all assume prior knowledge of which variables impact SLO fulfillment. Contrarily, our approach (1) gradually increases the SLO scope by forming device clusters that can span the entire CC, and (2) evaluates causal relations among environmental variables to shift the load from impacted devices.

6.7 Summary

This chapter presented a novel framework for collaborative and distributed edge intelligence that ensures decentralized SLO fulfillment. It allows CC systems to disaggregate high-level requirements and enforce them at the component they concern; thereby, we create self-adaptive devices that themselves ensure dynamic requirements. For each component, the framework is able to develop causal reasoning between environmental factors and SLO fulfillment. Resource-restricted devices that cannot create this knowledge were able to exchange and combine causal models according to their hardware characteristics. This accelerates the onboarding of unknown device types and simplifies horizontal scaling within the Edge. Contrarily, any attempt to achieve this centrally would struggle with heterogeneous device characteristics, the induced network latency, and the communication overhead. To increase SLO coverage and the action scope, devices collaborated as clusters under the supervision of a Fog node; this forms higher-level components that can again supervise their own set of SLOs. Consequentially, the cluster was able to use its extended environment to resolve SLO violations, e.g., by offloading computation among pertinent devices. Erecting these hierarchical structures provides an accurate representation of observable processes and infers how to fulfill the intricate requirements of multiple computational tiers.

We provided a prototype of the framework for a distributed video transformation use case and evaluated it according to 12 aspects; the results showed the potential of our approach for ensuring SLOs throughout CC tiers. For future work, we aim to dynamically update the structure of presented models and evaluate limitations regarding the number of SLOs and devices. Further, this chapter builds heavily on (causal) relations between SLO fulfillment and environmental factors; however, to prove causality, dedicated experiments must be integrated into the framework. Once this is established, the framework will provide necessary causal links to tame requirements in the CC.

CHAPTER 7

Conclusion

This chapter concludes the thesis. First, in Section 7.1 we give a brief summary of the covered topics and the presented contributions. Then, in Section 7.2 we revisit the research questions posited in the introduction. Finally, in Section 7.3 we give a brief outlook on future research directions and upcoming research challenges.

7.1 Summary

The ubiquity of IoT devices has heralded a transition of processing resources to the Edge of the network, where data can be processed under tight resource constrains. However, this has not shown to replace traditional Cloud computing; instead, computing tiers are merged into a cohesive platform – called the Computing Continuum (CC). While the CC promises unprecedented computing capabilities, it also increases the complexity for orchestrating applications. Contrarily to the traditional Cloud computing, processing requirements – formulated as Service Level Objectives (SLOS) – cannot be evaluated centrally due to the induces communication overhead. At the same time, Edge devices struggle to fulfill SLOs because of their limited hardware capacities, leaving an entire processing tier without adequate measures to ensure local SLOs.

In this thesis, we offer a framework for autonomous orchestration of CC system. First, in Chapter 2, we give an overview of contemporary research on the CC, including open research challenges and promising application areas. To support these use cases, we discuss (1) what types of SLOs are needed to constrain various aspects of CC architectures, and (2) how the SLO fulfillment can be ensured throughout external perturbations. Together, this envisions a behavioral model for context-aware service orchestration at any hierarchy, fostering collaboration between services to ensure higher-level SLOs.

While in some occasions it might be possible to recover the SLO fulfillment by provisioning more resources, making it the default behavior is too rigid. To that extent, Chapter 3

develops the idea of multi-dimensional elasticity strategies that are not fixed to one predetermined elasticity strategy, but can choose how to optimally adapt the system. To make this more tangible, we present a detailed use case in which we aim to control data gravity and data friction – two undesired concepts that occur when processing IoT data. To control data gravity and data friction, we provide a conceptual architecture of a MAPE-K framework that analyzes multiple sensory observations for evaluating SLOs, while choosing elasticity strategies according to the current context. In this case, this can mean to compress data if the quality allows it, or scaling resources if they are not completely depleted. While this scenario was not evaluated experimentally, this reference architecture greatly helped for implementing the latter chapters.

While intuition or expert knowledge might be a way to find the right elasticity strategy in a particular context, this gives no formal guarantee that this strategy will actually have the expected effect. Hence, in Chapter 4, we looked into ways to estimate the effects of elasticity strategies directly from data, which gives a likelihood of which action would provide the most utility. To implement this, we used processing metrics to create the most likely variable structure – encoded in a Bayesian Networks (BNs). We implemented this methodology for a distributed video processing use case, where a video should be transformed and streamed to a consumer under latency and energy SLOs. Our results showed how the trained BN could be used to infer informed scaling actions, e.g., set video $resolution = 240p$ because this likely ensures $latency < 20ms$ while consuming $energy < 10W$. While variable relations in the BN are not necessarily causal, our methodology repeatedly found the optimal actions under changing SLO thresholds.

In Chapter 5, we present a series of orchestration mechanisms that use the created BNs: (1) to optimize the microservice deployment in a distributed CC architecture, we analyzed dependencies between services by merging their Markov blankets (MBs) – a minimum representation of relevant variables in the BN. Consequently, our experiments showed to provide the optimal service deployment. Further, (2) to simplify the SLO definition for distributed components, we inferred lower-level thresholds for backend services according to higher-level SLOs, e.g., maximize streaming quality. In our evaluation, 12 microservices did ensure high-level SLOs whenever possible. The exception are cases with contradicting high-level SLOs, e.g., minimize energy and minimize latency; in such cases, our method reported these conflicts to the stakeholder. Lastly, (3) to optimize the global SLO fulfillment between heterogeneous device, we estimate the impact of shifting computation between devices, i.e., how both sides are affected by exchanging load. We evaluated this for a platoon of autonomous vehicles, where perception services (e.g., object detection) were shifted to less utilized vehicles whenever they could not be run locally.

The methodologies in this thesis are very dependent on the quality of the BNs; if these models would become inaccurate, any inferred adaptation would likely fail to show the desired effect. To continuously ensure model accuracy, Chapter 6 provides an ecosystem for continuously training BNs; hence, it provides the backbone for all other methodologies presented. Our lifelong learning strategy is fueled by Active Inference (AIF) – a curiosity-based concept from neuroscience – where agents seeks to model their

(processing) environment. Our experiment showed how AIF agent can ensure local SLO fulfillment by training BNs for processing services. Agents collaborate in two ways: (1) by exchanging BNs between themselves to speed up the onboarding of new devices, and (2) by creating hierarchical structures, where leader nodes optimize the SLO fulfillment of their member nodes. In a series of experiments we showed that the trained BNs are empirically interpretable and allow to explain the behavior of agents at a particular time – greatly boosting the trustworthiness into any inferred scaling action.

7.2 Research Questions

In Section 1.2 we posed three fundamental research question that have driven the research in this thesis. Their purpose was to (1) ensure that services are always orchestrated based on accurate and informed decision, (2) find elasticity strategies that suit the given context and quantify their impact on SLO fulfillment, and (3) optimize global SLO fulfillment by analyzing the dependencies and interactions between services. We now discuss the research questions and contextualize the contributions of this thesis.

RQ.1 How to continuously assure the accuracy of service orchestration models so that reactive elasticity strategies provide maximum utility?

This thesis mainly discussed one instance of a model that was used for service orchestration: Bayesian Networks (BNs). To ensure their accuracy despite concept drifts or temporary perturbations, it requires lifelong learning mechanisms that counter perturbations whenever they occur. Hence, model training cannot be a one-shot process. While there exist multiple ways to ensure model accuracy through continuous retraining, in Chapter 6 we chose Active Inference (AIF) for this task; in the following, we elaborate why: Contemporary Reinforcement Learning (RL) strategies often use simple exploration mechanisms, e.g., ϵ-greedy, where an initial high exploration rate depletes as the training converges. Contrarily, AIF uses two fundamental concepts to ensure accuracy over time: (1) it uses *surprise* to quantify the discrepancy between the expected and actual outcome; however, instead of avoiding these areas, the agent focuses on these areas and tries to improve its understanding. This is because (2) the agent uses a formal representation to express the potential model improvement when taking a certain action; for every action it compares the pragmatic value (e.g., expected SLO fulfillment) with the information gain (i.e., uncertainty in the model). While RL-based orchestration mechanisms often uses model-free training [GMP+21], AIF aims to ensure an accurate model.

As the processing environment changes over time, e.g., anomalous CPU load is introduced in the background, AIF will search for the reason why a certain configuration cannot fulfill SLOs anymore. For instance, the agent might detect that the additional CPU load was caused by a background thread updating the system; by incorporating the *update interval* into the sensory state of the BN, it can resolve this confounding variable when inferring a scaling action. Notably, these factors can also be discovered during runtime, the only prerequisite is that they must be tracked by the AIF agent as metrics.

RQ.2 How to efficiently choose between elasticity strategies by quantify their impact on both the SLO fulfillment and underlying processing hardware?

The generative models trained by AIF – in this case BNs – have the fundamental advantage that the expressed relations can be empirically verified, something that is generally not given by Neuronal Networks (NNs). Hence, BNs can also provide insights to stakeholders on *why* a certain elasticity strategy is useful in a specific context. As such, BNs can be used to create a reactive behavioral model, as envisioned in Chapter 2, where agents estimate the impact of adjusting different system variable, e.g., how would SLO fulfillment change when decreasing streaming quality or provisioning additional resources. By comparing the expected utility of these actions, the AIF agent identifies variables that serve well as elasticity strategies, which are uses to recover SLO fulfillment when necessary. One fundamental design choice here is whether to design the action space discrete of continuous; while in Chapter 4 we started off discrete, in Chapter 6 we investigated continuous actions. Apart from the different representation, the main difference lies in the granularity of the inferred actions. For instance, a continuous scaling action could infer that the optimal streaming resolution would be around $662p$, while a discrete one is bound to a limited amount of bins[1]. We found that the design choice – favoring discrete, continuous, or even hybrid relations – is very dependent on the situation: while discrete relations can be more efficient at training and inference, continuous relations may allow inferring actions that are closer to optimal.

To quantify the precise impact of computing services on the underlying processing hardware, we found that it is possible to extend any services' BN with the respective variables. This means, that by adding the hardware-related variables (e.g., energy, or CPU/GPU load) to the BN learning, we would also be able to infer what would be the expected hardware load under a certain configuration. For instance, when running a video processing service with a $resolution = 720p$, what would be the expected energy consumption and the claimed amount of CPU load. From this example, we can conclude that: (1) this allows to find system configurations that minimize energy consumption, which is highly needed given the exploding consumption of computing systems in the last years [ZKQ+24], and (2) this greatly helps when co-locate multiple computing services at a single device, i.e., for each service the required resources can be precisely estimated so they may not end up cannibalizing resources needed for other services.

While runtime metrics from services and the underlying hardware proved essential to form behavioral BNs, the complexity during training and inference increases with every variable. Hence, we were challenged to ensure that inference does not only provide accurate results, but also runs efficiently. To maintain a lean view on the factors that impact SLO fulfillment – while filtering out such that have not shown any – we applied the concept of the Markov Blanket (MB), which was first introduced in Chapter 3 and then systematically used throughout the thesis. The MB reduces a system's variables to those that either impact the internal state (i.e., in this case the SLO fulfillment), or

[1] A common example for these bins would be the resolutions offered on YouTube, e.g., 720p, 1080p.

those that are impacted by the internal state. In Chapter 6, filtering the MB has shown to decrease the time for inference, while decreasing the number of variables that must be tracked during runtime; hence, deciding more efficiently on an elasticity strategies.

RQ.3 How to model the interactions and dependencies between microservices to estimate the impact they have on each other's SLO fulfillment?

While the mechanisms developed for **RQ.2** have shown to ensure the SLO fulfillment for an individual component, e.g., a single processing service in the Computing Continuum (CC), the CC is composed of a multitude of services that have reciprocate influences on each other. To that extent, we investigated how BNs could reflect the conditional probabilities between processing services. To avoid testing n^2 service pairs for dependencies, we considered the microservice architecture: services usually form sequential pipelines, hence, we only have to evaluate the dependencies between services that directly interact, e.g., by exchanging data. In Chapter 5 we identified two approaches for this: (1) combining BNs that were trained independently according to their variable distributions, or (2) training a composite BN from one combined data set. While we did not compare the complexity of these two approaches, this poses an interesting question for future work, in particular, considering the complexity from exchanging metrics or model updates in a network. Using the composed BNs, we showed how to estimate the impact of their elasticity strategies on dependent services. This proved particularly important for microservice pipelines – we found that subsequent services might not be able themselves to ensure their local processing SLOs, but would have to request a certain service level from their predecessor. For instance, if a stream consumer requires a certain video resolution under a latency boundary, the remaining components – video provider and processing – must be aligned with these goals. As a consequence, we started constraining all dependent components accordingly, which showed to ensure global SLO fulfillment.

Constraining all parts of an application according to high-level objectives showed to greatly improve SLO fulfillment – matching the vision of *deepSLOs* described in Chapter 2. At the same time, it revealed an inherent problem with SLO-based orchestration: how to deal with conflicting requirements? For instance, if stakeholders wished to minimize both latency and energy consumption, this inevitable leads into a conflict. We find it is essential to highlight such cases to stakeholders, so that their requirements can be refined. This is possible using our methodology; consequently, the conflicts can be resolved by assigning a weight to both SLOs, i.e., representing their severity, so that one or multiple of them can be traded off. The result can still be optimal under the given SLOs.

7.3 Limitations & Future Work

Considering the contributions presented in this thesis, there is strong evidence that AIF is a fitting solution for achieving autonomous orchestration of CC systems. However, to maintain a clear scope, we excluded numerous challenges for future work that were not at the center of the posed research questions. To improve the presented concepts and help them transition to state-of-the-art, this section outlines four research directions that we

would like to highlight, namely: (1) to underline the generalizability of our approach, it must be evaluated in a large-scale testbed; (2) to clarify the fundamental differences with RL-based techniques, our AIF-based methodology must be contrasted further, including when to prefer which approach; (3) to ensure fast training and inference of BNs on resource-restricted devices, the methodology must be optimized even further; and (4) to increase the trust into inferred scaling actions, we must formally prove the causality of edges in the BN. In the following, we elaborate these points in more detail.

Evaluation in Large-Scale Testbed

Developing and evaluating a prototype in a physical processing environment often provides more entropy than running it solely in a simulation environment. To that extent, the methodologies presented in this thesis, from Chapter 3 up to Chapter 6, have all been experimentally evaluated in a physical testbed, often using Edge devices like Nvidia Jetson, or Edge servers with stronger GPUs. However, setting up a physical testbed and maintaining it throughout multiple research papers present an extraordinary effort; hence, we only used up to 5 devices per experiment. While simulations may be a good start to evaluate our ideas in larger setups, e.g., with more than 1000 nodes, we would only achieve the desired rigor by using a physical testbed of that size. While this was out of scope for this thesis due to the complexity to handle such an environment, we are very keen on evaluating our approach in such a large-scale environment.

Extensive Comparison with Reinforcement Learning

Throughout the research conducted for this thesis, a frequent comment we received was that AIF has close resemblance with RL. Consequently, we often pointed our that the approaches are not mutually exclusive, but can be used complementarily, as also reported by other researchers [TMSB20, FDK09]. To that extent, further research is required to define a clearer margin between these approaches and answer when to prefer one, or when best to combine them. As a matter of fact, this work was embedded in an open research gap; due to their novelty, many of the envisioned orchestration mechanisms have not yet been implemented by other researchers, for example, with RL. Hence, there were limited possibilities to compare our approaches with a larger amount of baselines. To that extent, we have started to provide these baselines ourselves, for example in our latest work [SMR+25] we started from a RL-based methodology, which we intend to extend in the next iteration with a AIF-based agent. Thus, bit by bit, we aim to compare our results with custom baselines or other upcoming orchestration mechanisms.

Optimize Bayesian Network Learning & Inference

While Cloud servers usually dispose of an abundance of resources, Edge devices, located at the read end of the CC, have clear hardware limitations. To that extent, the methodologies in this thesis must be tailored to the weaker end of the device spectrum; otherwise, Edge devices could end up incapable of performing decentralized decision-making, revoking

their gained autonomy. Although in Chapter 6 we analyzed the overhead of training BNs and the impact of MB on the inference time, this requires further experiments on different hardware, ideally in a larger-scale testbed, as discussed. Potential improvements that we can envision are: (1) pruning BN variables that do not show to have an impact on SLO fulfillment, (2) comparing the results of different algorithms for structure and parameter learning, (3) comparing the performance impact of using discrete, continuous, or hybrid relations, and (4) comparing our methods for merging BNs from subgraphs. Future work will gradually introduce these improvements to our methodologies.

Proving Causal Relations

This thesis used BN learning to extract the most likely structure for data; while there is a high probability that extracted relations are causal, there is no formal guarantee to this. Ultimately, edges that are proved to be causal provide stronger guaranteed to the model, improving its quality. To make CC systems more predictable, we declared it our goal to find causal implications between systems [PSDD24], which we did formally not achieve yet. While our evaluations throughout the thesis have shown that the created BNs are indeed empirically verifiable, we envision to apply dedicated experiments that can prove the causality of relations. However, this raises the question of the expected overhead for causal testing, which must be performed on resource-restricted devices and thus opposed the aforementioned optimization challenges. In any case, this requires further work to contrast the qualities of the created models with a reasonable training overhead.

Übersicht verwendeter Hilfsmittel

Throughout the research conducted for this thesis, no generative AI (GenAI) tools have been used to author contents (e.g., text, images, etc); this also excludes generating texts and rephrase them with GenAI. A minor exception to this is the German chapter "Kurzfassung" – the direct equivalent of the abstract – for which we used DeeplPro[2] to translate the text; the revision was again done manually, without any help of AI.

However, GenAI tools, like ChatGPT, have also shown to be of great aid [ADF+25] for achieving a first impression on a novel topic. As such, GenAI has been used over the course of some chapters to gain a quick overview into a certain research aspect, e.g., by asking ChatGPT[3] to explain "the core differences between Reinforcement learning and Active Inference". However, as pointed out before, no contents thereby generated were incorporated in the thesis – neither directly nor indirectly.

[2]Deepl Pro Translator, in its version of: 30. April 2025
[3]ChatGPT in its version GPT-4o mini of: 30. April 2025

Bibliography

[ADF+25] Jens Peter Andersen, Lise Degn, Rachel Fishberg, Ebbe K. Graversen, Serge P. J. M. Horbach, Evanthia Kalpazidou Schmidt, Jesper W. Schneider, and Mads P. Sørensen. Generative Artificial Intelligence (GenAI) in the research process – A survey of researchers' practices and perceptions. *Technology in Society*, 81:102813, June 2025.

[AOAL22] Daria Alekseeva, Aleksandr Ometov, Otso Arponen, and Elena Simona Lohan. The future of computing paradigms for medical and emergency applications. *Computer Science Review*, 45:100494, 2022.

[AQIR20] Afroj Alam, Sahar Qazi, Naiyar Iqbal, and Khalid Raza. Fog, edge and pervasive computing in intelligent internet of things driven applications in healthcare: Challenges, limitations and future use. *Fog, edge, and pervasive computing in intelligent IoT driven applications*, pages 1–26, 2020.

[ASLM13] Rodrigo F Almeida, Flávio R C Sousa, Sérgio Lifschitz, and Javam C Machado. On defining metrics for elasticity of cloud databases. 2013.

[AST+10] Constantin Aliferis, Alexander Statnikov, Ioannis Tsamardinos, Subramani Mani, and Xenofon Koutsoukos. Local Causal and Markov Blanket Induction for Causal Discovery and Feature Selection for Classification Part I: Algorithms and Empirical Evaluation. *Journal of Machine Learning Research*, 11, January 2010.

[AT23] Ankur Ankan and Johannes Textor. pgmpy: A Python Toolkit for Bayesian Networks, April 2023.

[ATG+24] Negin Akbari, Adel N. Toosi, John Grundy, Hourieh Khalajzadeh, Mohammad S. Aslanpour, and Shashikant Ilager. iContinuum: An Emulation Toolkit for Intent-Based Computing Across the Edge-to-Cloud Continuum. pages 468–474. IEEE Computer Society, July 2024.

[AW15] Hassan Alrehamy and Coral Walker. Personal Data Lake With Data Gravity Pull. August 2015.

[B+20] Pete Beckman et al. Harnessing the computing continuum for programming our world. In *Fog Computing*, pages 215–230. John Wiley & Sons, Ltd, April 2020.

[Bac89] F. I. Bacchus. *Representing and reasoning with probabilistic knowledge.* Artificial Intelligence. MIT Press, 1989.

[Bat17] Jo Bates. The politics of data friction. *Journal of Documentation*, 74, August 2017.

[BBD+14] Marcello M. Bersani, Domenico Bianculli, Schahram Dustdar, Alessio Gambi, Carlo Ghezzi, and Srđan Krstić. Towards the formalization of properties of cloud-based elastic systems. In *Proceedings of the 6th International Workshop on Principles of Engineering Service-Oriented and Cloud Systems*, PESOS 2014, pages 38–47, New York, NY, USA, May 2014. Association for Computing Machinery.

[BBL+20] Pratik Baniya, Gaurav Bajaj, Jerry Lee, Ardeshir Bastani, Clifton Francis, and Mahima Agumbe Suresh. Towards Policy-aware Edge Computing Architectures. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 3464–3469, December 2020.

[BDF+20] Pete Beckman, Jack Dongarra, Nicola Ferrier, Geoffrey Fox, Terry Moore, Dan Reed, and Micah Beck. Harnessing the computing continuum for programming our world. *Fog Computing: Theory and Practice*, pages 215–230, 2020.

[Bel16] Marta Beltran. Defining an Elasticity Metric for Cloud Computing Environments. In *Proceedings of the 9th EAI International Conference on Performance Evaluation Methodologies and Tools*, VALUETOOLS'15, pages 172–179, Brussels, BEL, January 2016. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[BJ12] Anh Bui and Chi-Hyuck Jun. Learning Bayesian Network Structure Using Markov Blanket Decomposition. *Pattern Recognition Letters*, 33, December 2012.

[BMS20] Elarbi Badidi, Zineb Mahrez, and Essaid Sabir. Fog computing for smart cities' big data management and analytics: A review. *Future Internet*, 12(11):190, 2020.

[BW20] Azzedine Boukerche and Jiahao Wang. Machine learning-based traffic prediction models for intelligent transportation systems. *Computer Networks*, 181:107530, 2020.

[BXA+22] Romil Bhardwaj, Zhengxu Xia, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Nikolaos Karianakis, Kevin Hsieh, Paramvir Bahl, and Ion

Stoica. Ekya: Continuous Learning of Video Analytics Models on Edge Compute Servers. pages 119–135, 2022.

[Cam19] Mark Campbell. Smart Edge: The Effects of Shifting the Center of Data Gravity Out of the Cloud. *Computer*, 52(12):99–102, December 2019.

[Cao23] Yinan Cao. Better Orchestration for SLO-Oriented Cross-site Microservices in Multi-tenant Cloud/Edge Continuum. In *Proceedings of the 24th International Middleware Conference*, New York, USA, December 2023.

[CCL+20] Chen Chen, Lanlan Chen, Lei Liu, Shunfan He, Xiaoming Yuan, Dapeng Lan, and Zhuang Chen. Delay-optimized v2v-based computation offloading in urban vehicular edge computing and networks. *IEEE Access*, 8:18863–18873, 2020.

[CDM+25] Valeria Cardellini, Patrizio Dazzi, Gabriele Mencagli, Matteo Nardelli, and Massimo Torquati. Scalable compute continuum. *Future Generation Computer Systems*, 166:107697, May 2025.

[CGGN+18] Valeria Cardellini, Tihana Galinac Grbac, Matteo Nardelli, Nikola Tanković, and Hong-Linh Truong. QoS-Based Elasticity for Service Chains in Distributed Edge Cloud Environments. In *Autonomous Control*. 2018.

[CGK+02] Jie Cheng, Russell Greiner, Jonathan Kelly, David Bell, and Weiru Liu. Learning Bayesian networks from data: An information-theory based approach. *Artificial Intelligence*, 137(1-2):43–90, May 2002.

[CH18] Min Chen and Yixue Hao. Task Offloading for Mobile Edge Computing in Software Defined Ultra-Dense Network. *IEEE Journal on Selected Areas in Communications*, 36(3):587–597, March 2018.

[CL24] Yanfei Chen and Sanmin Liu. A novel learning method for feature evolvable streams. *Evolving Systems*, May 2024.

[CLH22] Yew Leong Cheng, Meng Hee Lim, and Kar Hoou Hui. Impact of internet of things paradigm towards energy consumption prediction: A systematic literature review. *Sustainable Cities and Society*, 78:103624, 2022.

[CLPNR22] Valeria Cardellini, Francesco Lo Presti, Matteo Nardelli, and Gabriele Russo Russo. Runtime Adaptation of Data Stream Processing Systems: The State of the Art. *ACM Comput. Surv.*, 54(11s):237:1–237:36, September 2022.

[CLW+20] Chen Chen, Bin Liu, Shaohua Wan, Peng Qiao, and Qingqi Pei. An edge traffic flow detection scheme based on deep learning in an intelligent transportation system. *IEEE Transactions on Intelligent Transportation Systems*, 22(3):1840–1852, 2020.

[CPDM+23a] Victor Casamayor Pujol, Praveen Kumar Donta, Andrea Morichetta, Ilir Murturi, and Schahram Dustdar. Edge Intelligence—Research Opportunities for Distributed Computing Continuum Systems. *IEEE Internet Computing*, 27(4):53–74, July 2023. Conference Name: IEEE Internet Computing.

[CPDM+23b] Víctor Casamayor-Pujol, Praveen Kumar Donta, Andrea Morichetta, Ilir Murturi, and Schahram Dustdar. *Distributed Computing Continuum Systems – Opportunities and Research Challenges*. March 2023.

[CPMM+23] Víctor Casamayor-Pujol, Andrea Morichetta, Ilir Murturi, Praveen Kumar Donta, and Schahram Dustdar. Fundamental Research Challenges for Distributed Computing Continuum Systems. *Information*, 14:198, March 2023.

[CPRD21] Victor Casamayor Pujol, Philipp Raith, and Schahram Dustdar. Towards a new paradigm for managing computing continuum applications. In *IEEE 3rd International Conference on Cognitive Machine Intelligence, CogMI 2021*, pages 180–188, 2021.

[CPSX+24] Victor Casamayor Pujol, Boris Sedlak, Yanwei Xu, Praveen Kumar Donta, and Schahram Dustdar. DeepSLOs for the Computing Continuum. In *Proceedings of the 2024 Workshop on Advanced Tools, Programming Languages, and PLatforms for Implementing and Evaluating algorithms for Distributed systems*, ApPLIED'24, pages 1–10, New York, NY, USA, June 2024. Association for Computing Machinery.

[CQH19] Pengfei Chen, Yong Qi, and Di Hou. CauseInfer: Automated End-to-End Performance Diagnosis with Hierarchical Causality Graph in Cloud Environment. *IEEE Transactions on Services Computing*, 2019.

[CVGN+23] Gustau Camps-Valls, Andreas Gerhardus, Urmi Ninad, Gherardo Varando, Georg Martius, Emili Balaguer-Ballester, Ricardo Vinuesa, Emiliano Diaz, Laure Zanna, and Jakob Runge. Discovering causal relations and equations from data. *Physics Reports*, 1044:1–68, October 2023.

[DCPD23] Schahram Dustdar, Victor Casamayor Pujol, and Praveen Kumar Donta. On Distributed Computing Continuum Systems. *IEEE Transactions on Knowledge and Data Engineering*, 35(4):4092–4105, April 2023.

[DFP+24] Anastasiya Danilenka, Alireza Furutanpey, Victor Casamayor Pujol, Boris Sedlak, Anna Lackinger, Maria Ganzha, Marcin Paprzycki, and Schahram Dustdar. Adaptive Active Inference Agents for Heterogeneous and Lifelong Federated Learning, October 2024.

[DGH21] Daniel Del Gaudio and Pascal Hirmer. Towards Feedback Loops in Model-Driven IoT Applications. In Johanna Barzen, editor, *Service-Oriented*

Computing, Communications in Computer and Information Science, pages 100–108, Cham, 2021. Springer International Publishing.

[DGP+23] Muhammet Deveci, Ilgin Gokasar, Dragan Pamucar, Aws Alaa Zaidan, Xin Wen, and Brij B Gupta. Evaluation of cooperative intelligent transportation system scenarios for resilience in transportation using type-2 neutrosophic fuzzy vikor. *Transportation Research Part A: Policy and Practice*, 172:103666, 2023.

[DGST11] Schahram Dustdar, Yike Guo, Benjamin Satzger, and Hong-Linh Truong. Principles of Elastic Processes. *Internet Computing, IEEE*, 15:66–71, November 2011.

[Dig22] Digital Realty. Data Gravity Index DGx. Technical Report V1.5, 2022.

[DLZZ20] Hao Du, Supeng Leng, Ke Zhang, and Longyu Zhou. Cooperative Sensing and Task Offloading for Autonomous Platoons. In *IEEE GLOBECOM 2020*, December 2020.

[DM20] Schahram Dustdar and Ilir Murturi. Towards Distributed Edge-based Systems. In *2020 IEEE Second International Conference on Cognitive Machine Intelligence (CogMI)*, pages 1–9, Atlanta, GA, USA, October 2020. IEEE.

[DMCP+23] Praveen Kumar Donta, Ilir Murturi, Victor Casamayor Pujol, Boris Sedlak, and Schahram Dustdar. Exploring the Potential of Distributed Computing Continuum Systems. *Computers*, 12(10):198, October 2023.

[DPD22] Schahram Dustdar, Victor Casamayor Pujol, and Praveen Kumar Donta. On distributed computing continuum systems. *IEEE Transactions on Knowledge and Data Engineering*, 35(4):4092–4105, 2022.

[DPD23] Schahram Dustdar, Victor Casamayor Pujol, and Praveen Kumar Donta. On Distributed Computing Continuum Systems. *IEEE Transactions on Knowledge and Data Engineering*, 35(4):4092–4105, April 2023.

[DPH+19] L Minh Dang, Md Jalil Piran, Dongil Han, Kyungbok Min, and Hyeonjoon Moon. A survey on internet of things and cloud computing for healthcare. *Electronics*, 8(7):768, 2019.

[DSCPD23a] Praveen Kumar Donta, Boris Sedlak, Victor Casamayor Pujol, and Schahram Dustdar. Governance and sustainability of distributed continuum systems: a big data approach. *Journal of Big Data*, 10(1):53, April 2023.

[DSCPD23b] Praveen Kumar Donta, Boris Sedlak, Victor Casamayor Pujol, and Schahram Dustdar. Governance and sustainability of distributed continuum systems: a big data approach. *Journal of Big Data*, 10(1):1–31, 2023.

[DTZ+22] Zhenhua Duan, Cong Tian, Nan Zhang, Mengchu Zhou, Bin Yu, Xiaobing Wang, Jiangen Guo, and Ying Wu. A novel load balancing scheme for mobile edge computing. *Journal of Systems and Software*, 186:111195, April 2022.

[DXK23] Shi Dong, Yuanjun Xia, and Joarder Kamruzzaman. Quantum Particle Swarm Optimization for Task Offloading in Mobile Edge Computing. *IEEE TII*, August 2023.

[Dzu20] Nguyen Mau Dzung. Super Fast and Accurate 3D Object Detection based on 3D LiDAR Point Clouds (SFA3D), 2020.

[Edw10] Paul N. Edwards. *A vast machine: computer models, climate data, and the politics of global warming*. MIT Press, Cambridge, Mass, 2010. OCLC: ocn430736496.

[EMB+11] Paul Edwards, Matthew Mayernik, Archer Batcheller, Geoffrey Bowker, and Christine Borgman. Science Friction: Data, Metadata, and Collaboration. *Social studies of science*, 41:667–90, October 2011.

[FCC+19] Xiayan Fan, Taiping Cui, Chunyan Cao, Qianbin Chen, and Kyung Sup Kwak. Minimum-Cost Offloading for Collaborative Task Execution of MEC-Assisted Platooning. *Sensors*, 2019.

[FD08] Shunkai Fu and Michel C. Desmarais. Fast Markov Blanket Discovery Algorithm Via Local Learning within Single Pass. In *Advances in Artificial Intelligence*, pages 96–107. Springer, Berlin, Heidelberg, 2008.

[FDCS+23] Karl Friston, Lancelot Da Costa, Noor Sajid, Conor Heins, Kai Ueltzhöffer, Grigorios A. Pavliotis, and Thomas Parr. The free energy principle made simpler but not too simple, May 2023.

[FDK09] Karl J. Friston, Jean Daunizeau, and Stefan J. Kiebel. Reinforcement Learning or Active Inference? *PLOS ONE*, 4(7):e6421, July 2009.

[FFACP18] Jonathan Fürst, Mauricio Fadel Argerich, Bin Cheng, and Apostolos Papageorgiou. Elastic Services for Edge Computing. In *2018 14th International Conference on Network and Service Management (CNSM)*, pages 358–362, November 2018.

[FKH06] Karl Friston, James Kilner, and Lee Harrison. A free energy principle for the brain. *Journal of Physiology Paris*, 100(1-3):70–87, July 2006.

[Fri13] Karl Friston. Life as we know it. *Journal of The Royal Society Interface*, 10(86):20130475, September 2013.

[FSH+21] Sheng Feng, Haiyan Shi, Longjun Huang, Shigen Shen, Shui Yu, Hua Peng, and Chengdong Wu. Unknown hostile environment-oriented autonomous wsn deployment using a mobile robot. *Journal of Network and Computer Applications*, 182:103053, 2021.

[FSL+23] Wenhao Fan, Yi Su, Jie Liu, Shenmeng Li, Wei Huang, Fan Wu, and Yuan'an Liu. Joint task offloading and resource allocation for vehicular edge computing based on v2i and v2v modes. *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[GABZ23] Davide Ghio, Antoine L. M. Aragon, Indaco Biazzo, and Lenka Zdeborová. Bayes-optimal inference for spreading processes on random networks. *Physical Review E*, 108(4):044308, October 2023.

[GB22] Shichao Guan and Azzedine Boukerche. Intelligent Edge-Based Service Provisioning Using Smart Cloudlets, Fog and Mobile Edges. *IEEE Network*, 36(2):139–145, March 2022.

[GDB22] Vidushi Goyal, Reetuparna Das, and Valeria Bertacco. Hardware-friendly User-specific Machine Learning for Edge Devices. *ACM Transactions on Embedded Computing Systems*, 21(5):62:1–62:29, October 2022.

[GFB+23] Niloy Ganguly, Dren Fazlija, Maryam Badar, Marco Fisichella, Sandipan Sikdar, Johanna Schrader, Jonas Wallat, Koustav Rudra, Manolis Koubarakis, Gourab K. Patro, Wadhah Zai El Amri, and Wolfgang Nejdl. A Review of the Role of Causality in Developing Trustworthy AI Systems, February 2023.

[GH18] Mila Gascó-Hernandez. Building a smart city: lessons from Barcelona. *Communications of the ACM*, 61(4):50–57, March 2018.

[GLL20] Hongzhi Guo, Jiajia Liu, and Jianfeng Lv. Toward Intelligent Task Offloading at the Edge. *IEEE Network*, 34(2):128–134, March 2020.

[GLZ+21] Yujia Gao, Liang Liu, Xiaolong Zheng, Chi Zhang, and Huadong Ma. Federated sensing: Edge-cloud elastic collaborative learning for intelligent sensing. *IEEE Internet of Things*, 2021.

[GMP+21] Yisel Garí, David A. Monge, Elina Pacini, Cristian Mateos, and Carlos García Garino. Reinforcement learning-based application Autoscaling in the Cloud: A survey. *Engineering Applications of Artificial Intelligence*, 102:104288, June 2021.

[GRG22] Hui Guo, Lan-lan Rui, and Zhi-peng Gao. V2v task offloading algorithm with lstm-based spatiotemporal trajectory prediction model in svcns. *IEEE TVT*, 2022.

[GWK+19] Stefanos Gritzalis, Edgar R. Weippl, Sokratis K. Katsikas, Gabriele Anderst-Kotsis, A Min Tjoa, and Ismail Khalil, editors. *Trust, Privacy and Security in Digital Business: 16th International Conference, TrustBus 2019, Linz, Austria, August 26–29, 2019, Proceedings*, volume 11711 of *Lecture Notes in Computer Science*. Springer International Publishing, Cham, 2019.

[HCHC19] Yuyu Hu, Taiping Cui, Xiaoge Huang, and Qianbin Chen. Task Offloading Based on Lyapunov Optimization for MEC-assisted Platooning. In *2019 11th International Conference on Wireless Communications and Signal Processing (WCSP)*, pages 1–5, October 2019.

[HDAD23a] Abhishek Hazra, Praveen Kumar Donta, Tarachand Amgoth, and Schahram Dustdar. Cooperative Transmission Scheduling and Computation Offloading With Collaboration of Fog and Cloud for Industrial IoT Applications. *IEEE Internet of Things Journal*, March 2023.

[HDAD23b] Abhishek Hazra, Praveen Kumar Donta, Tarachand Amgoth, and Schahram Dustdar. Cooperative transmission scheduling and computation offloading with collaboration of fog and cloud for industrial iot applications. *IEEE Internet of Things Journal*, 10(5):3944–3953, 2023.

[HKR13] Nikolas Herbst, Samuel Kounev, and Ralf Reussner. Elasticity in cloud computing: What it is, and what it is not. *International Conference on Autonomic Computing*, pages 23–27, January 2013.

[HLK18] Yen-Chang Hsu, Zhaoyang Lv, and Zsolt Kira. Learning to cluster in order to transfer across domains and tasks. In *Sixth International Conference on Learning Representations (ICLR 2018)*, March 2018.

[HMD+22] Conor Heins, Beren Millidge, Daphne Demekas, Brennan Klein, Karl Friston, Iain Couzin, and Alexander Tschantz. pymdp: A Python library for active inference in discrete state spaces. *Journal of Open Source Software*, May 2022.

[HMDC19] Carol Habib, Abdallah Makhoul, Rony Darazi, and Raphaël Couturier. Health risk assessment and decision-making for patient monitoring and decision-support using wireless body sensor networks. *Information fusion*, 47:10–22, 2019.

[HMSS+22] Baydaa Hashim Mohammed, Hasimi Sallehuddin, Nurhizam Safie, Afifuddin Husairi, Nur Azaliah Abu Bakar, Farashazillah Yahya, Ihsan Ali, and Shaymaa AbdelGhany Mohamed. Building information modeling and

internet of things integration in the construction industry: A scoping study. *Advances in Civil Engineering*, 2022, 2022.

[HS05] M.N. Huhns and M.P. Singh. Service-oriented computing: key concepts and principles. *IEEE Internet Computing*, 9(1):75–81, January 2005. Conference Name: IEEE Internet Computing.

[HWH+23] Wei Hao, Zixi Wang, Lauren Hong, Lingxiao Li, Nader Karayanni, Chengzhi Mao, Junfeng Yang, and Asaf Cidon. Monitoring and Adapting ML Models on Mobile Devices, May 2023.

[JBP+23] Byeonghui Jeong, Seungyeon Baek, Sihyun Park, Jueun Jeon, and Young-Sik Jeong. Stable and efficient resource management using deep neural network on cloud computing. *Neurocomputing*, 521:99–112, February 2023.

[JWTI23] Matthijs Jansen, Linus Wagner, Animesh Trivedi, and Alexandru Iosup. Continuum: Automate Infrastructure Deployment and Benchmarking in the Compute Continuum. In *ACM SPEC 2023*, New York, NY, USA, April 2023.

[KBHH23] Tae Ho Kim, Sang Ho Bae, Chang Hun Han, and Bongsu Hahn. The design of a low-cost sensing and control architecture for a search and rescue assistant robot. *Machines*, 11(3):329, 2023.

[KBJ+20] Sampo Kuutti, Richard Bowden, Yaochu Jin, Phil Barber, and Saber Fallah. A survey of deep learning applications to autonomous vehicle control. *IEEE TITS*, 2020.

[KC03] J.O. Kephart and D.M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, January 2003.

[KCG+23] Neville Kenneth Kitson, Anthony C. Constantinou, Zhigao Guo, Yang Liu, and Kiattikun Chobtham. A survey of Bayesian Network structure learning. *Artificial Intelligence Review*, 56(8):8721–8814, August 2023.

[KDKA23] Mudassar Ali Khan, Ikram Ud Din, Byung-Seo Kim, and Ahmad Almogren. Visualization of remote patient monitoring system based on internet of medical things. *Sustainability*, 15(10):8120, 2023.

[KL03] Alexander Keller and Heiko Ludwig. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *Journal of Network and Systems Management*, 11(1):57–81, March 2003.

[KLM+23] Henna Kokkonen, Lauri Lovén, Naser Hossein Motlagh, Abhishek Kumar, Juha Partala, Tri Nguyen, Víctor Casamayor Pujol, Panos Kostakos, Teemu Leppänen, Alfonso González-Gil, Ester Sola, Iñigo Angulo, Madhusanka Liyanage, Mehdi Bennis, Sasu Tarkoma, Schahram Dustdar,

Susanna Pirttikangas, and Jukka Riekki. Autonomy and Intelligence in the Computing Continuum: Challenges, Enablers, and Future Directions for Orchestration, February 2023.

[KM22] Mohammad Ali Khoshkholghi and Toktam Mahmoodi. Edge intelligence for service function chain deployment in NFV-enabled networks. *Computer Networks*, 219:109451, December 2022.

[KMH+21] Dragi Kimovski, Roland Mathá, Josef Hammer, Narges Mehran, Hermann Hellwagner, and Radu Prodan. Cloud, Fog or Edge: Where to Compute? *IEEE Internet Computing*, 25(4):30–36, July 2021. arXiv:2101.10417 [cs].

[KPNS21] Jyotirmoy Karjee, S Praveen Naik, and N Srinidhi. Energy Profiling based Load-Balancing Approach in IoT-Edge for Split Computing. *2021 IEEE 18th India Council International Conference (INDICON)*, pages 1–6, December 2021.

[KPP+18] Michael Kirchhoff, Thomas Parr, Ensor Palacios, Karl Friston, and Julian Kiverstein. The Markov blankets of life: autonomy, active inference and the free energy principle. *Journal of The Royal Society Interface*, 2018.

[KPS+20] Roman Kolcun, Diana Andreea Popescu, Vadim Safronov, Poonam Yadav, Anna Maria Mandalari, Yiming Xie, Richard Mortier, and Hamed Haddadi. The Case for Retraining of ML Models for IoT Device Identification at the Edge, November 2020.

[KVT21] Mohamad Kashef, Anna Visvizi, and Orlando Troisi. Smart city as a smart service system: Human-computer interaction and smart city surveillance systems. *Computers in Human Behavior*, 124:106923, 2021.

[LCLW21] Zhihan Lv, Dongliang Chen, Ranran Lou, and Qingjun Wang. Intelligent edge computing based on machine learning for smart city. *Future Generation Computer Systems*, 115:90–99, 2021.

[LCZ18] Jinjin Lin, Pengfei Chen, and Zibin Zheng. Microscope: Pinpoint Performance Issues with Causal Graphs in Micro-service Environments. In Claus Pahl, Maja Vukovic, Jianwei Yin, and Qi Yu, editors, *Service-Oriented Computing*, Lecture Notes in Computer Science, pages 3–20, Cham, 2018. Springer International Publishing.

[LEB15] Sebastian Lehrig, Hendrik Eikerling, and Steffen Becker. Scalability, Elasticity, and Efficiency in Cloud Computing: a Systematic Literature Review of Definitions and Metrics. In *Proceedings of the 11th International ACM SIGSOFT Conference*, pages 83–92, New York, USA, May 2015.

[LHAES23] Qianlin Liang, Walid A. Hanafy, Ahmed Ali-Eldin, and Prashant Shenoy. Model-driven Cluster Resource Management for AI Workloads in Edge

Clouds. *ACM Transactions on Autonomous and Adaptive Systems*, 18(1):2:1–2:26, March 2023.

[Lin22] Linzaer. Ultra Fast Face-Detector, February 2022. https://github.com/Linzaer/Ultra-Light-Fast-Generic-Face-Detector-1MB.

[LL20] JongBeom Lim and DaeWon Lee. A Load Balancing Algorithm for Mobile Devices in Edge Cloud Computing Environments. *Electronics*, 9(4):686, April 2020.

[LLSY22] Lingyun Lu, Xiang Li, Jingxin Sun, and Zhihe Yang. Cooperative Computation Offloading and Resource Management for Vehicle Platoon: A Deep Reinforcement Learning Approach. In *IEEE Int Conf on High Performance Computing & Communications*, 2022.

[LMF+25] Sergio Laso, Ilir Murturi, Pantelis Frangoudis, Juan Luis Herrera, Juan M. Murillo, and Schahram Dustdar. A Multidimensional Elasticity Framework for Adaptive Data Analytics Management in the Computing Continuum, January 2025.

[LMYDM22] Luc Le Mero, Dewei Yi, Mehrdad Dianati, and Alexandros Mouzakitis. A survey on imitation learning techniques for end-to-end autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 23(9):14128–14147, 2022.

[LPS+19] Georgiy Levchuk, Krishna Pattipati, Daniel Serfaty, Adam Fouse, and Robert McCormack. Active Inference in Multiagent Systems: Context-Driven Collaboration and Decentralized Purpose-Driven Team Adaptation. In *Artificial Intelligence for the Internet of Everything*. Academic Press, 2019.

[LWL+23] Shuaibing Lu, Jie Wu, Pengfan Lu, Ning Wang, Haiming Liu, and Juan Fang. QoS-Aware Online Service Provisioning and Updating in Cost-Efficient Multi-Tenant Mobile Edge Computing. *IEEE Services Computing*, 2023.

[LZHH23] Mingyang Lyu, Yibo Zhao, Chao Huang, and Hailong Huang. Unmanned aerial vehicles for search and rescue: A survey. *Remote Sensing*, 15(13):3266, 2023.

[Mac10] Dave MacCrory. Data Gravity – in the Clouds – Data Gravitas, December 2010.

[MAGA+19] Ammar Awad Mutlag, Mohd Khanapi Abd Ghani, Net al Arunkumar, Mazin Abed Mohammed, and Othman Mohd. Enabling technologies for fog computing in healthcare iot systems. *Future generation computer systems*, 90:62–78, 2019.

[MAM+24] Ryan May, Sean Arms, Patrick Marsh, Eric Bruning, John Leeman, Kevin Goebbert, Jonathan Thielen, Zachary Bruick, and M. Drew Camron. MetPy: A Python Package for Meteorological Data, April 2024.

[MATM23] Leonardo Militano, Adriana Arteaga, Giovanni Toffetti, and Nathalie Mitton. The cloud-to-edge-to-iot continuum as an enabler for search and rescue operations. *Future Internet*, 15(2):55, 2023.

[MBB13] Mohamad Mehdi, Nizar Bouguila, and Jamal Bentahar. A QoS-Based Trust Approach for Service Selection and Composition via Bayesian Networks. In *2013 IEEE 20th International Conference on Web Services*, June 2013.

[MCM19] Andrea Morichetta, Pedro Casas, and Marco Mellia. EXPLAIN-IT: Towards Explainable AI for Unsupervised Network Traffic Analysis. In *Proceedings of the 3rd ACM CoNEXT Workshop on Big DAta, Machine Learning and Artificial Intelligence for Data Communication Networks*, Big-DAMA '19, pages 22–28, New York, NY, USA, December 2019. Association for Computing Machinery.

[MD22] Ilir Murturi and Schahram Dustdar. A Decentralized Approach for Resource Discovery using Metadata Replication in Edge Networks. *IEEE Transactions on Services Computing*, 15(5):2526–2537, September 2022.

[Men21] Vítor Hugo Menino. A Novel Approach to Load Balancing in P2P Overlay Networks for Edge Systems. 2021.

[MH22] Mohamed H. Mousa and Mohamed K. Hussein. Efficient UAV-based mobile edge computing using differential evolution and ant colony optimization. *PeerJ Computer Science*, 2022.

[MKBC21] Ernesto C. Martínez, Jong Woo Kim, Tilman Barz, and M. Cruz. Probabilistic Modeling for Optimization of Bioreactors using Reinforcement Learning with Active Inference. *Computer Aided Chemical Engineering*, 2021.

[MMD+19] Patric Marques, Diogo Manfroi, Eduardo Deitos, Jonatan Cegoni, Rodrigo Castilhos, Juergen Rochol, Edison Pignaton, and Rafael Kunst. An iot-based smart cities infrastructure architecture applied to a waste management scenario. *Ad Hoc Networks*, 87:200–208, 2019.

[MPK09] Ole Mengshoel, Scott Poll, and Tolga Kurtoglu. Developing Large-Scale Bayesian Networks by Composition: Fault Diagnosis of Electrical Power Systems in Aircraft and Spacecraft. January 2009.

[MPN+23a] Andrea Morichetta, V. Casamayor Pujol, Stefan Nastic, Schahram Dustdar, Deepak Vij, Ying Xiong, and Zhaobo Zhang. PolarisProfiler: A novel metadata-based profiling approach for optimizing resource management in

the edge-cloud continnum. In *18th Annual System of Systems Engineering Conference (SOSE)*, 2023.

[MPN+23b] Andrea Morichetta, Vıctor Casamayor Pujol, Stefan Nastic, Thomas Pusztai, Philipp Raith, Schahram Dustdar, Deepak Vij, Ying Xiong, and Zhaobo Zhang. Demystifying deep learning in predictive monitoring for cloud-native SLOs. 2023.

[MSRD23] Andrea Morichetta, Nikolaus Spring, Philipp Raith, and Schahram Dustdar. Intent-based management for the distributed computing continuum. In *2023 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pages 239–249. IEEE, 2023.

[MSXY22] Xiandong Ma, Zhou Su, Qichao Xu, and Bincheng Ying. Edge Computing and UAV Swarm Cooperative Task Offloading in Vehicular Networks. In *2022 International Wireless Communications and Mobile Computing (IWCMC)*, pages 955–960, May 2022.

[Mur22] Ilir Murturi. *Resource Management and Elasticity Control in Edge Networks*. PhD thesis, 2022.

[MWYY20] Yifang Ma, Zhenyu Wang, Hong Yang, and Lin Yang. Artificial intelligence applications in the development of autonomous vehicles. *IEEE Journal of Automatica Sinica*, 2020.

[MYG18] Carla Mouradian, Sami Yangui, and Roch H Glitho. Robots as-a-service in cloud computing: Search and rescue in large-scale disasters case study. In *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pages 1–7. IEEE, 2018.

[NGS+20] B Naveen Naik, Rekha Gupta, Ajay Singh, Shiv Lal Soni, and GD Puri. Real-time smart patient monitoring and assessment amid covid-19 pandemic–an alternative approach to remote monitoring. *Journal of Medical Systems*, 44:1–2, 2020.

[NKFW19] Sina Niedermaier, Falko Koetter, Andreas Freymann, and Stefan Wagner. On Observability and Monitoring of Distributed Systems – An Industry Interview Study. In Sami Yangui, Ismael Bouassida Rodriguez, Khalil Drira, and Zahir Tari, editors, *Service-Oriented Computing*, 2019.

[NMC07] Alexandru Niculescu-Mizil and Rich Caruana. Inductive Transfer for Bayesian Network Structure Learning. In *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, pages 339–346. PMLR, March 2007.

[NMP+20] Stefan Nastic, Andrea Morichetta, Thomas Pusztai, Schahram Dustdar, Xiaoning Ding, Deepak Vij, and Ying Xiong. SLOC: Service Level Objectives for Next Generation Cloud Computing. *IEEE Internet Computing*, 24(3), May 2020.

[NPM+21] Stefan Nastic, Thomas Pusztai, Andrea Morichetta, Victor Casamayor Pujol, Schahram Dustdar, Deepak Vii, and Ying Xiong. Polaris Scheduler: Edge Sensitive and SLO Aware Workload Scheduling in Cloud-Edge-IoT Clusters. In *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, pages 206–216, Chicago, IL, USA, September 2021. IEEE.

[NRF+22] Stefan Nastic, Philipp Raith, Alireza Furutanpey, Thomas Pusztai, and Schahram Dustdar. A Serverless Computing Fabric for Edge and Cloud. In *2022 IEEE 4th International Conference on Cognitive Machine Intelligence (CogMI)*, pages 1–12, Atlanta, GA, USA, December 2022. IEEE.

[NRRC24] Matteo Nardelli, Gabriele Russo Russo, and Valeria Cardellini. Compute Continuum: What Lies Ahead? In *Euro-Par 2023: Parallel Processing Workshops*, 2024.

[ope24] opencv. opencv at 4.9.0, 2024.

[OSF22] Murugaraj Odiathevar, Winston K.G. Seah, and Marcus Frean. A Bayesian Approach To Distributed Anomaly Detection In Edge AI Networks. *IEEE Transactions on Parallel and Distributed Systems*, December 2022.

[PD21] Victor Casamayor Pujol and Schahram Dustdar. Fog robotics–understanding the research challenges. *IEEE Internet Computing*, 25(5):10–17, 2021.

[PD23] Victor Casamayor Pujol and Schahram Dustdar. Towards a Prime Directive of SLOs. In *2023 IEEE International Conference on Software Services Engineering (SSE)*, pages 61–70, July 2023.

[Pea88a] Judea Pearl. *Probabilistic reasoning in intelligent systems : networks of plausible inference*. San Mateo, Calif. : Morgan Kaufmann, 1988.

[Pea88b] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[Pea09] Judea Pearl. Causal inference in statistics: An overview. *Statistics Surveys*, 3(none):96–146, January 2009.

[Pet21] Dana Petcu. Service Deployment Challenges in Cloud-to-Edge Continuum. *Scalable Computing: Practice and Experience*, November 2021.

[PGPA+18] Pierluigi Plebani, David Garcia-Perez, Maya Anderson, David Bermbach, Cinzia Cappiello, Ronen I. Kat, Achilleas Marinakis, Vrettos Moulos, Frank Pallas, Stefan Tai, and Monica Vitali. Data and Computation Movement in Fog Environments: The DITAS Approach. In Zaigham Mahmood, editor, *Fog Computing: Concepts, Frameworks and Technologies*, pages 249–266. Springer International Publishing, Cham, 2018.

[PK18] Christopher T.J. Prentice and Georgios Karakonstantis. Smart Office System with Face Detection at the Edge. In *2018 IEEE SmartWorld*, pages 88–93, October 2018.

[PM18] Judea Pearl and Dana Mackenzie. *The Book of Why: The New Science of Cause and Effect*. Basic Books, Inc., USA, 2018.

[PMN23] Victor Casamayor Pujol, Andrea Morichetta, and Stefan Nastic. Intelligent Sampling: A Novel Approach to Optimize Workload Scheduling in Large-Scale Heterogeneous Computing Continuum. In *2023 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, July 2023.

[PMP+21a] Thomas Pusztai, Andrea Morichetta, Victor Casamayor Pujol, Schahram Dustdar, Stefan Nastic, Xiaoning Ding, Deepak Vij, and Ying Xiong. SLO Script: A Novel Language for Implementing Complex Cloud-Native Elasticity-Driven SLOs. In *2021 IEEE ICWS*, pages 21–31, Chicago, IL, USA, September 2021. IEEE.

[PMP+21b] Thomas Pusztai, Andrea Morichetta, Víctor Casamayor Pujol, Schahram Dustdar, Stefan Nastic, Xiaoning Ding, Deepak Vij, and Ying Xiong. A Novel Middleware for Efficiently Implementing Complex Cloud-Native SLOs. In *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, September 2021.

[PMS+20] Akira-Sebastian Poncette, Lina Mosch, Claudia Spies, Malte Schmieding, Fridtjof Schiefenhövel, Henning Krampe, and Felix Balzer. Improvements in patient monitoring in the intensive care unit: survey study. *Journal of medical Internet research*, 22(6):e19091, 2020.

[PNM+22] Thomas Pusztai, Stefan Nastic, Andrea Morichetta, Víctor Casamayor Pujol, Philipp Raith, Schahram Dustdar, Deepak Vij, Ying Xiong, and Zhaobo Zhang. Polaris Scheduler: SLO- and Topology-aware Microservices Scheduling at the Edge. In *15th International Conference on Utility and Cloud Computing*, pages 61–70, December 2022.

[PPF22] Thomas Parr, Giovanni Pezzulo, and Karl J. Friston. *Active Inference: The Free Energy Principle in Mind, Brain, and Behavior*. The MIT Press, March 2022.

[PRD+20] Kellow Pardini, Joel JPC Rodrigues, Ousmane Diallo, Ashok Kumar Das, Victor Hugo C de Albuquerque, and Sergei A Kozlov. A smart waste management solution geared towards citizens. *Sensors*, 20(8):2380, 2020.

[PRD21] Víctor Casamayor Pujol, Philipp Raith, and Schahram Dustdar. Towards a new paradigm for managing computing continuum applications. In *2021 IEEE Third International Conference on Cognitive Machine Intelligence (CogMI)*, December 2021.

[PRK+19] Kellow Pardini, Joel JPC Rodrigues, Sergei A Kozlov, Neeraj Kumar, and Vasco Furtado. Iot-based solid waste management solutions: a survey. *Journal of Sensor and Actuator Networks*, 8(1):5, 2019.

[PRP+20] Ensor Rafael Palacios, Adeel Razi, Thomas Parr, Michael Kirchhoff, and Karl Friston. On Markov blankets and hierarchical self-organisation. *Journal of Theoretical Biology*, 486, February 2020.

[PSDD24] Víctor Casamayor Pujol, Boris Sedlak, Praveen Kumar Donta, and Schahram Dustdar. On Causality in Distributed Continuum Systems. *IEEE Internet Computing*, 28(2):57–64, March 2024.

[QBJ+20] Haoran Qiu, Subho S. Banerjee, Saurabh Jha, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer. FIRM: An Intelligent Fine-grained Resource Management Framework for SLO-Oriented Microservices. pages 805–825, 2020.

[QLZH18] Guanhua Qiao, Supeng Leng, Ke Zhang, and Yejun He. Collaborative task offloading in vehicular edge multi-access networks. *IEEE Communications Magazine*, 56(8):48–54, 2018.

[RCVA22] Daniel Rosendo, Alexandru Costan, Patrick Valduriez, and Gabriel Antoniu. Distributed intelligence on the edge-to-cloud continuum: A systematic literature review. *Journal of Parallel and Distributed Computing*, 166:71–94, 2022.

[RGC15] Aaditya Ramdas, Nicolas Garcia, and Marco Cuturi. On Wasserstein Two Sample Testing and Related Families of Nonparametric Tests, October 2015.

[RMBG21] Sanaz Rabinia, Haydar Mehryar, Marco Brocanelli, and Daniel Grosu. Data Sharing-Aware Task Allocation in Edge Computing Systems. In *2021 IEEE International Conference on Edge Computing (EDGE)*, pages 60–67, September 2021. ISSN: 2767-9918.

[RPN+22] Dumitru Roman, Radu Prodan, Nikolay Nikolov, Ahmet Soylu, Mihhail Matskin, Andrea Marrella, Dragi Kimovski, Brian Elvesæter, Anthony Simonet-Boulogne, Giannis Ledakis, Hui Song, Francesco Leotta, and

Evgeny Kharlamov. Big Data Pipelines on the Computing Continuum: Tapping the Dark Data. *Computer*, 55(11):74–84, November 2022.

[RRS+22] Abderahman Rejeb, Karim Rejeb, Steve Simske, Horst Treiblmaier, and Suhaiza Zailani. The big picture on the internet of things and the smart city: a review of what we know and what we need to know. *Internet of Things*, August 2022.

[RTG15] Rasmus Rothe, Radu Timofte, and Luc Van Gool. DEX: Deep EXpectation of Apparent Age from a Single Image. In *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*, pages 252–257, Santiago, Chile, December 2015. IEEE.

[RVM+23] Banoth Ravi, Blesson Varghese, Ilir Murturi, Praveen Kumar Donta, Schahram Dustdar, Chinmaya Kumar Dehury, and Satish Narayana Srirama. Stochastic modeling for intelligent software-defined vehicular networks: A survey. *Computers*, 12(8), 2023.

[RWC+20] Gregory B Rehm, Sang Hoon Woo, Xin Luigi Chen, Brooks T Kuhn, Irene Cortes-Puch, Nicholas R Anderson, Jason Y Adams, and Chen-Nee Chuah. Leveraging iots and machine learning for patient diagnosis and ventilation management in the intensive care unit. *IEEE Pervasive Computing*, 19(3):68–78, 2020.

[SAR+21] Kumar A. Shukla, Shahanawaj Ahamad, G.Nageswara Rao, Avein Jabar Al-Asadi, Ankur Gupta, and Makhan Kumbhkar. Artificial Intelligence Assisted IoT Data Intrusion Detection. In *ICCCT 2021*, December 2021.

[SBA20] Bharath Sudharsan, John G. Breslin, and Muhammad Intizar Ali. Edge2Train: a framework to train machine learning models (SVMs) on resource-constrained IoT edge devices. In *Proceedings of the 10th International Conference on the Internet of Things*, IoT '20, pages 1–8, New York, NY, USA, October 2020. Association for Computing Machinery.

[SBIB+24] Andrea Soltoggio, Eseoghene Ben-Iwhiwhu, Vladimir Braverman, Eric Eaton, Benjamin Epstein, Yunhao Ge, Lucy Halperin, Jonathan How, Laurent Itti, Michael A. Jacobs, Pavan Kantharaju, Long Le, Steven Lee, Xinran Liu, Sildomar T. Monteiro, David Musliner, Saptarshi Nath, Priyadarshini Panda, Christos Peridis, Hamed Pirsiavash, Vishwa Parekh, Kaushik Roy, Shahaf Shperberg, Hava T. Siegelmann, Peter Stone, Kyle Vedder, Jingfeng Wu, Lin Yang, Guangyao Zheng, and Soheil Kolouri. A collective AI via lifelong learning and sharing at the edge. *Nature Machine Intelligence*, 6(3):251–264, March 2024.

[SBPF21] Noor Sajid, Philip J. Ball, Thomas Parr, and Karl J. Friston. Active inference: demystified and compared. *Neural Computation*, 2021.

[SBZ18] Ragini Sharma, Saman Biookaghazadeh, and Ming Zhao. Are Existing Knowledge Transfer Techniques Effective For Deep Learning on Edge Devices? In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '18, pages 15–16, New York, NY, USA, June 2018. Association for Computing Machinery.

[SCK+21] Wonik Seo, Sanghoon Cha, Yeonjae Kim, Jaehyuk Huh, and Jongse Park. SLO-Aware Inference Scheduler for Heterogeneous Processors in Edge Platforms. *ACM Transactions on Architecture and Code Optimization*, 18(4):1–26, December 2021.

[SCPDD23] Boris Sedlak, Victor Casamayor Pujol, Praveen Kumar Donta, and Schahram Dustdar. Controlling Data Gravity and Data Friction: From Metrics to Multidimensional Elasticity Strategies. In *2023 IEEE International Conference on Software Services Engineering (SSE)*, pages 43–49, Chicago, IL, USA, July 2023.

[SCQC23] Andrés L. Suárez-Cetrulo, David Quintana, and Alejandro Cervantes. A survey on machine learning for recurring concept drifting data streams. *Expert Systems with Applications*, 213:118934, March 2023.

[Scu10] Marco Scutari. Learning Bayesian Networks with the bnlearn R Package. *Journal of Statistical Software*, 35:1–22, July 2010.

[SCZ+16] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*, 3(5):637–646, October 2016.

[SD16] Weisong Shi and Schahram Dustdar. The Promise of Edge Computing. *Computer*, 49(5):78–81, May 2016.

[SFBS20] Mohammad Shahverdy, Mahmood Fathy, Reza Berangi, and Mohammad Sabokrou. Driver behavior detection and classification using deep convolutional neural networks. *Expert Systems with Applications*, 2020.

[SFW22] Ryan Smith, Karl J. Friston, and Christopher J. Whyte. A step-by-step tutorial on active inference and its application to empirical data. *Journal of Mathematical Psychology*, 107:102632, April 2022.

[SIIA21] Nicholas Chieng Anak Sallang, Mohammad Tariqul Islam, Mohammad Shahidul Islam, and Haslina Arshad. A CNN-based smart waste management system using tensorflow lite and LoRa-GPS shield in internet of things environment. *IEEE Access*, 9:153560–153574, 2021.

[SJK+20] Manu Sharma, Sudhanshu Joshi, Devika Kannan, Kannan Govindan, Rohit Singh, and HC Purohit. Internet of things (iot) adoption barriers of smart cities' waste management: An indian context. *Journal of Cleaner Production*, 270:122047, 2020.

[SMB21] Jacopo Soldani, Giuseppe Montesano, and Antonio Brogi. What Went Wrong? Explaining Cascading Failures in Microservice-Based Applications. In Johanna Barzen, editor, *Service-Oriented Computing*, pages 133–153, Cham, 2021. Springer International Publishing.

[SMD22] Boris Sedlak, Ilir Murturi, and Schahram Dustdar. Specification and Operation of Privacy Models for Data Streams on the Edge. In *2022 IEEE 6th International Conference on Fog and Edge Computing (ICFEC)*, pages 78–82, Messina, Italy, May 2022. IEEE.

[SMDD23] Boris Sedlak, Ilir Murturi, Praveen Kumar Donta, and Schahram Dustdar. A Privacy Enforcing Framework for Transforming Data Streams on the Edge. *IEEE Transactions on Emerging Topics in Computing*, 2023.

[SMR+25] Boris Sedlak, Andrea Morichetta, Philipp Raith, Victor Casamayor Pujol, and Schahram Dustdar. Towards Multi-dimensional Elasticity for Pervasive Stream Processing Services. In *2025 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, 2025.

[SMW+24] Boris Sedlak, Andrea Morichetta, Yuhao Wang, Yang Fei, Liang Wang, Schahram Dustdar, and Xiaobo Qu. SLO-Aware Task Offloading Within Collaborative Vehicle Platoons. In Walid Gaaloul, Michael Sheng, Qi Yu, and Sami Yangui, editors, *Service-Oriented Computing*, pages 72–86, Singapore, December 2024. Springer Nature.

[SPDD23] Boris Sedlak, Victor Casamayor Pujol, Praveen Kumar Donta, and Schahram Dustdar. Designing Reconfigurable Intelligent Systems with Markov Blankets. In *Service-Oriented Computing*, pages 42–50. Springer Nature Switzerland, 2023.

[SPDD24a] Boris Sedlak, Victor Casamayor Pujol, Praveen Kumar Donta, and Schahram Dustdar. Active Inference on the Edge: A Design Study. In *2024 IEEE PerCom Workshops*, pages 550–555, March 2024.

[SPDD24b] Boris Sedlak, Victor Casamayor Pujol, Praveen Kumar Donta, and Schahram Dustdar. Equilibrium in the Computing Continuum through Active Inference. *Future Generation Computer System*, 160:92–108, 2024.

[SPDD24c] Boris Sedlak, Victor Casamayor Pujol, Praveen Kumar Donta, and Schahram Dustdar. Markov Blanket Composition of SLOs. In *2024 IEEE International Conference on Edge Computing and Communications (EDGE)*, pages 128–138, Shenzhen, China, 2024.

[SPDD24d] Boris Sedlak, Víctor Casamayor Pujol, Praveen Kumar Donta, and Schahram Dustdar. Diffusing High-level SLO in Microservice Pipelines.

In *2024 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pages 11–19, Shanghai, China, 2024.

[SPJC18] Jang-Ping Sheu, Yi-Cian Pu, R.B. Jagadeesha, and Yeh-Cheng Chang. An efficient module deployment algorithm in edge computing. In *IEEE Wireless Communications and Networking Workshops (WCNCW)*, April 2018.

[SPM+24] Boris Sedlak, Victor Casamayor Pujol, Andrea Morichetta, Praveen Kumar Donta, and Schahram Dustdar. Adaptive Stream Processing on Edge Devices through Active Inference, September 2024.

[SSS19] Mauro Scanagatta, Antonio Salmerón, and Fabio Stella. A survey on bayesian network structure learning from data. *Progress in Artificial Intelligence*, 8:425–439, 2019.

[T+22] William Tärneberg et al. The 6G Computing Continuum (6GCC): Meeting the 6G computing challenges. In *International Conference on 6G Networking*, July 2022.

[TAS03] Ioannis Tsamardinos, Constantin F. Aliferis, and Alexander Statnikov. Time and sample efficient discovery of Markov blankets and direct causal relations. New York, USA, August 2003. Association for Computing Machinery.

[TDFS21] László Toka, Gergely Dobreff, Balázs Fodor, and Balázs Sonkoly. Machine Learning-Based Scaling Management for Kubernetes Edge Clusters. *IEEE Transactions on Network and Service Management*, 18(1):958–972, March 2021.

[THB12] Vangelis Tasoulas, H. Haugerud, and Kyrre M. Begnum. Bayllocator: A Proactive System to Predict Server Utilization and Dynamically Allocate Memory Resources Using Bayesian Networks and Ballooning. December 2012.

[TK24] Minh-Ngoc Tran and YoungHan Kim. Optimized resource usage with hybrid auto-scaling system for knative serverless edge computing. *Future Generation Computer Systems*, 152:304–316, 2024.

[TMSB20] Alexander Tschantz, Beren Millidge, Anil K. Seth, and Christopher L. Buckley. Reinforcement Learning through Active Inference, February 2020.

[Tog22] Mesut Togacar. Detecting attacks on IoT devices with probabilistic Bayesian neural networks and hunger games search optimization approaches. *Transactions on Telecommunications Technologies*, 2022.

[TW22] Ming Tang and Vincent W.S. Wong. Deep Reinforcement Learning for Task Offloading in Mobile Edge Computing Systems. *IEEE Transactions on Mobile Computing*, 21(6):1985–1997, June 2022.

[TZV+08] Mukarram Tariq, Amgad Zeitoun, Vytautas Valancius, Nick Feamster, and Mostafa Ammar. Answering what-if deployment and configuration questions with wise. *ACM SIGCOMM Computer Communication Review*, 2008.

[VCB21] Matthew J. Vowels, Necati Cihan Camgoz, and Richard Bowden. D'ya like DAGs? A Survey on Structure Learning and Causal Discovery, March 2021.

[VF23] Victor Velepucha and Pamela Flores. A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges. *IEEE Access*, 2023.

[VLS23] Miroslav Vanis, Zdenek Lokaj, and Martin Srotyr. A Novel Algorithm for Merging Bayesian Networks. *Symmetry*, 15(7):1461, July 2023.

[VM24] Rejin Varghese and Sambath M. YOLOv8: A Novel Object Detection Algorithm with Enhanced Performance and Robustness. In *ADICS*, 2024.

[VRP22] Matteo Vagnoli and Rasa Remenyte-Prescott. Updating conditional probabilities of Bayesian belief networks by merging expert knowledge and system monitoring data. *Automation in Construction*, August 2022.

[Wam23] Innocent Gicheru Wambui. Improving Traffic Flow Using LSTM Networks in Python: A Step-by-Step Guide, August 2023.

[WCB+23] Yuxin Wu, Changjun Cai, Xuanming Bi, Junjuan Xia, Chongzhi Gao, Yajuan Tang, and Shiwei Lai. Intelligent resource allocation scheme for cloud-edge-end framework aided multi-source data stream. *EURASIP Journal on Advances in Signal Processing*, 2023, May 2023.

[WLYW20] Hao Wang, Zhaolong Ling, Kui Yu, and Xindong Wu. Towards efficient and effective discovery of Markov blankets for feature selection. *Information Sciences*, 509, January 2020.

[WOK17] Jürgen Walter, Dušan Okanović, and Samuel Kounev. Mapping of Service Level Objectives to Performance Queries. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*, ICPE '17 Companion, pages 197–202, New York, NY, USA, April 2017. Association for Computing Machinery.

[WQQ+21] Cong Wang, Jiongming Qin, Cheng Qu, Xu Ran, Chuanjun Liu, and Bin Chen. A smart municipal waste management system based on deep-learning and internet of things. *Waste Management*, 135:20–29, 2021.

[WSM+22] Abdul Waheed, Munam Ali Shah, Syed Muhammad Mohsin, Abid Khan, Carsten Maple, Sheraz Aslam, and Shahab Shamshirband. A comprehensive review of computing paradigms, enabling computation offloading and task execution in vehicular networks. *IEEE Access*, 10:3580–3600, 2022.

[WWC+21] Chong Wang, Lide Wang, Huang Chen, Yueyi Yang, and Ye Li. Fault Diagnosis of Train Network Control Management System Based on Dynamic Fault Tree and Bayesian Network. *IEEE Access*, 9:2618–2632, 2021.

[WWD+23] Gongcheng Wang, Weidong Wang, Pengchao Ding, Yueming Liu, Han Wang, Zhenquan Fan, Hua Bai, Zhu Hongbiao, and Zhijiang Du. Development of a search and rescue robot system for the underground building environment. *Journal of Field Robotics*, 40(3):655–683, 2023.

[WWZ+17] Qingyao Wu, Hanrui Wu, Xiaoming Zhou, Mingkui Tan, Yonghui Xu, Yuguang Yan, and Tianyong Hao. Online Transfer Learning with Multiple Homogeneous or Heterogeneous Sources. *IEEE Transactions on Knowledge and Data Engineering*, 29(7):1494–1507, July 2017.

[WZL+18] Tian Wang, Jiyuan Zhou, Anfeng Liu, Md Zakirul Alam Bhuiyan, Guojun Wang, and Weijia Jia. Fog-based computing and storage offloading for data synchronization in iot. *IEEE Internet of Things Journal*, 6(3):4272–4282, 2018.

[XDL+23] Bin Xu, Tao Deng, Yichuan Liu, Yunkai Zhao, Zipeng Xu, Jin Qi, Sitao Wang, and Dan Liu. Optimization of cooperative offloading model with cost consideration in mobile edge computing. *Soft Computing*, 27(12):8233–8243, June 2023.

[XDTZ20] Zhengzhe Xiang, Shuiguang Deng, Javid Taheri, and Albert Zomaya. Dynamical Service Deployment and Replacement in Resource-Constrained Edges. *Mobile Networks and Applications*, 25(2):674–689, April 2020.

[XKK20] Fatos Xhafa, Burak Kilic, and Paul Krause. Evaluation of IoT stream processing at edge computing layer for semantic data enrichment. *Future Generation Computer Systems*, 105:730–736, April 2020.

[XSB+18] Tianwei Xing, Sandeep Singh Sandha, Bharathan Balaji, Supriyo Chakraborty, and Mani Srivastava. Enabling Edge Devices that Learn from Each Other: Cross Modal Training for Activity Recognition. In *Proceedings of the 1st International Workshop on Edge Systems, Analytics and Networking*, pages 37–42, Munich Germany, June 2018. ACM.

[XZLH20] Xiong Xiong, Kan Zheng, Lei Lei, and Lu Hou. Resource Allocation Based on Deep Reinforcement Learning in IoT Edge Computing. *IEEE Journal on Selected Areas in Communications*, 38(6):1133–1146, June 2020.

[YKAQ22] Mohammad Yazdi, Faisal Khan, Rouzbeh Abbassi, and Noor Quddus. Resilience assessment of a subsea pipeline using dynamic Bayesian network. *Journal of Pipeline Science and Engineering*, 2(2):100053, June 2022.

[ZCC+23] Qiyang Zhang, Xiangying Che, Yijie Chen, Xiao Ma, Mengwei Xu, Schahram Dustdar, Xuanzhe Liu, and Shangguang Wang. A Comprehensive Deep Learning Library Benchmark and Optimal Library Selection. *IEEE Transactions on Mobile Computing*, pages 1–14, 2023.

[ZFFP24] Anastasios Zafeiropoulos, Nikos Filinis, Eleni Fotopoulou, and Symeon Papavassiliou. AI-Assisted Synergetic Orchestration Mechanisms for Autoscaling in Computing Continuum Systems. *IEEE Communications Magazine*, 2024.

[Zha20] Changhao Zhang. Design and application of fog computing and internet of things service platform for smart city. *Future Generation Computer Systems*, 112:630–640, 2020.

[ZJXZ23] Zhe Zhang, Ju Jiang, Haiyan Xu, and Wen-An Zhang. Distributed dynamic task allocation for unmanned aerial vehicle swarm systems: A networked evolutionary game-theoretic approach. *Chinese Journal of Aeronautics*, December 2023.

[ZKQ+24] Muhammad Zakarya, Ayaz Ali Khan, Mohammed Reza Chalak Qazani, Hashim Ali, Mahmood Al-Bahri, Atta Ur Rehman Khan, Ahmad Ali, and Rahim Khan. Sustainable computing across datacenters: A review of enabling models and techniques. *Computer Science Review*, 52:100620, May 2024.

[ZLC+19] Fenghua Zhu, Yisheng Lv, Yuanyuan Chen, Xiao Wang, Gang Xiong, and Fei-Yue Wang. Parallel transportation systems: Toward iot-enabled smart urban traffic control and management. *IEEE Transactions on Intelligent Transportation Systems*, 21(10):4063–4071, 2019.

[ZMC+22] Sm Zobaed, Ali Mokhtari, Jaya Prakash Champati, Mathieu Kourouma, and Mohsen Amini Salehi. Edge-MultiAI: Multi-Tenancy of Latency-Sensitive Deep Learning Applications on Edge. pages 11–20. IEEE Computer Society, December 2022.

[ZP94] N. Zhang and D. Poole. A simple approach to Bayesian network computations. In *Engineering-Economic Systems, Stanford*, 1994.

[ZRR+22] Zhizhou Zhang, Murali Krishna Ramanathan, Prithvi Raj, Abhishek Parwal, Timothy Sherwood, and Milind Chabbi. {CRISP}: Critical Path Analysis of {Large-Scale} Microservice Architectures. pages 655–672, 2022.

[ZTL+19] Fan Zhang, Xuxin Tang, Xiu Li, Samee U Khan, and Zhijiang Li. Quantifying cloud elasticity with container-based autoscaling. *Future Generation Computer Systems*, 98:672–681, 2019.

[ZZL23] Ziyang Zhang, Yang Zhao, and Jie Liu. Octopus: SLO-Aware Progressive Inference Serving via Deep Reinforcement Learning in Multi-tenant Edge Cluster. In *Service-Oriented Computing*, Cham, 2023.