TU WIEN Informatics

Efficient Transmission and Recovery of Salient Information

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

eingereicht von

Dipl.-Ing. Alireza Furutanpey, BSc
Matrikelnummer 01507319

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ. Prof. Dr. Schahram Dustdar

Diese Dissertation haben begutachtet:

Kerstin Bunte	Adlen Ksentini

Wien, 18. Oktober 2025

Alireza Furutanpey

TU WIEN Informatics

Efficient Transmission and Recovery of Salient Information

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Dipl.-Ing. Alireza Furutanpey, BSc
Registration Number 01507319

to the Faculty of Informatics

at the TU Wien

Advisor: Univ. Prof. Dr. Schahram Dustdar

The dissertation has been reviewed by:

Kerstin Bunte	Adlen Ksentini

Vienna, October 18, 2025

Alireza Furutanpey

Erklärung zur Verfassung der Arbeit

Dipl.-Ing. Alireza Furutanpey, BSc

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, 18. Oktober 2025

Alireza Furutanpey

Acknowledgements

First and foremost, I thank my parents for their unwavering support, and Trudi Hess for hers toward my family. The completion of this thesis marks nearly a decade spent studying, teaching, and researching in Computer Science. There were moments when I wanted to abandon it all. Not because of the technical challenges, but due to circumstance and internal conflict, as I came to terms with the realities of academia. Yet I do not regret this path. In part, it is the satisfaction of solving demanding problems that, while not world-changing, have reached and inspired others, some of whom expressed gratitude in personal correspondence. More importantly, it allowed me to meet people I could not do justice to, even if I spent the remainder of the thesis describing how fond I am of them. Thomas Rausch for recognizing how unhappy I was with my earlier trajectory. Waldemar Hummer for always making time when I have requested his counsel. Alexander Knoll, for his support, kindness, and friendship. Renate Weiss, Christine Kamper, Margret Steinbuch, and Clarissa Schmid for providing invaluable assistance that allowed me to focus on research and teaching. Andrea Morichetta for bailing me out in Chicago and sharing the kind of hopeful visions that sustain you. Victor Casamayor Puyol, for his presence alone, could lift morale. Pantelis Frangoudis for reminding me not to take everything too seriously, tempering my Reviewer 3-induced rage, and always being there to listen. Kerstin Bunte for her hospitality and for helping me improve as a researcher. Matthias Wödlinger for sharing a common passion that helped me not to question my sanity too often. Florian Kowarsch for always motivating me to raise my standards and for having discussions that were among my greatest sources of inspiration. My students, especially Marvin Seidl, who may have learned research from me but taught me much about engineering during our time as undergrads. The people at AWS who have influenced the later stages of my PhD. In particular, Valentin Flunkert, whose talent is matched only by his kindness. Anastasiya Danilenka, for being a wonderful collaborator and friend. Philipp Raith, for not only enduring my antics and practical jokes but also becoming one of my closest friends.

Lastly, I owe the deepest gratitude to Schahram, my advisor and mentor, for his wisdom, patience, and trust. I will be forever grateful for his guidance, which was pivotal to my growth.

The research presented in this thesis was funded by TU Wien research funds.

Kurzfassung

Diese Dissertation widmet sich grundlegenden Problemen, die durch die Verbreitung von Geräten in zunehmend verteilten Systemen entstehen. Während Ressourcen von zentralisierten Clouds zu einem heterogenen Edge-Cloud-Kontinuum verteilt werden, überlasten Datenmengen aus Inferenzanfragen und die für das Sammeln erforderliche Telemetrie die Netzwerke. Bestehende Lösungen können die anspruchsvollen Anforderungen von Edge-Cloud-Systemen nicht angemessen erfüllen, da sie zu komplex für die Einführung sind, Ergebnisse liefern, die schwierig zu reproduzieren sind, oder die Grundursachen nicht behandeln. Wir fokussieren uns auf Methoden, die mit geringem Aufwand erhebliche Verbesserungen bei der Integration in bestehende Systeme erzielen. Eine minimal-opinionierte Referenzarchitektur betont die Kompatibilität mit bestehenden Systemen und die Bedeutung der Beobachtbarkeit für automatisierte Entscheidungsmechanismen wie die Terminplanung. Eine strenge empirische Methodik für Machine-Learning-Forschung in Edge-Cloud-Systemen zeigt die Bedeutung der Einfachheit beim Entwurf von Methoden, die in komplexen Systemen laufen sollen. Ein Trainingsalgorithmus und ein neuartiger Wissensdestillationsansatz für aufgabenunabhängige Kompression erreichen erhebliche Datenratenreduzierungen und übertrifft sogar die konkurrenzfähigsten Baseline-Verfahren. Das Grundmodell besteht aus einem Encoder mit nur 140.000 Parametern und ist effizient genug, dass die Latenzstrafe durch den Kodierungs- und Dekodierungsoverhead mehr als durch die reduzierten Übertragungskosten ausgeglichen wird. Eine Erweiterung behandelt Netzwerke, in denen Konnektivität nur intermittierend verfügbar ist und Durchsatz bevorzugt wird, wodurch das herunterladbare Datenvolumen um über zwei Größenordnungen erhöht wird. Schließlich verbessert eine Methode, die Algorithmen zur Konstruktion statistischer Zusammenfassungen für die Überwachung erweitert, deren Kodierungseffizienz. Die Erweiterung erhält nachweislich die zugrunde liegenden mathematischen Garantien des Basisalgorithmus und die Kompatibilität mit bestehenden Systemen.

Zusammen schaffen die Beiträge eine prinzipielle Grundlage für skalierbare, transparente und kodierungseffiziente Systeme, die nur die wichtigsten Informationen unter strengen Beschränkungen für neue Paradigmen im verteilten Computing wiederherstellen und übertragen.

Abstract

The thesis concerns fundamental problems that emerge from the proliferation of devices in increasingly distributed systems. As resources migrate from centralized clouds to a heterogeneous edge-cloud continuum, payloads from inference requests and telemetry required for monitoring overwhelm networks. Existing solutions cannot adequately meet the demanding requirements of edge-cloud systems as they are too complex for adoption, report results that are virtually impossible to reproduce, or do not address root causes. We follow a strict bottom-up approach, focusing on simple methods that yield significant improvements with little effort to integrate into existing systems. A minimally opinionated reference architecture emphasizes legacy compatibility and the importance of observability for automated decision mechanisms, such as scheduling. A rigorous empirical methodology for Machine Learning research in edge-cloud systems demonstrates the importance of simplicity when designing methods that are expected to run in complex systems. A training algorithm and novel knowledge distillation approach for task-agnostic compression achieves significant rate reductions, outperforming even the most competitive baseline. The base model consists of an encoder with just 140,000 parameters and is efficient enough for the latency penalty from the encoding and decoding overhead to be more than offset by the reduced transmission costs. An extension handles networks where connectivity is only intermittently available and throughput is favored, increasing downlinkable data volume by over two orders of magnitude. Finally, a method that augments algorithms for constructing statistical summaries for monitoring improves their coding efficiency. The augmentation provably maintains the base algorithm's underlying mathematical guarantees and compatibility with legacy systems.

Together, the contributions establish a principled foundation for scalable, transparent, and coding-efficient systems that recover and transmit only the most salient information under stringent constraints for emerging paradigms in distributed computing.

Contents

Publications

The research presented in this thesis is partly based on the following publications:

- Furutanpey, A., Barzen, J., Bechtold, M., Dustdar, S., Leymann, F., Raith, P., and Truger, F., 2023, July. Architectural vision for quantum computing in the edge-cloud continuum. In 2023 IEEE International Conference on Quantum Software (QSW) (pp. 88-103).
- Furtuanpey, A., Raith, P., and Dustdar, S., 2024. FrankenSplit: Efficient Neural Feature Compression with Shallow Variational Bottleneck Injection for Mobile Edge Computing. IEEE Transactions on Mobile Computing.
- Furutanpey, A., Zhang, Q., Raith, P., Pfandzelter, T., Wang, S. and Dustdar, S., 2025. FOOL: Addressing the Downlink Bottleneck in Satellite Computing with Neural Feature Compression. IEEE Transactions on Mobile Computing.
- Furutanpey, A., Frangoudis, P.A., Szabo, P. and Dustdar, S., 2025. Adversarial Robustness of Bottleneck Injected Deep Neural Networks for Task-Oriented Communication. In 2025 IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN).
- Furutanpey, A., Walser, C., Frangoudis, P.A., and Dustdar, S., 2025. Leveraging Neural Graph Compilers in Machine Learning Research for Edge-Cloud Systems. Under Review.
- Furutanpey A. and Flunkert V., 2025. Codeword Optimal Quantile Approximation. Under Review.

CHAPTER 1

Introduction

Mobile data traffic is growing with an annual rate of more than 15% and is expected to double by 2030, with 5G subscriptions alone projected to reach 6.3 billion [Eri25]. The surge in bandwidth demand underscores the fundamental trade-offs between communication cost, latency, and information fidelity, motivating new frameworks for inference in distributed systems. In particular, efficient data transmission and reliable recovery of salient information are indispensable for semi- or fully decentralized platforms with resources organized in hierarchical networks.

1.1 Motivation

The increased data availability, specialized hardware, and algorithmic advancements have elevated Artificial Neural Networks (ANNs) as an enabler for numerous problem domains. As foundational models become more reliable, platforms can accommodate a wide range of applications using existing cloud infrastructure. When applications are latency sensitive, computation may be pushed to mobile clients at the network's edge by applying model compression methods, such as quantization [JKC+18, LHC+24], and knowledge distillation [GYMT21, WY22]. Problems arise when there are stringent constraints on the solution quality and request completion times [Sat17]. The edge-cloud continuum addresses the limitations of cloud and edge computing by organizing resources hierarchically in a distributed network ranging from constrained edge devices to cloud data centers [DPD22]. The paradigm promises increased resource efficiency and meets the requirements of even the most demanding applications with the same convenience for client programmers as public cloud computing service providers. A basic requirement is observability since automated decision mechanisms, such as orchestrators and load balancers, cannot function without information on the constantly updating system states. Observability requires extracting telemetry from an increasing number of devices. For

example, the EU Rolling Plan for ICT standardization predicts that by 2030, 50 billion connected devices worldwide will be in use, generating zettabytes of data [Com25].

1.2 Problem statement

While walled garden edge-cloud offerings exist for select services [XFM+21], semi- or fully decentralized platforms still have to emerge [NRF+22, RD21]. We identify the data volume generated by ANN inference requests and required by monitoring services as the primary inhibitor for scalability. Therefore, the thesis is concerned with the fundamental problem of managing the accelerating bandwidth costs as systems become increasingly distributed. The challenges it addresses are motivated by basic requirements of a semi- or fully decentralized platform [RRFD23]. We refer to a platform as a system of systems where clients can indiscriminately deploy their applications. The platform handles scaling and managing the underlying hardware infrastructure. Platforms may exploit locally available resources at end devices, fog nodes from telecommunication providers, or source existing infrastructure from third parties, such as VMs hosted in data centers from public and private cloud services [RLF+20]. Client programmers enter contracts with platforms based on application requirements. The contracts are typically expressed as Service Level Agreements (SLAs) and consist of multiple Service Level Objectives (SLOs)[NMP+20]. Beyond providing a reference architecture, we do not pile onto the countless studies that introduce opinionated suggestions on implementation details, advocate going in a certain direction, or solve toy placement and scheduling problems. Instead, the thesis views the problems through the lens of an *operator* of a distributed platform. The operator represents a collective of engineers and system designers tasked with building the platform.

The thesis empathizes with the operator and does not wish to further burden them with fantastic ideas or over-engineered systems accompanied by elaborate diagrams. It acknowledges that, stripped of all complexity, obfuscation, and mysticism, the edge-cloud compute continuum is an exercise in system integration. A simple problem, yet arguably the most ambitious paradigm of a distributed system. Worthwhile contributions must be bottom-up, easily adoptable, and enable the emergence of such platforms. Accordingly, the thesis makes minimal assumptions about the system design and components of the platform. The following briefly describes the components and challenges we address within the thesis's scope. In-depth technical explanations are deferred to later chapters.

1.2.1 Data Compression

While building a compute continuum is an exercise in system integration, scaling it is primarily problem in data compression. To significantly reduce data volume, lossy compression algorithms are required as perfect reconstruction of the input is needlessly strict. The challenge is to design algorithms that are *semantically* lossless, focusing on salient information and keeping the overhead to recover context minimal. The following

elaborates on what semantic losslessness implies for the two system components and their respective modalities considered in the thesis.

Distributed Inference Engines Inference Engines are APIs that client programmers can call for *intelligent tasks*. We refer to intelligent tasks as problems where classical control structures cannot provide solutions tractably or with sufficient precision [FBB+23]. Operators may easily fulfill prediction performance-related targets by offloading requests to cloud-style data centers. However, exclusively relying on offloading leaves resources at or close to the device idle, forcing application instances to compete for bandwidth. A popular approach is partitioning ANN inference by deploying partitioned models across the network [WL23]. This may solve memory challenges without degrading prediction performance using quantization, but its applications are limited when the primary concern is latency, and it is impractical to adopt for platforms.

At the application layer, the dominant modalitiesies are visual, such as video feeds generated by camera sensors [Cis20]. Transform coding is the commonly used framework for lossy compression. Using rate-distortion theory, the objective is to find the number of bits (bitrate, or rate) to store an encoded representation to restore it according to a set distortion constraint [Sha59]. Here, semantic losslessness refers to a distortion constraint that empirically quantifies whether we hit the prediction quality-related SLO.

Monitoring & Data Analytics Statistical summaries of distributions are a fundamental building block in data analytics and monitoring system states [TMN20]. Monitoring the system state to pass enough information to an orchestrator is particularly taxing on infrastructure. A distributed platform where local resources may be scarce increases the importance of an accurate scheduler while making it more challenging to schedule accurately. Compared to a centralized platform, federations are less demarcated, dropouts are more frequent, hardware is less standardized, and networks are more heterogeneous [Rau21]. Irrespective of how well a scheduler may work in sterile benchmarking environments, it will always be limited by the quality and recency of the information on which it can base decisions. Platforms must also offer monitoring and analytics services to clients for their applications. Assuming a central entity with a fully observable system state by aggressively extracting telemetry is not feasible, as bandwidth is already scarce to serve requests due to application payloads. For example, with a research extension for Geo-Distributed Kubernetes, centrally collecting status information from 500 clusters, each hosting only a *single* worker and control plane node, was shown to incur a monitoring and traffic overhead of 230 Mbit/s for the centralized data collecting [HP24]. Note that this figure involves *only* resource-level metrics and does not consider more detailed application-level probes.

For telemetry and monitoring, the modality is univariate data streams. TC is unsuitable for this modality, since the data must be processed during ingestion in at most small buffered chunks [WM15]. Instead, the principled approach to lossy compression of streams is creating summaries using sampling and sketch algorithms [CY20]. The thesis considers

the class of algorithms with strong guarantees [CV20], so semantic losslessness refers to provably maintaining equivalent guarantees.

1.2.2 Operational Challenges

Operational challenges in this thesis refer to the effort required to support a particular application type and to adopt a method that enables or further improves the efficiency of its associated deployment style.

Server-Side Transparency and Compatibility

As seamless integration is a primary concern in the compute continuum, methods that maintain server-side transparency are invaluable. When transparency is strict, operators can use existing legacy infrastructure without implementing modifications and rules for each service, client application, or subsystem.

Figure 1.1 illustrates a generic scenario where the operator maintains three sizes from the same model family, each associated with a different cost and expected accuracy.

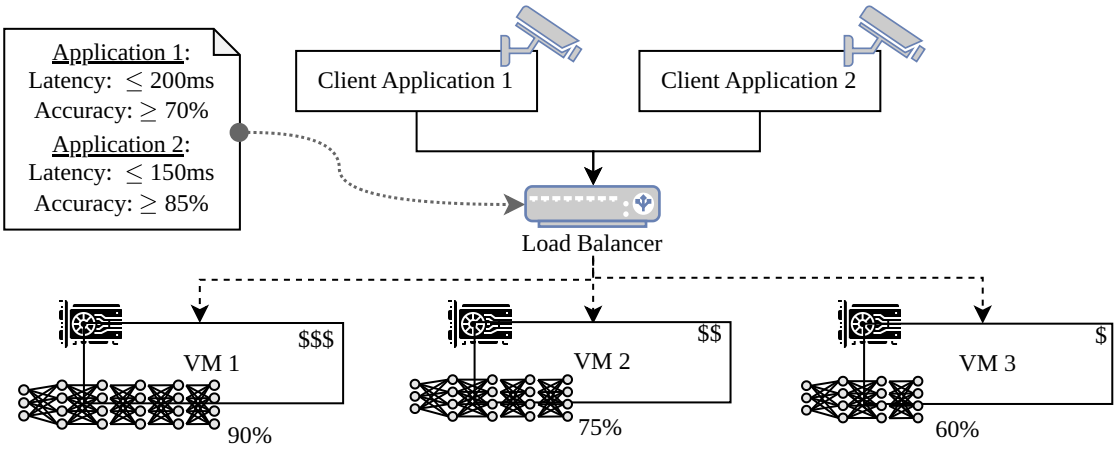


Figure 1.1: SLO-aware Load Balancing.

The load balancer aims to meet the latency and prediction accuracy SLO target for two client applications and keeping the platform's cost down. The platform can trivially meet the accuracy target by routing all requests to the largest model. It may manage peak hours and keep costs down with a more informed strategy without breaching SLAs by temporarily routing requests to smaller models. Now, assume an operator wishes to improve resource efficiency by drawing from local resources where possible. Assume operators apply a common approach to the above-discussed ANN partitioning that requires finetuning the deeper layers deployed at the server [FRD24]. Since end devices may not have the resources for local computation, operators must maintain two sets of weights to route requests to the fine-tuned or finetunedights conditionally. Expecting an operator to maintain multiple versions of the same architecture for modest efficiency gains

is unreasonable. Clearly, methods intended to solve problems concerning the edge-cloud continuum must aim for transparency and compatibility with existing infrastructures.

Next, consider that platforms must extract telemetry for application analytics or system monitoring. Figure 1.2 illustrates an operator integrating the monitoring services of their platform seamlessly using existing infrastructure.

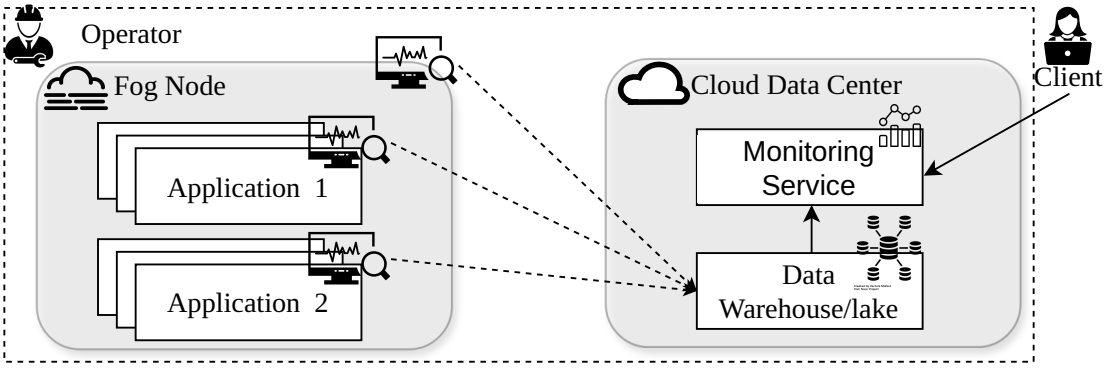


Figure 1.2: Extracting Telemetry for Monitoring.

This is only possible when relying on the same algorithm to create statistical summaries [CY20, WM15, GZR23]. As devices and applications proliferate, operators may need to find novel algorithms that can reduce bandwidth requirements instead of reducing the data resolution. However, operators must progressively introduce modifications and special rules if the algorithms break server-side transparency.

Performance-Critical Applications

An application is *performance-critical* when it requires state-of-the-art prediction performance from large models. Operators can support various applications using foundational models. A foundational model may be a feature extractor where clients upload a small dataset to tune predictors, or supports numerous tasks without requiring additional data. Figure 1.3 exemplifies the former.

Either way, end-devices *offload* requests to remote servers where resources are seemingly horizontally scalable.

Latency-Sensitive Applications

An application is *latency-sensitive* when milliseconds determine the difference between an acceptable and unacceptable user experience. For example, Augmented Reality (AR) applications require less than 16 milliseconds to achieve perceptual stability [HCH+14]. For such applications, operators may push computation to end-user devices as illustrated in Figure 1.4.

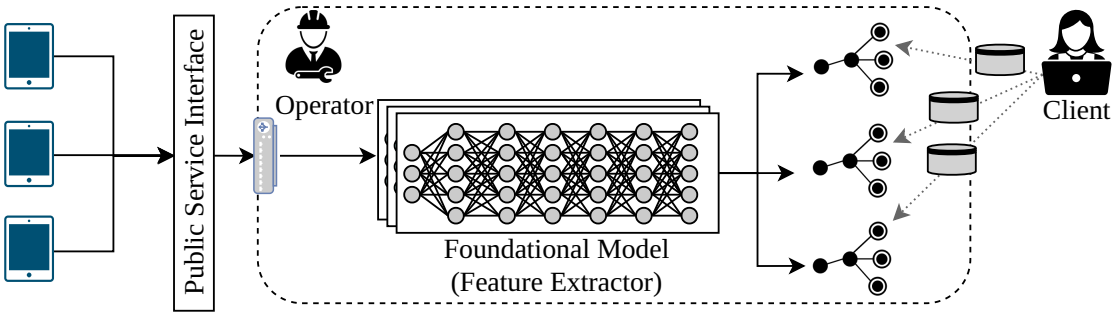


Figure 1.3: Operator maintains foundational models and handles horizontal scaling. Clients supply datasets to support their applications.

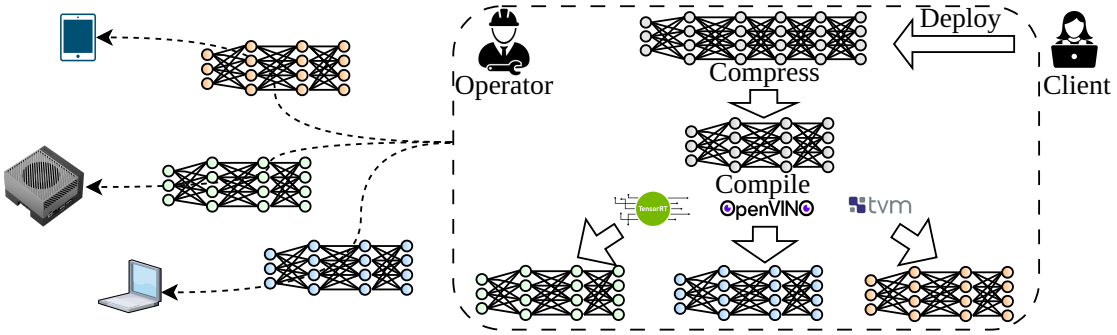


Figure 1.4: Clients provide the model for their application. Operator handles optimization and deployment to end-devices with varying properties

Since the local resources of end-devices are constrained, it is necessary to compress and optimize the model for the target hardware [DLH+20]. Managing model optimization and deployment for clients incurs significantly more operational overhead than processing requests server-side with foundational models. End-user devices[1] are less predictable and more heterogeneous than data center hardware. Vendors have varying support for the underlying operations of ANN architectures and maintain different, often proprietary, software stacks for optimization. The NVIDIA Jetson device lineup has TensorRT, Intel CPUs are surprisingly potent for ANN inference with OpenVINO, and ApacheTVM search heuristic is a powerful default option for arbitrary edge devices without dedicated software support [FWR+25].

Performance-Critical and Latency-Sensitive Applications

Most challenging are application types that are latency-sensitive *and* performance-critical. The deployment strategy is a combination of the previous two operators must install edge servers or provision fog nodes near clusters of end-user devices. Such servers may have high-end hardware analogous to data centers, but horizontal scaling is limited. During

[1] End-user devices may also refer to small devices in proximity to the application host.

peak hours, schedulers must carefully decide which requests can temporarily be rerouted to more remote servers without breaching SLAs. Without mitigation strategies, the limited bandwidth will inevitably result in network congestion whenever numerous clients cluster. For platforms to scale in such situations, they require fast methods to reduce transmission costs *and* request latency, irrespective of where the request is routed. The challenge is to design lightweight encoders that can significantly reduce transmission costs by exploiting local resources while maintaining prediction quality constraints on downstream tasks. This strategy corresponds to the distributed inference engine, which we will illustrate and elaborate on in later chapters.

1.2.3 Research Questions

Three overarching research questions drive the thesis.

RQ1. How can we design (I) Methods and (II) Experiments for Machine Learning Research in Edge-Cloud Systems? Evaluating rate-distortion performance is straightforward with standardized benchmarks, such as Kodak[Fra99] and CLIC [TST+20]. However, designing methods and experiments yielding meaningful empirical evidence on efficacy in Edge-Cloud systems under real-world conditions has two distinct challenges.

1. *What is the real gain for operators when implementing a proposed method?* Channel conditions and hardware are highly heterogeneous and progressively advancing. Experiments for data-driven compression algorithms that learn nonlinear transforms using ANNs must consider the increasingly diverse landscape of special-purpose accelerators [SICM23]. Even when hardware has comparable specifications, vendor-locked software optimization often leads to incomparable performance in real-world runtime environments.

2. *How can operators assess the adverse effects of lossy compression for downstream prediction?* Conventional reference-based metrics from image compression, such as Peak Signal-to-Noise Ratio (PSNR), Multi-Scale Structural Similarity Index Measure (MS-SSIM), or Learned Perceptual Image Patch Similarity (LPIPS), cannot provide conclusive evidence on whether an algorithm reduces the performance of prediction models on unknown tasks.

RQ2. What approach exploits local resources best to (I) transparently meet the requirements of (II) latency-sensitive and (III) system-critical applications? End-user devices are increasingly equipped with powerful special-purpose hardware that we aim to exploit to meet the requirements of demanding applications. From the problem statement, we derive the three requirements to assess whether an approach makes the best use of locally available resources

1. *How high is the operational burden on the operator and how much knowledge on client tasks is required?* We seek methods that generalize to (near-) arbitrary tasks with minimal assumptions on their properties. Competitive methods are self-supervised and function without access to a labeled dataset. They exhibit strong results on tasks without seeing the test *and* train set. Their adoption should require minimal operational overhead, for example, by *not* introducing complex runtime systems or finetuning large foundational models.

2. *Under what conditions does the encoding and decoding latency penalty outweigh the transmission cost reduction?* Hardware and channel properties determine the conditions. The more constrained the hardware and the less constrained the channel, the more challenging it is to reduce request latency through compression. A competitive method will yield lower request latency even for lower-end user devices in urban environments with developed and highly available infrastructure. When connectivity is only sporadic or in fixed intervals, the priority is on processing throughput.

3. *Is the data integrity preserved?* Substantial reductions in data volume require intrinsically lossy methods. Accordingly, we must determine whether salient information is preserved. Human interpretability is an implied requirement, even if the primary criterion is the performance of downstream prediction. Client applications may need to store imagery when ANNs are used for decisions. Especially when human costs are involved, review and intervention by human experts must be possible. Note that this requirement is distinct from perceptual quality, which focuses on realism [The24, BM19, BM18].

RQ3. What are (I) suitable approaches to construct statistical summaries, and (II) can we improve their coding efficiency without breaking server-side transparency? Unlike RQ2, this RQ concerns data volume generated to support the underlying systems and services, not the applications that turn on them.

1. *What are suitable algorithms to construct statistical summaries in heterogeneous environments?* Here, heterogeneity refers to the statistical properties of telemetry resulting from hardware and application properties. An algorithm must handle a large data stream with minimal assumptions on input properties and construct summaries that operators and clients can trust to base decisions on.

2. *Can we improve the coding efficiency of existing summarization algorithms without breaking server-side transparency?* Maintaining compatibility with existing systems requires *strict* server-side transparency. The challenge is to improve existing algorithms encoding or serialization logic that maintains compatibility with server-side decoding or deserialization.

1.3 Methodology & Contributions

The research questions closely align in their objective, but adequately answering them requires varying methodologies. Our studies are predominantly empirical, drawing from image compression that benchmark rate-distortion performance on standardized datasets [YMT23], complemented with typical methodologies from systems research to assess the system's efficacy in real-world environments [RRP+22]. Still, we perform extensive theoretical analysis where safety guarantees are necessary. In answering the research question, we contribute architectures, frameworks, algorithms, and complete systems. The core contributions are novel and designed from the ground up to address problems of the compute continuum. They significantly progress the state-of-the-art, not just in isolated use cases with over-optimized implementations, but as general approaches in their corresponding problem domain. Limitations and open problems remain, which are transparently discussed. We ensure reproducibility by extensively describing our experiment designs, open-sourcing the accompanying code, and the experiment frameworks. Indeed, at the time of writing, studies that reproduce results and use the published methods included in the thesis as baselines have already emerged [ZC25, FLW+25].

The following summarizes the thesis in four main contributions, each mapping to open challenges and the research questions. The thesis structure follows the contributions enumeration. **C1.** provides the high-level context of the problem statement and RQs. **C2** maps to RQ1, **C3.** to RQ2, and **C4** to RQ3.

C1. A Complete Reference Architecture for a Hybrid Distributed Platform ANN inference workload has special treatment as specialized hardware accelerators are integral to every computational tier. However, the definition of intelligence as a means to solve problems unfeasible for classical control structures is not limited to classical ML. Since the architecture aims to be a *complete* reference, it acknowledges the importance of quantum chips as a new class of specialized hardware. Analogous to mobile accelerators for ANNs, the architecture enables applications to exploit (mobile) Quantum Processing Units (QPUs) for an orthogonal class of intelligent tasks, and to enhance the solution quality or efficiency of ANN inference.

We draw parallels between past work on edge intelligence in classical computing and integrating quantum resources into hybrid systems. We elaborate on monitoring, detailing the levels of observability necessary for schedulers to cope with the complexity in a distributed platform. Lastly, we discuss solution approaches for distributed inference engines. The contribution provides the overarching context that motivates the core contributions. At the time of writing, the corresponding publication [FBB+23] is one of the most viewed and cited documents within all proceedings of the conference where it was presented[2].

[2] https://ieeexplore.ieee.org/xpl/conhome/1847584/all-proceedings/popular

C2. Designing and Evaluating ML Methods in Edge-Cloud Systems Graph compilers that optimize computational graphs of neural networks can improve latency or throughput by orders of magnitude without loss in prediction performance. This contribution shows that it is crucial to understand how graph compilers can invalidate relative performance differences between architectural archetypes of neural networks.

We introduce a framework to utilize such tools for designing, implementing, and deploying experiments in research for Edge or Edge-Cloud systems that rely on neural components [FWR+25]. We perform a comprehensive empirical analysis across varying architectural families on a heterogeneous physical testbed. We demonstrate how vendors prioritize optimizing different layer compositions using vendor-agnostic and vendor-specific graph compilers. While the corresponding study of this contribution was conducted and published towards the end of the thesis, we could see how architectural types see varying degrees of support from preliminary results. Accordingly, we focus on fundamentals, such as training objectives, and deliberately follow a simple encoder design, using widely supported layers.

C3. End-to-End Systems for Neural Feature Compression We introduce Shallow Variational Bottleneck Injection (SVBI), which dedicates local resources exclusively to compression. The encoder achieves considerable rate reductions and is task agnostic. Operators may deploy a single encoder to end-devices and support arbitrary network architectures for downstream tasks. Server-side transparency is maintained by not requiring labeled datasets or finetuning foundational models. Meaningful theoretical analysis of compression algorithms with non-trivial sources in complex systems is not feasible. Hence, studies are empirical, and multiple publications are associated with this contribution to provide sufficient evidence. The initial work introduces the general approach and a novel distillation algorithm [FRD24]. It exhaustively evaluates on standardized datasets and request times under varying channel conditions. Besides requiring only a single encoder and maintaining server-side transparency, it significantly outperforms the then state of the art in split computing. A follow-up considers the opposite extreme, where data is generated at the space's edge rather than in urban environments [FZR+25], addressing the downlink bottleneck in satellite computing, where network connectivity is only intermittently available. It extends SVBI to improve compression performance and introduce image recovery components. Evaluation of satellite imagery from several datasets shows that the system increases downlinkable data volume by two orders of magnitude.

C4. Transparently Improving the Coding Efficiency of Statistical Summaries

The contribution concerns randomized online quantile approximation with strong guarantees for efficient data transmission and persistence in distributed systems. While existing research has primarily focused on minimizing in-memory representation size [CV20], optimizing their encoding has seen little attention.

We extend the comparison-based computational model with a communication model that enforces receiver-side transparency to ensure compatibility with legacy systems. A verification procedure formally maintains the strong underlying mathematical properties. We introduce a principled method for augmenting existing algorithms that yield optimal codeword length with virtually no overhead to the core summarization procedure and strictly maintain server-side transparency. Additionally, we generalize the approach to jointly encoding locally available summaries. Extensive experiments supplement the formal analysis to demonstrate the efficacy of augmented algorithms, with further gains in coding efficiency when jointly optimizing sequentially constructed summaries.

CHAPTER 2

A Reference Architecture for the Edge-Cloud Continuum

This chapter introduces an architecture that provides an overview of deployment strategies, focusing on the inference engine, and explaining the importance of telemetry in edge-cloud systems. The architecture is minimally opinionated, follows established best practices, and is considerate of what the software architecture research community is familiar with. The novelty is the integration of quantum resources and the more thought-out components to provide schedulers with sufficient observability.

2.1 Integrating and Provisioning Resources in Hybrid Compute Platforms

Deploying edge applications on mobile quantum devices is approaching with the recent advancements of diamond-based QPUs [Gmb] that allow quantum computation at room temperature [Ltd23]. Hence, quantum computers may become widely available for individuals and organizations.

2.1.1 Introduction

Noisy intermediate-scale quantum (NISQ) computers are error-prone, contain only a limited number of qubits, and impose restrictions on the depth of successfully executable circuits [LB20]. Yet, algorithms tailored towards NISQ devices started to demonstrate the viability of quantum computers in various fields, ranging from molecule simulation [GEBM19] to machine learning [CVH+22] and optimization problems [CK19]. Evidently, to advance research and development into practical applications of quantum algorithms, increasing the accessibility of quantum computers by introducing adequate abstractions is effective. Owing to the limited availability, complexity, and cost of QPUs,

quantum computation for the masses may currently only be viable through cloud services that can hide the low-level machinery behind a convenient interface. However, while cloud providers can decrease the complexity and cost, we cannot exclusively rely on the efforts of hardware manufacturers to increase the accessibility of resources by simplifying the production and installation of QPUs. Instead, we must draw from our experiences in classical computing on the long-term limitations of relying exclusively on centralized public cloud platforms. Besides the privacy-related risks of entrusting third-party providers with sensitive data, the cloud computing paradigm bears numerous downsides, such as vendor lock-in and data centers posing a single point of failure vulnerable to outages. Additionally, a narrow cloud-centric view is inefficient, leaving resources closer to the client idle by indiscriminately offloading tasks to remote data centers.

The edge-cloud continuum addresses the limitations of cloud computing by organizing resources in a hierarchical distributed network ranging from constrained edge devices to cloud data centers. After decades of relying on centralized architectures, the transition is slow, with semi- or fully decentralized platforms still needing to emerge [NRF+22, RHS+21a, RRFD23]. The chapter elaborates on the potential of edge-cloud continuum for classical and quantum computing while explaining the interplay of numerous classical components a system must stitch together into a cohesive unit. We design a complete reference architecture for a distributed hybrid platform that can automate orchestrating hybrid applications. For components most relevant to the thesis, we describe key implementation challenges and possible solution approaches. The focus is on exploiting hybrid resources from a hierarchical network and explaining the problems not covered in the literature on classical computing when integrating numerous classical and quantum components into one cohesive unit.

2.1.2 Background & Related work

Cloud-centric platforms have paved the way for cost-efficient and large-scale applications to be accessible to the public. However, the emerging edge-cloud continuum accentuates the drawbacks of centralized architectures. Promising application paradigms, such as Edge Intelligence [DZF+20], heavily rely on the edge-cloud continuum and require autonomous management over the large and heterogeneous system. The success of these applications is tied to available platforms that need to support developers in designing, writing, testing, deploying, and managing them. This section introduces concepts fundamental to our architectural vision and summarizes related work.

Orchestration

The services of centralized platforms that provide access to quantum computers can be combined with classical applications. For example, Amazon offers event-based processing for its quantum offerings. This forces practitioners and researchers to build hybrid applications by manually combining separate quantum and classical components. They are further burdened with selecting different QPU technologies, devices, and com-

pilers [GARV+21]. Despite quantum applications consisting of classical and quantum components, they follow the same framework as classical computing in dividing orchestration into workflow technologies managing control flows, and provisioning technologies handling the deployment of application components [WBLZ21]. Hence, we can reduce infrastructure complexity by extending existing systems to support quantum applications. Wild et al. present Tosca4Q, which extends Tosca to support workloads relying on quantum computers [WBH+20]. Weder et al. introduce Quantum Application Archives (QAAs), allowing orchestration methods to treat quantum applications as self-contained entities [WBLZ21]. Later, Leymann et al. propose extending QAAs through a marketplace, with an architecture for a collaborative software platform to consider the development process [LBF+20].

Quantum Platforms

Several cloud offerings provide access to Quantum Computing as a Service (QCaaS), but it is burdensome to integrate managed quantum services cohesively into classical applications. Garcia-Alonso et al. [GARV+21] present their proof-of-concept implementation of a Quantum API Gateway, recommending a quantum computer target to run a given quantum application for Amazon Braket[1]. Beisel et al. [BBG+23] propose *Quokka*, a microservice-based framework to model and deploy quantum workflows. They propose a set of microservices that model the typical quantum workflow based on *Variational Quantum Algorithms (VQAs)* [CAB+21]. This workflow comprises circuit generation, execution, error mitigation, objective evaluation, and parameter optimization. The advantage of the approach is the complete decoupling of pre-processing, execution, and post-processing that follows a flexible workflow definition. Salm et al. [SBB+20] present a concept that automatically handles the analysis of quantum algorithms and the selection of quantum computers. Grossi et al. [GCA+21] build a prototypical platform inspired by Serverless Computing through which quantum developers can deploy their applications. They employ a scheduler that focuses on queue management and result retrieval. Leymann et al. [LBF+20] propose an architecture for a collaborative software platform for quantum applications that encompasses the development process and deployment aspect through a marketplace for quantum applications.

The studies so far have shown how platforms can enhance collaboration, improve the development of applications, and simplify deployment aspects, such as dynamically selecting an adequate quantum computer. Extensions, such as modeling serverless applications [WBK+18, WBH+20], to edge-cloud systems are required to realize the seamless integration of resources in a hierarchical network.

Serverless Edge Computing

A key problem of edge-cloud applications is the autonomous orchestration of applications that exploit resources at different tiers in the network [DZF+20]. Manual management is

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

infeasible in these large-scale and geo-distributed infrastructures, so a platform that can autonomously manage application deployments is required.

Serverless Edge Computing is the extension of Serverless Computing that abstracts the underlying infrastructure and transparently deploys applications packaged as functions across edge-cloud systems [ATC+21, NRF+22]. We argue that autonomous management and simplified application development and deployment are enablers of the emerging quantum computing paradigm. Nguyen et al. [NUB24] present a holistic serverless platform that supports classic, quantum, and hybrid applications. Conversely, we envision a platform that spans the edge-cloud continuum and manages application deployments across heterogeneous infrastructures. The increased complexity stems from the composed applications and the sophisticated and fine-grained monitoring.

Task Partitioning

Task partitioning in classical edge computing and quantum computing are two distinct research areas that address orthogonal problems. Still, they share a common motivation in dividing a task into subtasks executable by geo-distributed nodes to handle resource limitations or increase resource efficiency.

Partitioning in classical edge computing concerns distributing load for resource efficiency [LLJL19]. In quantum computing, splitting tasks between classical and quantum nodes is necessary for near-term applications to cope with the limitations of NISQ devices [LB20], and most common hybrid classical-quantum splitting patterns assign fixed roles to components [WBLV21a]. Patterns for quantum computation are typically designed to execute a particular class of algorithms and do not consider applications where a quantum algorithm is just one of several subtasks [WBLV21b]. As the limitations of quantum computation will gradually diminish, a platform should be able to accommodate new emerging patterns. Further, for near- and long-term QPUs, the platform should dynamically adjust the workload between quantum and classical nodes according to target Service Level Objectives (SLOs), internal and external conditions, such as load and bandwidth, respectively.

Variational Quantum Algorithms *Variational Quantum Algorithm (VQA)* is a generic framework for optimizing the parameters of a quantum circuit on a classical computer [CAB+21]. Depending on the target task, we can derive more specific algorithms, such as *Variational Quantum Eigensolver* for approximating the lowest eigenvalue of a matrix [TCC+22] or *Quantum Approximate Optimization Algorithms (QAOAs)* to approximate the solution of a combinatorial optimization problem [SA19]. Another notable instance of VQAs is *Quantum Neural Networks (QNNs)*, which aim to improve the representation of classical neural networks with embeddings in the Hilbert space. Note, in literature, the distinction between VQAs, Quantum Machine Learning (QML), and QNNs is blurry; thus, for clarity, we refer to QNNs as models that are built and trained for typical ML tasks, such as Feature Extraction, Regression, or Classification.

Warm-Starting The term *warm-starting* is ambiguous due to its widespread usage in classical and quantum computing. For example, it may refer to techniques that reduce resource usage in machine learning and optimization [AA20]. In classical serverless computing, warm-starting typically relates to methods for preparing execution environments, such as reusing running containers [MEHW18]. For the remainder of the thesis, warm-starting refers to a general strategy for partially computing or preparing a quantum algorithm's output on auxiliary devices. Notably, warm-starting methods are not limited to classical-to-quantum and may be quantum-to-quantum or quantum-to-classical.

Hybrid classical-quantum systems can benefit from various warm-starting methods that utilize previously obtained solutions, approximations, or trained models [TBB+24]. For example, following the assumption that optimal variational parameters for similar problem instances solved with VQAs are in proximity, parameters can be transferred between instances as an initial point to warm-start from and improve upon [GLL+21]. Moreover, approximations that are cheaply generated by efficient classical algorithms can be utilized to initialize quantum circuits with a quantum state biased towards potential solutions rather than starting from a neutral initial state [EMW21]. On the other hand, QNNs can benefit from pre-trained models through transfer learning, i.e., adapting a classical or hybrid model trained for a general task and training it further to tackle a similar or more precise subtask [MBI+20].

As these warm-starting methods comprise a source algorithm from which information is drawn and a target algorithm that is enhanced with it, it indicates potential ways of distributing both classical and quantum computational efforts in the continuum.

Depth and Widthwise A(Q)NN Partitioning In classical computing, depth or widthwise partitioning refers to whether layers are split horizontally and vertically.

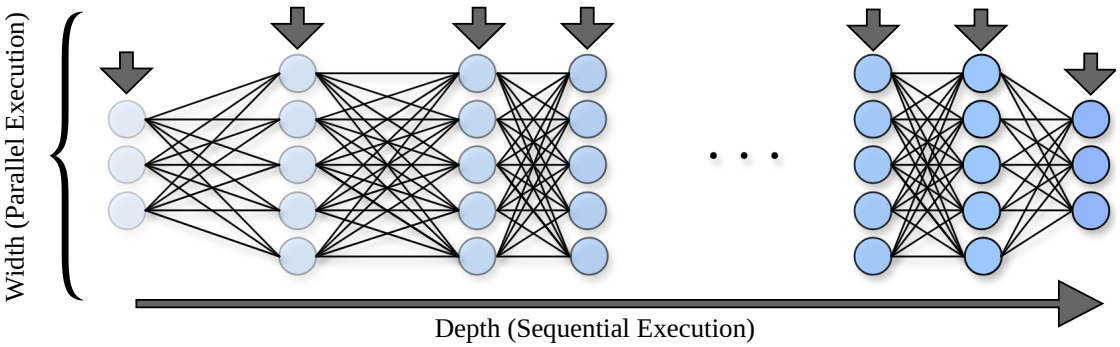


Figure 2.1: ANN Processing Flow

Figure 2.1 illustrates the width and depthwise execution of classical ANNs. Methods that split horizontally typically aim to facilitate resource efficiency by distributing layers across several devices. It increases resource efficiency, but does not lend itself to. Splitting layers vertically implies distributing cut-out chunks of a neural network, for example, across

channels in a block of convolutional layers. It can overcome the memory limitations of mobile devices. Moreover, it may alleviate batch-width frictions, which Chapter 3 will discuss.

In quantum computing, depthwise partitioning may refer to stacking QNN circuits to mitigate cascading gate errors and accumulating noise during training [BF20]. Conversely, widthwise partitioning facilitates parallelization by horizontally splitting a layer or circuit. Quantum circuit cutting concerns address some limitations of NISQ devices by partitioning larger circuits into several smaller subcircuits [BBL+23]. Classical widthwise ANN partitioning is less common since accelerators parallelize the execution of layers. Still, in highly constrained environments without access to server-grade hardware, methods such as parallelizing filter computation of convolutional layers across devices are sensible [ZBG18]. Classical depthwise and quantum widthwise partitioning are essential for inference engines in distributed platforms, and select approaches are detailed in Section 2.2.7. The following first introduces the platform architecture.

2.2 Architecture Design

2.2.1 Architecture Planes

The architecture differentiates between four planes according to their function and intended interaction with other components, clients, or client programmers. Each plane exposes private or public APIs. Private APIs are only accessible by internal components, whereas public APIs are accessible by external entities, such as client applications.

Execution Plane Application instances run on the *execution plane*. It consists of *Public APIs*, *Hardware Hosts*, and *Worker Clusters*. The public APIs allow clients to interact with the application via access points. The hardware hosts are subdivided according to the supported application types, namely into classical and quantum nodes. The quantum hosts are further subdivided according to their *grade*, i.e., QPUs are server-grade and MQPUs are mobile-grade. Figure 2.2 illustrates component organization and interaction.

The classical hosts additionally subdivide to consider ANN workloads. Each node is registered by the *device registry* that associates data describing their capacity, such as memory size, VRAM, or qubits, and stores it in the *metadata* storage. The metadata storage is a highly available key-value storage, such as ETCD[2] used in Kubernetes, accessible by other components. Especially for the continuum, where nodes can arbitrarily join and leave the system, the metadata must be highly available and not remain stale to measure the system's overall capacity accurately. The *Workers* consist of at least one hardware host and represent the application environments that may rely on one or multiple hardware nodes. Since quantum and classical environments are separate, it is necessary to distinguish between classical and quantum worker clusters. Quantum

[2] `https://etcd.io/`

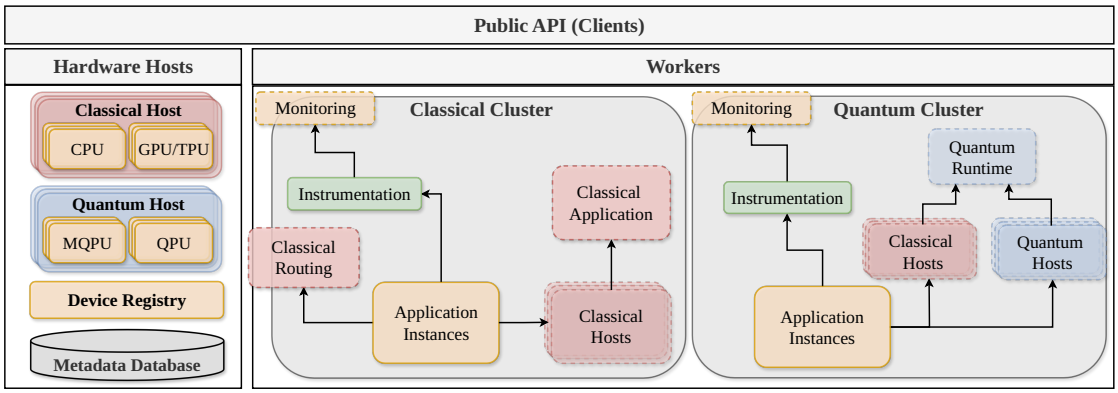


Figure 2.2: Execution Plane

applications require classical workers for auxiliary tasks, such as pre-processing and measurement. Once a worker completes a task, the classical part is responsible for forwarding the result. Intermediate results may be forwarded to another worker cluster when tasks are partitioned. The final result is forwarded to the client. Telemetry tools send data to the monitoring system by accessing private APIs of the Provenance Plane.

Provenance Plane The *Provenance Plane* encapsulates a highly available distributed Provenance database through which real-time monitoring data is stored and shared. It consists of a *Quantum Provenance* and a *Classical Provenance* system, offering private APIs to access and store data. It is crucial to design methods that consider the intrinsic properties of QPUs to create reliable and predictable systems for quantum workloads. Error rates vary depending on a QPU's current state, so exclusively collecting classically relevant data, such as load, for quantum and hybrid applications is insufficient to ensure SLOs with solution quality targets are fulfillable. Classical monitoring for a platform deploys telemetry tools (Execution Plane) alongside the application to collect data on workload trends and resource usage of function instances. Quantum provenance system gathers information orthogonally, on the state of QPUs to analyze errors, such as properties of Quantum Circuits, QPUs, Compilation, and Execution, as proposed by Weder et al. [WBL+21].

To separate concerns, we suggest the software design reflect the orthogonal handling of information as shown in Figure 2.3. This permits flexible adjustment of the granularity of information according to workload properties. A platform can utilize provenance data for error mitigation and to aid schedulers with upholding SLOs by assessing the currently expected solution quality. Platforms that support hybrid applications should integrate quantum provenance with classical telemetry to form one cohesive monitoring system for simplified access to various heterogeneous devices. The objective of a monitoring system is to collect the minimal data necessary for informing schedulers to uphold SLOs. Conceiving hybrid systems is non-trivial, as finding a balance is already challenging for classical edge-cloud and hybrid systems monitoring [GST+18, RRD+22]. A system that

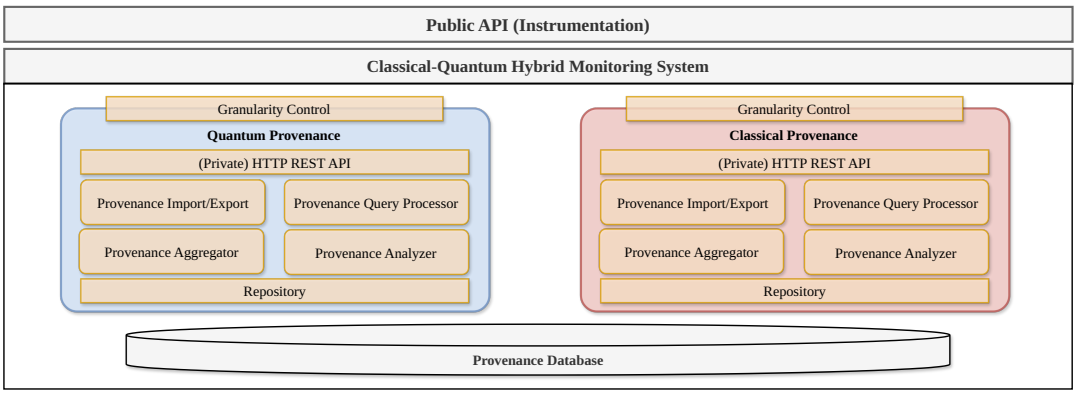


Figure 2.3: Provenance Plane

trivializes monitoring by aggressively collecting data may ease the task of a scheduler but can cause resource congestion across the edge-cloud continuum [GST+18]. Conversely, collecting insufficient data will considerably reduce the system's scalability as it cannot appropriately route requests or scale resources up and down. We argue that a *granularity mechanism* capable of adjusting the frequency of quantum and classical monitoring data according to the current workload's characteristics is one of the principal challenges that future work must address before a hybrid platform can emerge. Nevertheless, monitoring itself is simply a precondition. The following describes how our architecture supports resource efficiency and elasticity based on available monitoring data.

The importance of monitoring system states has been a key motivation for Chapter 6. The method it introduces does not directly concern quantum or classical provenance, as the diverse input properties require algorithms with no assumptions on distribution properties or value ranges.

Elasticity Plane The *elasticity plane* is the central organ of the decision mechanisms that allocate resources and route requests according to client SLOs, metadata, and monitoring data.

Figure 2.4 illustrates component organization and interaction. The *Control clusters* are subdivided between Quantum and Classical Control Clusters. The former manages quantum application instances, such as scale-out quantum coordinators, while the latter manages classical application instances, such as horizontal scaling of applications. Each control cluster manages a set of worker clusters (see Execution Plane) that are dynamically scaled up or down according to the application instances the cluster can control. Monitoring Agents are responsible for the worker clusters and relay data to the control cluster's monitoring broker. A *Monitoring Broker* disseminates the data across the components. A *Classical Autoscaler* and *Classical Scheduler* maintain a local and global view of the control cluster's state. Classical autoscalers and schedulers exist in Quantum Control Clusters, since Quantum Coordinator Applications (See Control Plane) are classic.

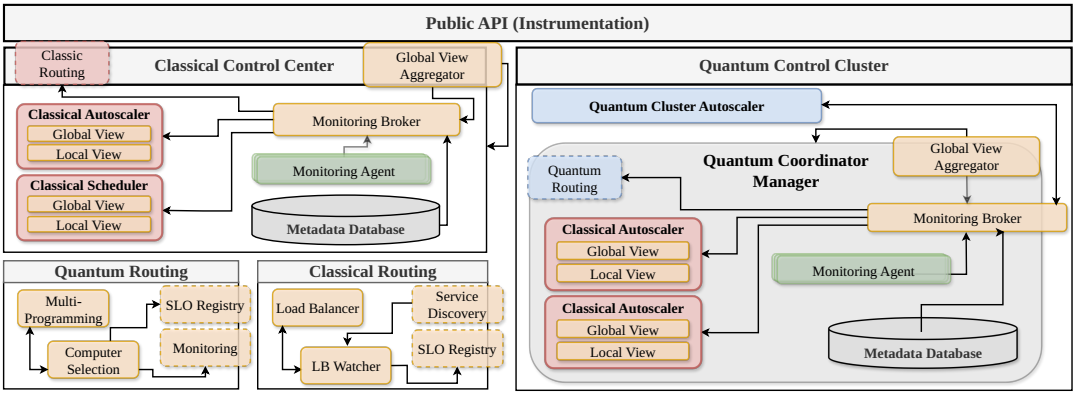


Figure 2.4: Elasticity Plane

The *Quantum Coordinator Manager* supervises the Classical Autoscaler and Scheduler to scale and place the Quantum Coordinator Application (see Control Plane) instances. The *Local View* contains fine-grained information about the Worker Clusters, such as CPU usage per second. The *Global View* contains coarse-grained aggregate information, such as average CPU usage over an hour, about neighboring control clusters. The coarse-grained data consists of fine-grained local data collected by a Global View Aggregator that periodically publishes a summary, i.e., the global view consists of exchanged summaries of local views. The control cluster's messaging topology and broker partially address the granularity control of monitoring data and permit an elastic control mechanism that can scale the entire system by adequately allocating its limited resources. The implementation of the autoscaler and scheduler is interchangeable, and system designers may experiment with various methods. The *Quantum Cluster Autoscaler* is inspired by the work of Tamiru et al. and Gandhi et al. and is an SLO-aware cluster autoscaler capable of adding and removing Quantum Hosts from worker clusters to process the incoming workload. *Classical Routing* is inspired by the work by Raith et al. [RRD+22] and consists of a *Load Balancer* and a *Load Balancer (LB) Watcher*. The Load Balancer is a high-throughput and low-overhead component that redirects incoming requests to application instances or other clusters (e.g., because no application instance is running). The Load Balancer Watcher is SLO-aware and periodically refreshes the load balancer's state to update the decision mechanism. The *Quantum Routing* component differentiates itself from Classical Routing, as it considers additional challenges to improve the resource efficiency of quantum hosts. *Quantum Computing Selection* is particularly valuable for the edge-cloud continuum as it introduces further heterogeneity. The selection method is another freely interchangeable component. For example, system designers may opt to use the method proposed by Quetschlich et al. [QBW23] and replace it once they find a more suitable alternative. *Multi-programming* in quantum computing has a comparable role to *virtualization* in classical computing, i.e., it allows sharing of the resources of quantum computers among multiple circuits [DTNQ19]. However, unlike in classical computing, where we can readily select existing mature virtualization methods, multi-programming

is more involved and should be considered together during encoding [OSVM22] and influences quantum computer selection.

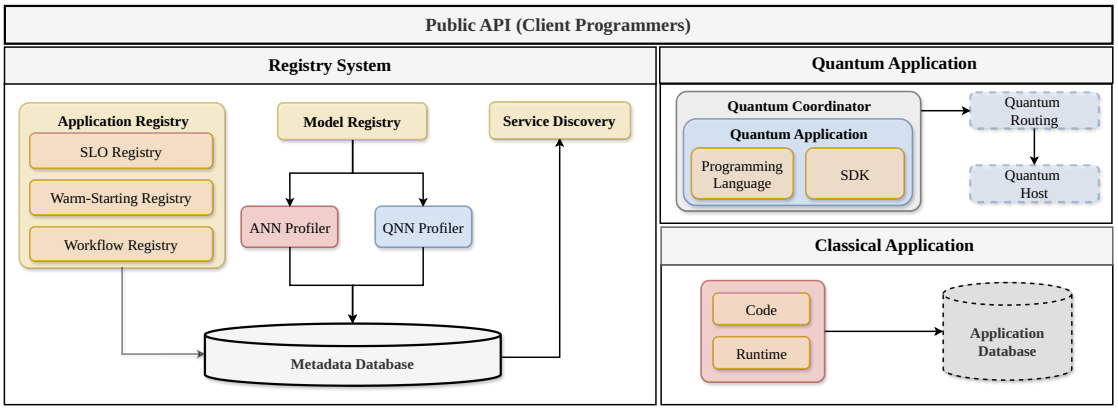


Figure 2.5: Control Plane

Control Plane The Control plane exposes an API to client programmers to deploy and manage their applications. The public API should allow client programmers to define hybrid applications as workflows without separately deploying classical and quantum parts. The platform analyzes the workflow during the registry and partitions it into classical and quantum applications. The *Application Registry* encapsulates several registries responsible for storing application services, SLO targets, and parameterized models. *The Model Registry* is a repository of readily available models. The models may either be classical neural networks or quantum circuits. In addition, profilers associate static metadata with each model, such as the number of parameters, circuit depth, and layer types. Chapter 3 will emphasize the importance of extracting the correct metadata. Profiler metadata supplements the schedulers and autoscalers with valuable information to predict resource usage more accurately. Service Discovery enables Quantum and Classical Routing to locate running application instances. The architecture supports warm-starting as a first-class citizen, which we discuss in the following.

2.2.2 Warm-Starting at the Edge

Warm-starting aligns with the objectives of a distributed hybrid platform, i.e., it facilitates drawing from resources across the continuum. Ideally, applications can pre-process input before passing it to a remote server. Warm-starting methods in the context of quantum computing are categorizable into Classical-To-Quantum (C2Q), Quantum-To-Quantum (Q2Q), or Quantum-To-Classical (Q2C) [TBB+24]. Each category has an input and an output format. For example, C2Q expects classical input, and the output format should suit a quantum algorithm.

Nevertheless, we argue that system designers must subdivide the categories further to include neural input and output formats, such as Neural-To-Quantum (N2Q) or Classic-

To-Neural (C2N), for two reasons. First, neural methods rely on AI accelerators that may not be present or have alleviated energy consumption, i.e., it is indispensable for a scheduler to know hardware properties to hit SLO targets. Second, although the output of a classical ANN is classical, the network weights may be tuned to extract features tailored for a particular class of algorithms.

2.2.3 Current and Future Role of Warm-Starting

Currently, a common motivation for warm-starting is to reduce the dependency on QPU time due to cost and limited availability [TBB+24]. However, we stress that the importance of warm starting lies in improving the solution quality by combining classical and quantum algorithms. Moreover, once QPUs mature to a point where we can entirely forgo classical computation, the research focus can shift to Q2Q warm-starting, where smaller client devices can partially onload quantum algorithms for resource efficiency. For example, warm-starting can be a means to embed performance guarantees of classical algorithms into quantum algorithms and can reduce the amount of training data required for (Q)ML.

A downside of warm-starting is that it increases the applications' complexity as it introduces more parts that must be managed. It is crucial to introduce a convenient interface that supports warm-starting as a first-class citizen to further shift complexity from client programmers to platform providers. Notably, warm-starting is chainable. Hence, we can naturally integrate warm-starting methods in the edge-cloud continuum if a platform exposes an interface that resembles the hierarchical properties regarding device capacities in the network.

The following describes the requirements and proposed solution approaches for a hierarchical warm-starting programming model. Hierarchical refers to how the warm-starting methods are composable in a *pipeline* that resembles their resource usage requirements. The proposed interface does not rely on any assumption regarding the availability and limitations of QPUs, i.e., it treats each method in the pipeline as exchangeable building blocks. The current progress of available QPUs is considered by informing client programmers about the feasibility of their planned warm-starting pipeline. For example, the platform could disable support for Q2Q warm-starting at the network's edge until MQPUs find widespread adoption in end devices, such as smartphones.

2.2.4 Hierarchical Warm-Starting

In our running example, the MAR platform aims to improve resource efficiency with hierarchical warm-starting. The load heavily fluctuates for city-scale applications according to date and time. By chaining warm-starting hierarchically, idle computational resources of edge and fog nodes can be utilized, e.g., to reduce offloading to the cloud.

Warm-starting is a broad term encompassing numerous classes of methods [TBB+24]. The challenge is to conceive an interface flexible enough to remain convenient without exposing low-level details, such as manually selecting devices and fallback mechanisms,

to the clients. An interface would be maximally flexible if it forces client programmers to define every single step of the execution, where to deploy which part at which node, and to manually configure the quantum executions (e.g., device, compiler) for every method in the pipeline. An interface that does not restrict the configuration space is especially undesirable when considering the heterogeneity of the continuum. Specifically, it would not be sufficient to provide a single configuration for a method, and there is no guarantee for the availability of a particular device configuration at the edge or fog. Conversely, constraining client programmers exclusively to a list of pre-implemented solutions is counterproductive as it hinders innovation, i.e., they should at least be able to (optionally) provide their warm-starting method.

To summarize, the responsibility of a platform is to build the infrastructure and provide adequate abstractions to access the resources. A dedicated interface for warm-starting should allow clients to define composable workflows to process a warm-starting pipeline, i.e., client programmers can register pre-processing steps for warm-starts through the control plane from our architecture that may run on CPUs, TPUs, or QPUs deployed at the client device or fog nodes.

Consider Figure 2.6 for the following example. Three clients execute the same application using the public API of the control plane. The client programmers defined a warm-starting pipeline for their applications. Hence, the pipeline and its methods are placed in the warm-starting registry, and the system decides where to position the models in the continuum based on the profiler metadata. However, the clients request varying target qualities; hence, the platform applies different intensities of warm-starting before sending the task to a quantum cloud vendor. Intensity refers to the expected solution quality of a quantum algorithm with an input processed by a warm-starting method.

In Figure 2.6, client C1 cannot achieve any pre-processing at the edge due to energy constraints (indicated by an empty battery symbol) and therefore forwards the input in its original classical representation. Since C1 registered at least one pre-processing step at the fog, the load balancer routes the request to a fog node, applying the step that maps to a C2Q warm-start. Conversely, if C1 had the resources to pre-process its input for a target neural network, the fog node would have taken over processing its output further to warm-start the quantum task, resulting in a chain of C2N and N2Q warm-start. Client C2 can perform resource-conscious pre-processing for a C2N warm-start. Still, since it does not achieve the required target quality, the request is routed to a fog node with available QPUs to apply further pre-processing for an N2Q warm-start. Client C3's request is directly routed to the cloud by the load balancer, since C3 had enough onboard resources for pre-processing the task to the target quality without relying on fog nodes.

2.2.5 Distributed Inference Engines

While the last two sections introduced high-level concepts of the architecture, this section focuses on lower-level system designs for platform designers and how client programmers may implement a distributed hybrid application. We extend our running example and

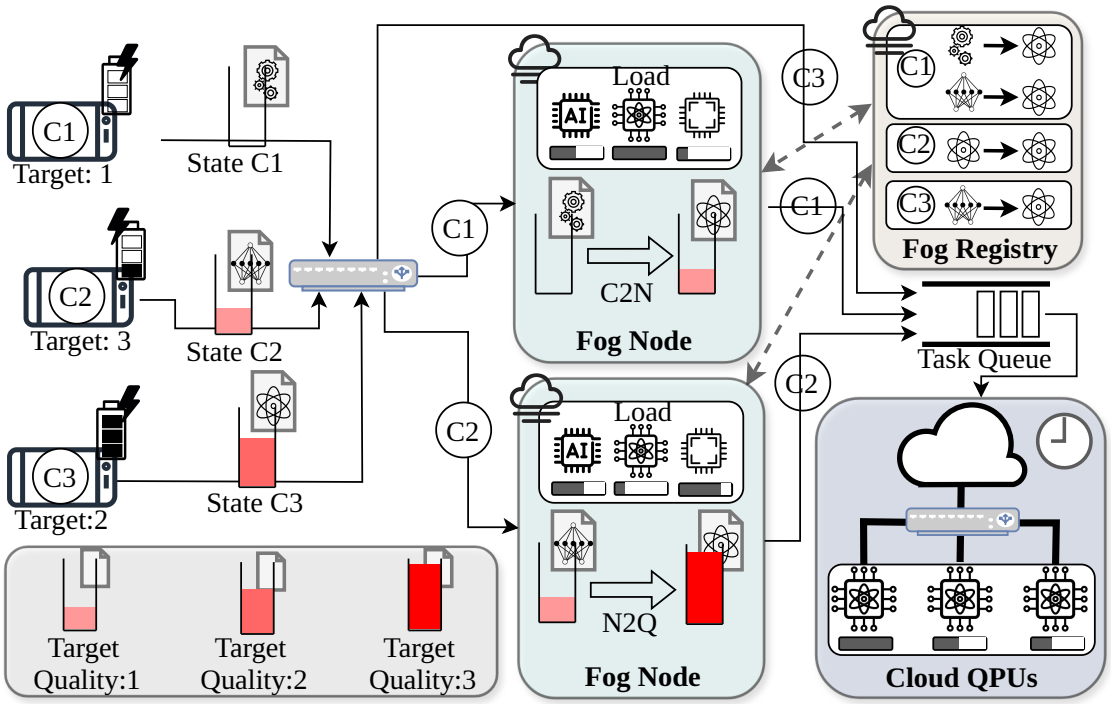


Figure 2.6: High-level Sequence of Hierarchical Warm-Starting

discuss how the MAR platform may support an inference engine with hybrid classical-quantum neural networks. An inference engine refers to the inner system of an interface that client programmers can utilize for intelligent tasks. (A)QNNs are uniquely qualified as a representative application beyond ANN inference to cover how a distributed platform can adapt as the limitations of QPUs are gradually lifted for three reasons. First, a hybrid QNN is composed of parameterized classical and quantum nodes. Contrastingly, other VQAs typically operate non-parameterized classical components exclusively for pre-/post-processing and to optimize the parameters. For a hybrid QNN, there are additional components with no statically predefined role assignments, so we can represent how systems adapt as they progressively replace classical with quantum nodes proportional to availability and advancements in QPUs. Second, we can split a neural network horizontally by layer or cut individual layers vertically and view each partition as an isolated computational graph. Then, we can emulate the behavior of a complex task with numerous classical and quantum subtasks that a system must coordinate to compute a single solution. Third, ANN partitioning and collaborative inference are well-established research areas in CEC that consider the heterogeneity of classical hardware and the resource asymmetry between edge, fog, and cloud nodes [MLR22]. Hence, we can directly extend existing work to determine whether QPUs may improve the quality of classical methods and are not restricted to orthogonal problems infeasible for classical computers.

Chapter 4 and Chapter 5 will introduce methods suitable for distributed inference engines,

but only for classical computing. An extensive study on hybrid methods is not within the scope of the thesis. However, in Chapter 7, we present preliminary results for future work that extend the method in Chapter 4 to warm start Hybrid QNNs.

2.2.6 Implementation Challenges

Unlike regular business logic, applications often draw from specialized hardware for intelligent tasks, such as visual recognition or complex optimization problems. Irrespective of whether QPUs are included, the platform must treat inference requests as a workload with distinct characteristics.

Volatility The scheduler must dynamically adapt to two sources of volatility. First, outages in the fog domain are frequent. Moreover, unlike in the cloud, classical fog resources cannot seamlessly scale horizontally, i.e., requests may have to be routed to the cloud. Second, fog and cloud QPUs may be scarce, and depending on their current state, they may not hit the target solution quality SLO.

Device Heterogeneity While the challenges of heterogeneity of classical components are only tangentially related to the integration of QPUs, minimal consideration regarding the numerous accelerators is necessary. Compilers map classical ANN to computational graphs, and vendors have varying support for operations, limiting the available layer types and activation functions [LL20].

Task Chaining and Bandwidth Consumption To fully draw from the resources on the continuum, we require methods that onload some computation on client-side accelerators. However, mobile devices can typically only host a single network in memory, and swapping out ANN weights from storage incurs significant overhead. Hence, latency-sensitive applications sending subsequent inference requests for different tasks must offload, leaving valuable resources idle. Additionally, when numerous clients compete for limited bandwidth by streaming high-dimensional image data, the limited bandwidth will inevitably lead to erratic response delays.

Optional Quantum Embeddings Although the availability of QPUs is steadily increasing, clients cannot currently expect the same graceful scaling of classical resources in the cloud. Depending on the load, hitting latency SLOs with quantum layers may not be possible. Accordingly, the inference engine should be flexible enough to skip computing quantum embeddings for near- and intermediate-term devices.

Utilizing Mobile Quantum Devices Diamond quantum accelerators are expected to be mature enough soon for commercial use [Gmb, Ltd23]. However, regardless of how MQPUs improve, analogous to classical hardware, we assume that server-grade hardware will consistently outperform its mobile counterparts. The challenge is to conceive methods that can leverage the advantages of MQPUs, ideally without sacrificing solution quality.

2.2.7 Methods for Hybrid Distributed Inference Engines

The simple solution for accommodating multiple tasks and configurable solution qualities considers a separate ANN for each variation. This is difficult to maintain and inflexible, forcing client programmers to implement and redeploy entire architectures for each task. Instead, we describe how partitioning methods lead to composable architectures that naturally define small deployable applications.

Classical Depthwise Partitioning Sequentially applying horizontally split layers is a particular form of hierarchical warm-starting. Platforms can include pre-trained ANNs in the warm-starting registry of the Control Plane. Additionally, client programmers may register modules according to their requirements. For example, edge devices can optionally perform preliminary feature compression, and fog nodes can apply a small- or medium-sized feature extractor according to solution quality and latency targets. We require an encoder suitable for constrained end devices composed of operations widely supported by the various vendors of AI accelerators. The encoder should perform initial feature extraction and find a minimal representation for a sufficient statistic on several downstream tasks to reduce bandwidth consumption. Then, the server can select an interchangeable ANN for additional feature extraction according to the configured latency and accuracy SLO. To handle the limited availability of QPUs, the QNN should be optional.

Quantum Circuit Cutting Quantum circuit cutting addresses challenges. The idea is to cut large circuits that require many qubits widthwise into smaller subcircuits requiring fewer qubits [BBL+23] by strategically cutting circuit wires [PHOW19, BPS23] and gates [MF21, BBL+23]. Figure 2.7 illustrates an example in which one wire and two gates are cut.

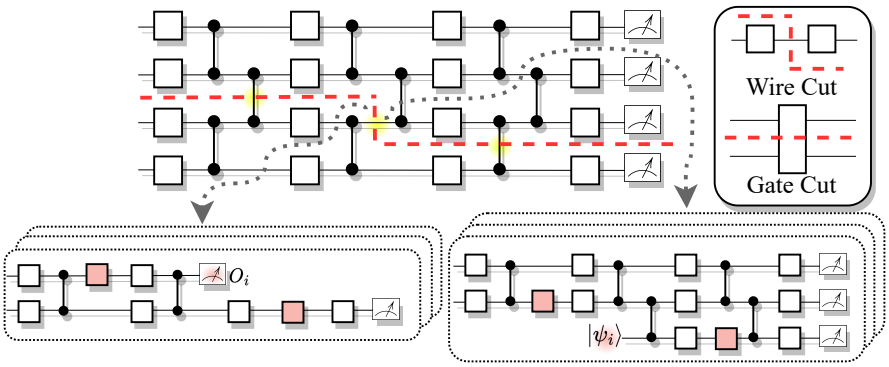


Figure 2.7: Circuit Cutting Basics

Wire cutting separates circuit wires through multiple measurements with different observables O_i and subsequent initializations of the qubit to state $|\psi_i\rangle$, while gate cutting

substitutes two-qubit gates with varying combinations of single-qubit gates. The stacked subcircuits in Figure 2.7 indicate the generation of multiple variations for each subcircuit.

The widthwise partitioning enables each subcircuit instance to be executed individually on smaller quantum devices, which may be more readily available. Following the execution of these subcircuits, a classical post-processing procedure is applied to recombine the results obtained from the individual subcircuits, ultimately reconstructing the output of the original circuit as a linear combination of the subcircuit results. This approach facilitates the distribution of quantum circuit computations across multiple QPUs without necessitating quantum communication. As a result, quantum circuit cutting offers the opportunity to harness the power of several smaller MQPUs at the edge, enabling the computation of larger quantum circuits. Moreover, it promotes the more flexible placement of quantum circuits across resources of the compute continuum. Additionally, once MQPUs are widely available, we can leverage them to parallelize the execution of subcircuits. This parallelization can alleviate the overhead associated with each cut, significantly improving computational efficiency and scalability.

Inference Flow Figure 2.8 illustrates the flow for a conditionally depthwise ANN and widthwise partitioned QNN inference across a hierarchical network. End-user devices host varying client applications. For example, one client classifies artwork in a museum to retrieve a description using a virtual tour guide. Another client may be interested in retrieving descriptions of the local fauna. Clients with AI accelerators apply neural

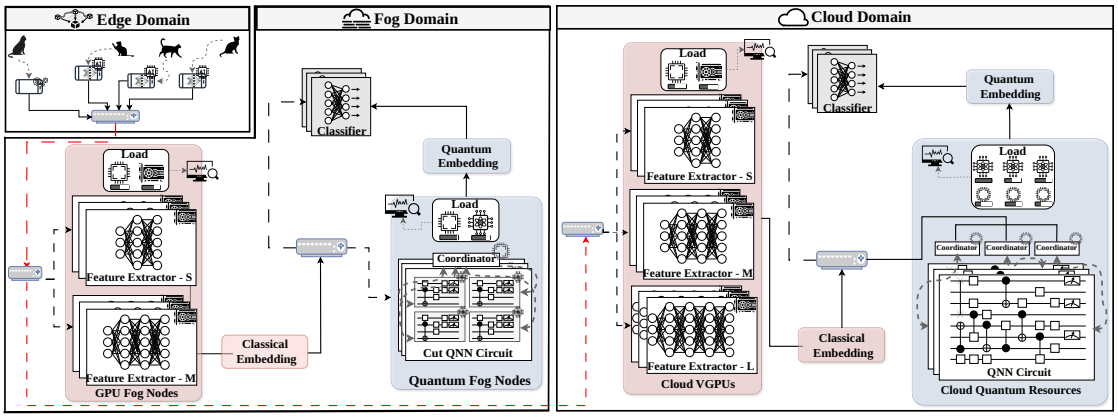


Figure 2.8: Inference Engine Request Flow

compression methods for preliminary feature extraction. Solid black lines represent the flow of a request within a domain. The dashed red lines represent inter-domain data transfers. The dashed gray lines represent the provenance data collected adjacent to inference requests by a runtime spanning all domains. The runtime is detailed by the Elasticity Plane of our architecture in Figure 2.4. It collects data to periodically update the SLO-aware load balancer to perform informed decisions based on the state of each participating node and client configurations. Dashed black lines represent an

alternative path, i.e., one request flows only to one of the choices. An edge load balancer routes the request to the load balancer of a fog cluster. Depending on load and client requirements, the request is routed to a Fog GPU node or the cloud. After feature extraction, a load balancer decides whether the classical embedding should be passed to a QNN before classification. The QNN may be executed on a Quantum Fog Node or sent to a remote cloud provider, for example, due to privacy or availability. However, based on our assumption, the MQPUs are more constrained than the server-grade QPUs from cloud providers. Hence, circuit cutting methods can aid MQPUs in achieving a target solution quality. A request ends after the label is sent as a response to the client.

Application Preparation and Deployment The individual operations of an ANN form a computational graph. Moreover, partitioning methods naturally demarcate a monolithic ANN into connectable vertices. The vertices represent coarse-grained classical layers or QNN circuits, and one or multiple consecutive vertices form one *depthwise partitioned deployment unit*. Alternatively, we can further partition a vertex to create one *widthwise partitioned deployment unit*, for example, with circuit cutting. Notably, depthwise methods define isolated compute nodes, which we can transparently combine with widthwise methods, i.e., from an outside view, an adequate abstraction can present a cut circuit as a single coarse-grained layer. The client programmers may provide hints to the platform via annotations, but the application should be deployable as a single (monolithic) workflow. It is the responsibility of the Control Plane of our architecture to create the deployment units before spawning Quantum and Classical Application instances. From the point of view of the client programmers, they have deployed a single application. However, the runtime system should be aware that the application is split into multiple parts. Figure 2.9 illustrates an example with a computational graph of coarse-grained layers. A coarse-grained layer consists (recursively) of finer-grained layers. The nodes are enumerated to indicate the processing sequence, and a subindex indicates a branching path. A node with the same index and subindex implies the same partition deployed on different nodes. Partitions 1-2 are grouped depthwise and will be deployed as one unit on edge devices. Partition 3.2 is deployed on cloud and fog nodes, while Partition 3.1 is a different model deployed exclusively on cloud nodes. For example, Partitions 3.1 and 3.2 could be Feature Extractor L and M from Figure 2.8. An SLO-aware load balancer routes the output of Partition 2 to a variation of Partition 3. Lastly, the output of Partition 3 is passed on to one of the instances of Partition 4. Partition 4 is deployed on fog and cloud nodes. However, it is a QNN circuit that a server-grade cloud QPU can execute, but must be partitioned widthwise for the mobile-grade MQPUs at the fog nodes.

2.3 Summary

This chapter presented a distributed hybrid platform integrating quantum and classical resources in a hierarchically organized network. It summarized existing literature in quantum and classical computing. The presented applications focused on facilitating

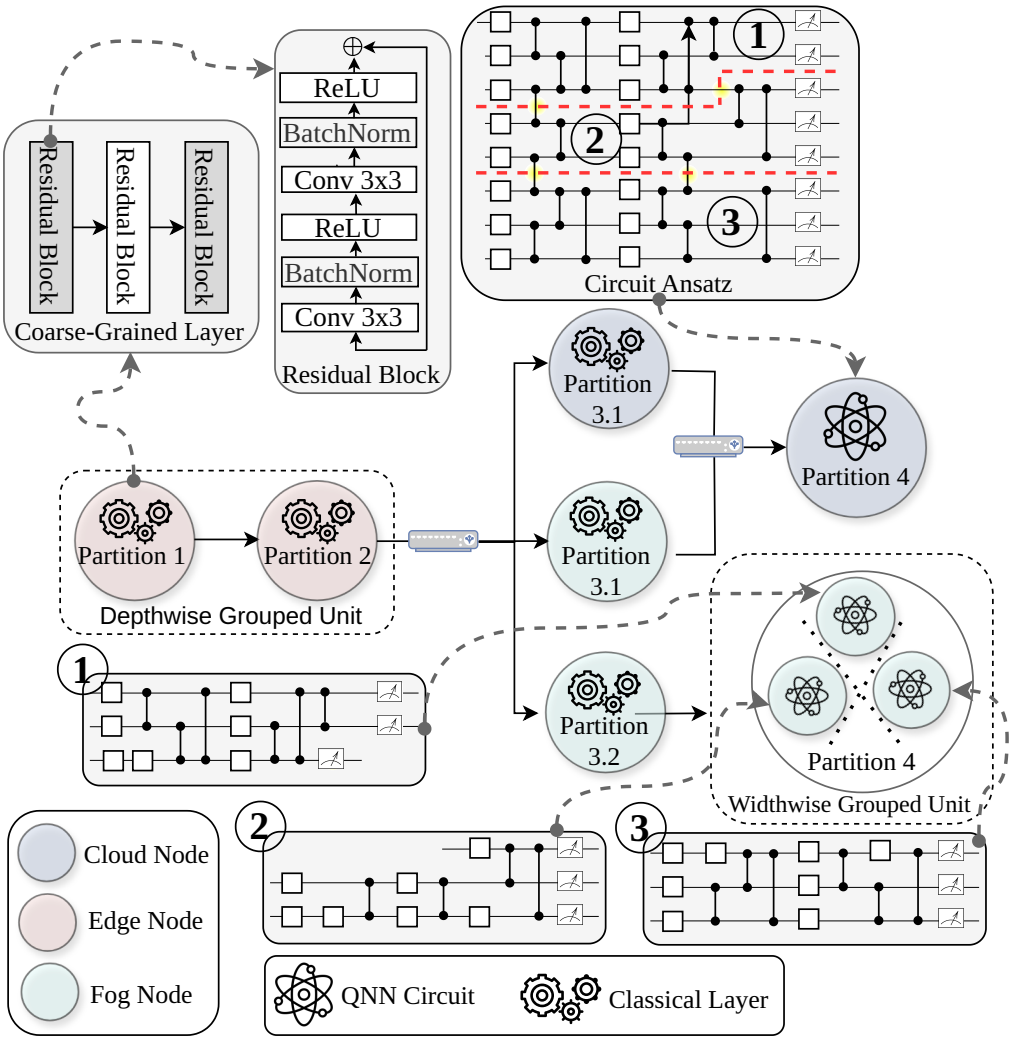


Figure 2.9: Coarse Grained Deployment Units

research efforts in quantum applications with warm-starting and ANN inference. The key insight is the importance of seamless integration of components and that scaling such platforms relies on large volumes of information at varying degrees of granularity.

CHAPTER 3

Design Considerations for Neural Networks in Edge-Cloud Systems

The chapter was motivated by observing substantial differences in performance between testbeds and real-world experimentations, particularly when applying *readily* available tools for optimizing performance. While absolute performance differences do not invalidate results, *diverging relative performance rankings* across competing approaches invalidate claims that authors deduce based on empirical data. We show that vendor-specific software optimization stacks are a key source of unexpected performance reordering. The chapter introduces a methodology and software tool to increase the confidence and reproducibility of empirical studies in edge–cloud systems.

3.1 Graph Compilers for Artifical Neural Networks

The pervasiveness of ANNs in modern computing systems has generated significant demand for methods to improve the efficiency of available hardware. As computational complexity increases and deployment scenarios diversify, optimizing ANN execution becomes indispensable for practical applications across various computational platforms [RHS+21b]. Among the most promising optimization approaches are graph compilers, which optimize the computational graphs of neural networks to enhance scheduling, improve data flow, and exploit dedicated hardware modules. Unlike other model optimization methods, such as model compression [DLH+20] with pruning [CZS24], quantization [LHC+24], or knowledge distillation [GYMT21], graph compilers can improve throughput or latency by orders of magnitude with no loss in accuracy.

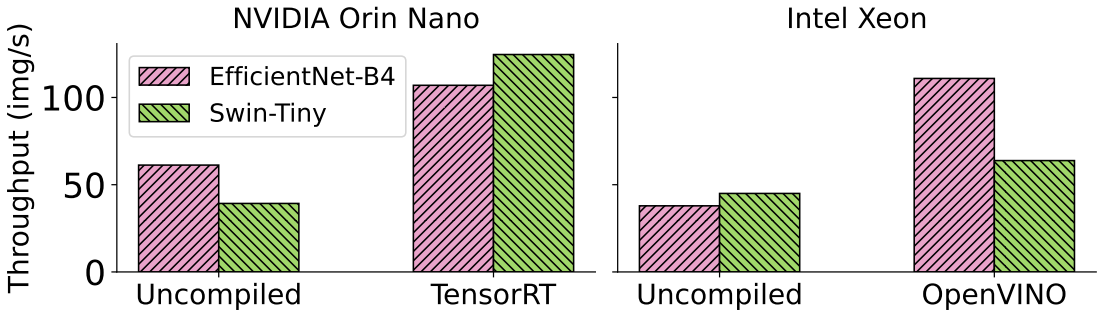


Figure 3.1: Graph Compilers Reversing Throughput Rankings.

3.1.1 Trusting Research Results for Real-World Operations

While these compilers can be used independently, they may also be combined with model compression or acceleration methods, such as quantization, that trade off efficiency for accuracy. The potential performance improvements are substantial. Yet, fully leveraging graph compilers presents distinct challenges. Graph compilers and other vendor-specific optimizations can completely alter the relative performance across competing architectures. Figure 3.1 not only precisely exemplifies this behavior, but also demonstrates that the exact inverse holds for a different device-compiler pair.

The models are comparably large, and the batch size is 8. On the Orin, the convolutional-based EfficientNet has a higher throughput than the transformer-based Swin. Applying TensorRT significantly improves throughput for both models. However, the EfficientNet is now slower than the Swin. On the Xeon with OpenVINO, we observe the exact inverse behavior. Swin is faster before compilation, and compilation yields throughput gains for both models, but Swin is now slower than EfficientNet. Section 3.2.3 will detail experiment configurations.

Arguably, the increasing complexity of optimizing neural networks creates a disconnect between academic research and real-world applicability, despite directly addressing practical problems. When designing novel machine learning algorithms for Edge(-Cloud) Systems [DZF+20], it is crucial to understand how graph compilers can invalidate relative performance differences between architectural archetypes. A common problem when extending research work into real-world systems is determining whether reported performance improvements, regarding resource usage or throughput, from the latest advancements will generalize to the target hardware. This problem stems not from a lack of rigor by researchers but from the inherent heterogeneity of the AI accelerator landscape [RMJ+22]. This insight was a key motivation in our previous works [FRD24, FZR+25, FFSD25], where we deliberately opted for simplified encoder architectures with widely supported operations to ensure that reported results would generalize across vendors. While these and similar research contributions are valuable, their practical application requires further consideration, often creating a needlessly high barrier for practitioners by having to navigate complex optimization landscapes. To narrow the gap

between research contributions and their applications in real systems, we introduce an automated tool that integrates with existing profilers commonly used in edge or cloud frameworks (e.g., for model selection [CSM+20]). The tool streamlines graph compiler benchmarking over heterogeneous compute infrastructure to facilitate iterative empirical analysis.

3.1.2 Background

Graph compilers analyze and optimize computational graphs representing neural networks as nodes (operations) and edges (data dependencies). They provide abstraction to lower-level implementation details by converting models from high-level frameworks (PyTorch, TensorFlow) into hardware-agnostic intermediate representations. Moreover, they may apply transformations that improve execution speed and memory efficiency across AI accelerators, and hardware-specific code generation for low-level kernels tailored to target architectures, such as CUDA for NVIDIA GPUs, and OpenCL for FPGAs.

High-level Network Architecture Organization

Figure 3.2 illustrates how most modern architectures organize layers.

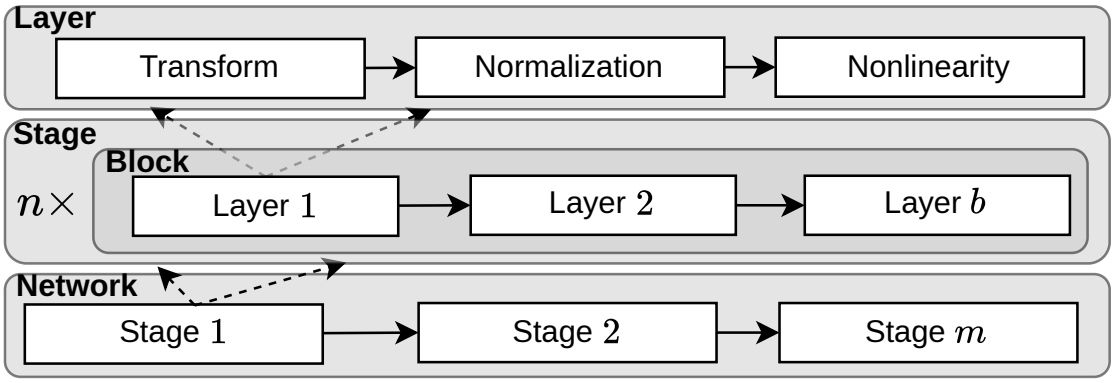


Figure 3.2: Network Architecture Layer Organization

Each layer applies a linear transform, normalization, and introduces non-linearity with an activation function. Layers are grouped into blocks, which may be more complex, as shown here, such as the ResNet bottleneck [HZRS16] that uses two 1×1 convolutional layers to reduce the number of channels, before increasing them again. A stage consists of a sequence of repeated blocks. Finally, an architecture consists of at least one stage, and each stage may have a variable number of blocks. The difference between models of different sizes from the same architecture is typically their width and block ratios. For example, the block ratio in Swin-Tiny is 2:2:**6**:2 and in Swin-Base 2:2:**18**:2 [LLC+21]. Graph Compilers can improve throughput by exploiting the repeated patterns present in such organizations and by providing specialized hardware modules for particular

compositions (e.g., `Conv-BatchNorm-ReLU`). The following briefly summarizes software and hardware optimizations that compilers commonly use.

3.1.3 Software-Level Optimizations

Operator Fusion Operator fusion combines multiple operations, such as convolution and activation, into a single computational kernel. Without operator fusion, each operation would write intermediate results to memory and then read them back for the next operation. By fusing operations, the compiler generates a single kernel that executes all the fused operations sequentially within the same execution context. This eliminates redundant memory accesses and reduces the overhead of launching multiple kernels.

Constant Folding Constant folding identifies subgraphs where all inputs are constants and precomputes them at compile time. This reduces runtime computation by eliminating the need to compute results that do not depend on dynamic inputs repeatedly.

Layout Transformation Different hardware architectures have specific data layout preferences for optimal performance. For example, NVIDIA GPUs with Tensor Cores prefer the BHWC (batch size, height, width, channels) format over BCHW (batch size, channels, height, width). Layout transformations reorganize tensor data into these preferred formats during compilation. These transformations ensure that memory accesses are coalesced and aligned with hardware requirements, improving throughput.

Hardware and Kernel-Level Optimizations

Kernel Fusion Kernel fusion is similar to operator fusion, but at the kernel level. Similarly to operator fusion, kernel fusion combines multiple operations into one kernel execution to reduce kernel launch overheads. However, kernel fusion operates at a lower level and can merge operations with finer granularity.

Memory Latency Hiding Memory latency hiding overlaps computation with data transfers using asynchronous execution techniques. Specifically, by overlapping data movement (e.g., between global memory and shared memory) with computation, the memory access latencies appear instantaneous, i.e., "hidden." Similarly to dynamic batching, it involves a static code analysis and code generation for execution paths. The execution paths facilitate asynchronous memory transfers and efficient scheduling of threads, such that some threads perform computations while others wait for data transfers to complete. For example, in matrix multiplication on GPUs, while one block of threads computes partial results using data already loaded into shared memory, another block loads the next set of data from global memory asynchronously.

Sparse Computation Sparse computation exploits sparsity in weights or activations. It leverages tensor sparsity patterns (e.g., weights with many zero values) to skip

unnecessary calculations and reduce storage requirements and memory use. In particular, specialized sparse matrix formats like Compressed Sparse Row or Block Sparse Row store only non-zero elements efficiently. Hardware accelerators often include optimized sparse matrix multiplication routines that exploit these formats. For example, consider a sparse neural network where 70% of weights are zero due to unstructured pruning. Instead of performing dense matrix multiplication on all elements, sparse matrix multiplication algorithms process only non-zero elements stored in CSR format. Then, multiplying an input vector with a sparse weight matrix skips zero-weighted connections, reducing computation time and memory bandwidth usage.

3.1.4 Related Work

Shuvo et al. [SICM23] provide an excellent review on techniques for utilizing AI accelerators, but it is focused on lower-level tricks for a particular class of hardware. Zhou & Yang benchmark TensorRT [ZY22], but only on convolutional architectures. Li et al. [LLL+21] provide a broad overview of existing compilers, but only include rudimentary evaluation on behavior in practice. Like our work, Xing et al. examine graph compilers on different hardware (CPUs, GPUs) [XWW+19], but the evaluation only considers individual operations and convolutional-based architectures, without addressing important factors, such as batch size, depth, width, etc. Conversely, this work evaluates graph compilers on various networks from varying architectural families and leverages the broad results to draw generalizable insights. The work by Jajal et al. [JJT+24] shares similarities in examining computational graph optimization of varying architectural styles and vendors, but the focus is on interoperability, and specifically the issues that may be encountered when converting models to ONNX. The work in [ZJS+25] also examines computational graph optimization, but more generally focuses on uncovering bugs in the development cycle of systems that train and deploy deep neural networks. The work in [ZXWZ23] shares similarity in advocating for a design strategy that is mindful of the underlying hardware acceleration. Still, it is an entirely qualitative assessment without any empirical analysis. Lastly, Zhang et al. [ZLC+22, ZCC+24] introduce libraries for benchmarking and provide comprehensive results, but include only mobile platforms and do not examine graph compilers.

3.2 Graph Compiler-guided Method Design

We implement *NGraphBench*, a library that permits quick, automated, empirical evaluation of graph compilers in a heterogeneous cluster. However, the focus of the work is not the implementation details of the library, and we only mention high-level details for evaluation transparency in Section 3.2.3. Instead, the focus is on effectively utilizing empirical compiler benchmark results to iteratively conceive and refine ML methods, with a clear application focus on edge-cloud systems.

3.2.1 NGraphBench Library

NGraphBench exposes a uniform interface for accessing and integrating graph compiler APIs. Users can provide their models in ONNX or native PyTorch and configure experiments, such as compiler-device pairs, compiler flags, repetitions, and model initialization parameters.

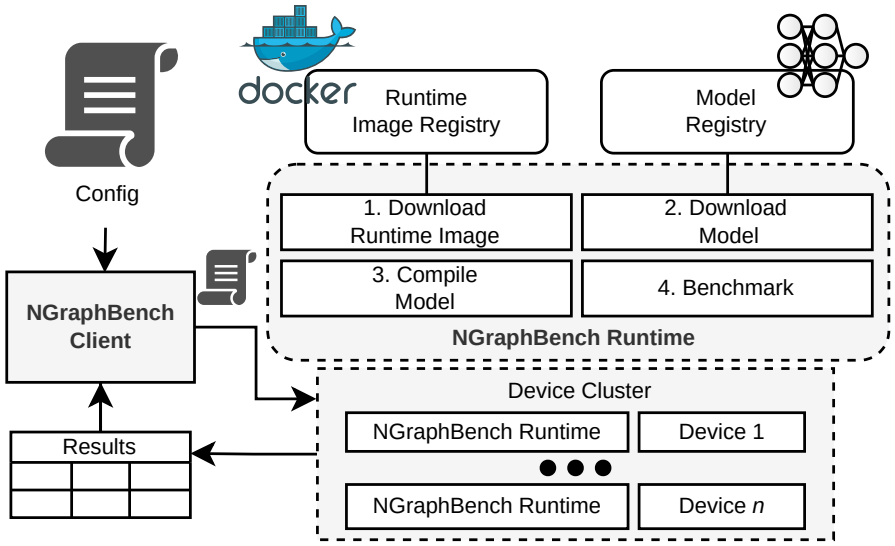


Figure 3.3: NGraphBench high-level flow.

Figure 3.3 illustrates the high-level application flow. The client coordinates the experiments for participating clients in a cluster. Each device may evaluate multiple compilers. Crucially, the compilation is done *locally* on the target devices, i.e., we are *not* using hardware simulators, which are likely to result in worse optimizations. Each device may benchmark multiple compilers, and will persist results in predefined checkpoints periodically (e.g., to resume on a crash). After benchmarking, the devices will report the results to the client. Once all devices have reported their results, the client will tear down the benchmarking environments and terminate the application.

3.2.2 Pragmatic Research Design for Practical Systems

The disconnect between academic research and practical deployment is particularly problematic when optimizing neural networks for heterogeneous hardware. While novel architectures may excel in controlled benchmarks, their performance can vary dramatically when deployed with different graph compilers across diverse hardware. We propose a methodological framework that incorporates compiler effects throughout the research process, as illustrated in Figure 3.4.

This framework divides the research process into three phases, each integrating compiler optimization considerations:

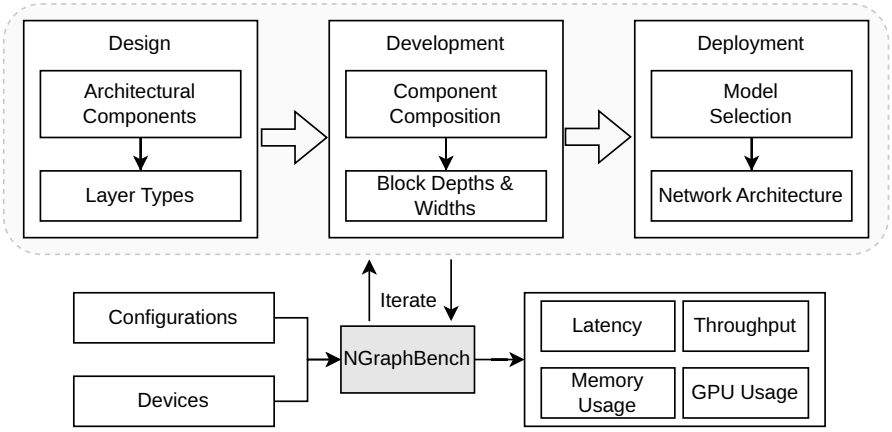


Figure 3.4: Iterative refinement guided by empirical analysis

Design Phase The design phase prioritizes architectural choices with widespread hardware and compiler support. Using our work in [FZR+25] as a case study, we selected variational compression methods for orbital edge computing applications where processing must occur within finite time windows. Rather than optimizing for theoretical metrics like the number of Multiply-And-Accumulate (MAC) operations or parameter counts, we introduced the *Transfer Cost Reduction per Second (TCR/s)* metric to balance compression efficiency against computational throughput. This approach enabled the evaluation of different architectural paradigms (convolutional vs. transformer-based) against practical deployment metrics.

Development Phase During development, researchers must examine how block compositions affect model performance and compiler-optimized throughput. Our analysis revealed that increasing model depth often yields disproportionate throughput gains when target hardware incorporates vendor-specific compiler support. Similarly, compiler optimizations can effectively mitigate width adjustments that should reduce throughput. Researchers can identify viable block compositions and configurations that maximize performance within deployment constraints through systematic evaluation with graph compilers.

Deployment Phase The final phase involves comprehensive testing on target hardware with appropriate compiler optimizations. In our case study, without graph compiler optimization, increased model width significantly deteriorated batch parallelization efficiency, contradicting development-phase expectations. This resulted in selecting marginally smaller models for constrained devices despite 30% worse compression performance. The cause was what we refer to as the *batch-width scaling friction*. By slightly increasing the convolutional channels (i.e., the width), the TCR/s has significantly dropped due to reduced processing throughput. Such counterintuitive outcomes highlight the critical importance of evaluating compiler-hardware interactions throughout the research process.

In short, the methodology aims to bridge the gap between academic innovation and practical deployment by integrating graph compiler considerations into each research phase. While this approach requires additional empirical testing, it ensures that reported performance improvements generalize across deployment scenarios, ultimately producing more valuable contributions for practitioners working with heterogeneous hardware environments.

3.2.3 Empirical Analysis

The analysis is motivated by the three phases of our compiler-guided framework from Section 3.2.2. We examine: (1) differential compiler support across architectural styles to inform design decisions in Figure 3.5; (2) depth scaling and batch-width friction mitigation effects to guide component composition in Section 3.2.4, Section 3.2.4; and (3) analyze graph compiler effects on resource usage to reason about unexpected throughput gains or losses, such as from adverse effects by concurrent tasks in Section 3.2.4. We restrict the study to vision models to control confounders and isolate compiler effects, and block-level experiments cover convolutional and attention compositions. However, we emphasize that the results generalize to problem domains that rely on ANNs for function approximation, such as NLP or Deep Reinforcement Learning. Moreover, we isolate compilation from model compression, such as quantization [LHC+24], pruning [CZS24], or knowledge distillation [GYMT21], as compilers alter execution without affecting accuracy or training. In contrast, model compression introduces accuracy trade-offs that would confound compiler-specific effects.

Experiment Design

We include TensorRT and OpenVINO to represent vendor-specific compilers and Apache TVM to represent vendor-agnostic compilers with hardware-level optimizations. The ONNX and TorchScript runtimes represent a software-level optimization approach. The evaluation exclusively focuses on applying graph compilers without fundamentally altering prediction behavior, i.e., it does *not* consider quantization and other model compression methods. For experiments with a relative measure that relies on a baseline (e.g., speedup factors, BSR), we use the PyTorch dynamic computational graph and refer to it as the *identity*. We repeat each experiment 100 times and report the average with standard deviations. Compilation is performed on the *native hardware* without hardware simulators. The experiments are performed end-to-end by deploying them on a physical testbed cluster using the NGraphBench library (Section 3.2). The following details the testbed and configurations to facilitate reproducibility. We emphasize that in this work, the NGraphCompiler library is exclusively for convenience and is not required to reproduce our results.

Testbed We implement a physical testbed with relevant specifications summarized in Table 3.1. For clarity, we will refer to Server 1 and Server 2 as “GPU” and “Xeon” respectively, i.e., the chip we compile for and run the neural network on. The Orin Nano

Table 3.1: Testbed Device Specifications

Device	CPU	GPU
Server 1	8x Ryzen 5700G @ 3.80 Ghz (x86)	RTX 4070
Server 2	8x Xeon Skylake @ 3.0 Ghz (x86)	N/A
Orin Nano	6x Cortex-A78 @ 2.0 Ghz (ARM)	Amp. 512 CC 16 TC

uses `Jetpack 6.2`, which is based on `Ubuntu 22.04`. Hence, the other devices use `Ubuntu 22.04 LTS` with Linux kernel version 5.15. We prioritize consistency over using the latest versions. Table 3.2 reports the oldest versions installed on the devices.

Table 3.2: Library Versions

Library	Version
ONNXRuntime	1.19.2
TensorRT	10.4.0
ApacheTVM	0.18.dev0
OpenVINO	2024.3.0

Library	Version
PyTorch	2.4.1
CUDA	12.5
cuDNN	9.3.0
timm	1.0.15

Compiler Configurations Except for Apache TVM, we use intuitive default configurations for graph compilers (e.g., optimize for throughput instead of latency in OpenVINO when the evaluation criterion is throughput). To remain vendor-agnostic, TVM takes a fundamentally different approach to optimization than vendor-specific compilers. TVM can fuse arbitrary patterns and support new operations, if it can find them [CMJ+18]. Vendor-specific frameworks compile fast, but are limited to pre-defined fusion patterns or operations. TVM's tuning involves running many candidate kernels on the hardware or a simulator to measure performance, yielding highly optimized code, potentially matching or exceeding vendor libraries. The caveat is that TVM traverses an exponentially

Table 3.3: Contrasting Compile Times in Seconds

Model	Batch Size	**Intel Xeon**		**GeForce RTX**	
		OpenVINO	TVM	TensorRT	TVM
ResNet-101	1	2.249	18,022.663	11.863	57,307.949
	32	4.080	28,327.363	16.089	58,466.694
EfficientNet-B5	1	3.109	36,241.423	53.332	69,364.992
	32	5.646	49,765.446	68.360	61,700.123
ConvNeXt-Base	1	4.830	36,431.073	10.004	16,533.937
	32	6.763	68,943.533	19.394	18,404.425
DeiT-Base	1	4.764	11,464.397	6.320	5230.073
	32	6.264	36,199.033	13.827	6157.337
Swin-Base	1	9.338	88,095.557	17.545	27,378.831
	32	12.628	13,4262.784	29.204	14,287.950

scaling search space, such that finding an *optimal* computational graph for a single experiment may take weeks or months. Therefore, we cap the number of trials at 1500 with early stopping after 150 using the `xgb` tuner, as we empirically determined on a subset of models that increasing the number of trials beyond 1500 yields diminishing results. Moreover, we only apply TVM to the off-the-shelf models on the *native hardware* and omit it from the block-level evaluation due to time constraints. Table 3.3 shows the compile times for the largest models. Notice that even after limiting the number of trials, the compilation time may take more than 19 hours. Lastly, notice that the compilation time increases with batch size only for Apache TVM. Unlike OpenVINO and TensorRT, the TVM search heuristic relies on real measurements for each candidate graph, where the runtime scales with the batch size.

Network Architecture & Layer Composition We perform experiments on off-the-shelf architectures and more fine-grained blocks. Evaluating widespread models yields general insights, such as whether vendors favor a particular architectural style. Table 3.4 summarizes the architecture specifications.

Table 3.4: Network Architecture Specifications

Architecture	Style	Parameters	MACs
ResNet-18	Convolutional	11,689,512	1,814,083,944
ResNet-50	Convolutional	25,557,032	4,089,238,376
ResNet-101	Convolutional	44,549,160	7,801,511,784
EfficientNet-B3	Convolutional	12,233,232	962,729,320
EfficientNet-B4	Convolutional	19,341,616	1,503,740,472
EfficientNet-B5	Convolutional	30,389,784	2,356,534,504
DeiT-Small	Transformer	22,059,496	79,557,352
DeiT-Medium	Transformer	38,849,512	115,513,320
DeiT-Base	Transformer	86,585,320	201,581,032
Swin-Tiny	Transformer	28,328,674	52,152,040
Swin-Small	Transformer	49,737,298	66,312,424
Swin-Base	Transformer	71,125,762	94,739,176
ConvNeXt-Tiny	Hybrid	28,589,128	322,371,592
ConvNeXt-Small	Hybrid	50,223,688	411,391,240
ConvNeXt-Base	Hybrid	88,591,464	646,530,408

We use the `timm` [Wig19] library that ensures consistent implementations to access off-the-shelf architectures, so exact parameter and MAC counts may differ slightly from those reported in original publications. We consider five architectural families and three consecutively increasing model sizes per family. We include two convolutional-based (ResNets [HZRS16], EfficientNets [TL19]) and two transformer-based (Swins [LLC+21], DeiTs [TCD+21]). Additionally, we include ConvNeXts [LMW+22] as a hybrid approach that is a convolutional-based model but includes design principles from transformers.

Table 3.5: Per-Block Specifications

Convolutional		**Multi-Head Attention**	
Channels	Params Per Block	Embedding Dimensions	Params Per Block
64	37,056	128	66,048
96	83,232	256	263,168
128	147,840	384	591,360
256	590,592	512	1,050,624

Table 3.5 summarizes the per-block specifications. Blocks allow us a more targeted evaluation of depth (i.e., investigate optimization as we stack repeated blocks) and width, and reduce noise from certain implementation quirks or other factors that affect compiler efficacy.

The multi-head attention (MHA) block uses ReLU nonlinearity. We use channels in convolutional blocks and embedding dimensions for MHA blocks to parameterize block widths when investigating batch-width friction. We found that varying kernel and input sizes similarly affect batch-width friction as increasing the channels. To simplify, we only report results with the kernel size fixed at 3×3 and input size $3 \times 244 \times 244$. In MHA block experiments, we fix the sequence length to ten for the input tensor and consider the embedding dimensions to parameterize the block width.

Measuring Batch Parallelization

We can measure the Relative Throughput Rate (RTR) as

$$RTR_c(b) := \frac{T_c(b)}{T_c(1)}$$

where $T_c(b)$ is the throughput in samples per second for batch size b when compiler c is used. The RTR quantifies the unnormalized parallelization rate. When scaling is perfect, the throughput linearly scales as a function of the batch size. Note that once RTR drops below 1, the batching reduces absolute throughput. Moreover, perfect scaling does not happen in practice for larger batch sizes, so we will visualize the decay in batch scaling efficiency with the Absolute Scaling Efficiency (ASE) measure:

$$ASE_c(b) := \frac{T_c(b)}{b \cdot T_c(1)}.$$

The ASE normalizes the RTR, so any reduction from a perfect parallelization rate directly indicates reduced scaling efficiency. For example, when batch parallelization is perfect, the ASE stays consistently 100%, irrespective of the batch size. Conversely, decreasing ASE implies diminishing returns from increasing the batch size. As discussed in Section 3.2.2, it is interesting to see whether compilers can mitigate the decay, i.e., maintain scaling

efficiency at higher batch sizes. We measure this with the Batch Scaling Resilience (BSR) as follows:

$$BSR_c(b) := \frac{ASE_c(b)}{ASE_{identity}(b)}.$$

The BSR is a relative measure that can quantify, without eyeballing, the improvement in mitigating friction compared to a baseline compiler. Compilers with BSR values greater than 1 consistently across varying configurations demonstrate that they can improve the batch scaling efficiency for target hardware.

We emphasize that ASE and BSR are analytical instruments that quantify compiler-induced scaling behavior; they are not conceptual innovations but make observed patterns measurable and comparable across settings. In particular, measuring BSR as we increase the batch size for different width configurations can determine whether a compiler can alleviate the scaling friction, which we elaborate on in the following.

Batch-Width Scaling Friction

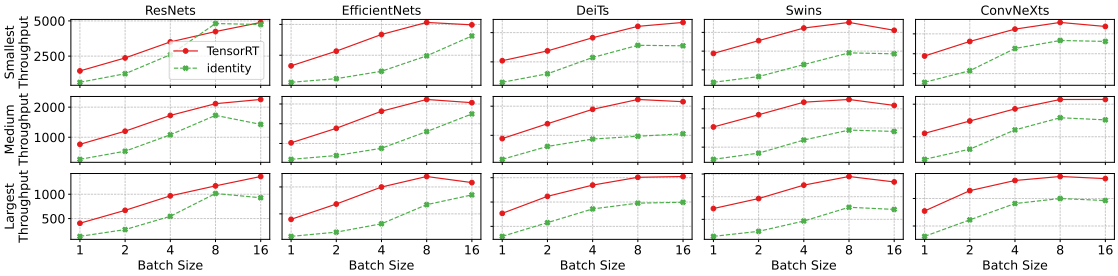
The *batch-width scaling friction* describes the joint effects of increasing model width and batch size parameters on scaling efficiency that significantly impact systems that prioritize processing throughput, such as in [FZR+25]. To empirically assess the efficacy of compilers to mitigate the effect, we provide a minimally formal definition.

Definition 1. *(Batch-Width Scaling Friction) Batch-width scaling friction is present when, for fixed b, $ASE_c(b, w_2) < ASE_c(b, w_1)$ with widths $w_2 > w_1$ in the operating range, which indicates that increasing width reduces parallelization efficiency.*

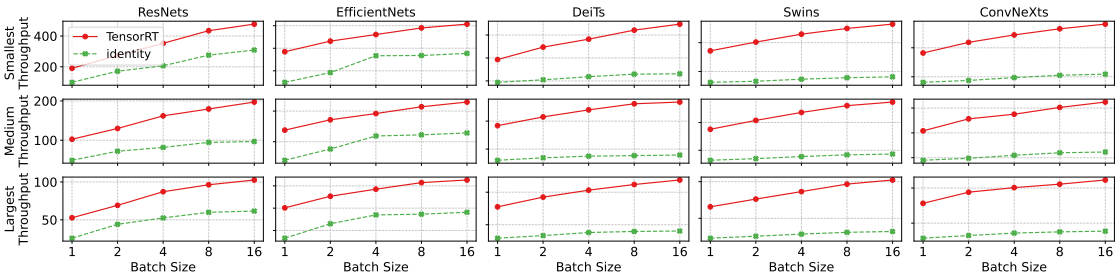
3.2.4 Compiler Support Across Architectural Styles

Figure 3.5 compares the absolute throughput of the vendor-specific compilers with the dynamic uncompiled graphs. It shows how compiling results in significant throughput gains, except when resources are scarce, the performance saturates. Each row corresponds to a model size, and each column to an architectural family (e.g., smallest for ResNets is ResNet-18). The y-axis scaling is non-uniform, highlighting the strong relationship between compiler efficacy and architectural style. Notice that even if the absolute throughput across model sizes is offset, the throughput scaling as we increase the batch size is strikingly similar within a family. The caveat is that the varying device capacities obfuscate the results, making it challenging to assess compilation efficacy. Hence, we report relative values for the remainder of the evaluation but summarize partially aggregated measurements using absolute values in Table 3.6. Figure 3.6 plots the throughput multiplier for the five architectural families on different devices-compiler pairs. It contrasts the multiplicative throughput increase relative to the uncompiled baseline.

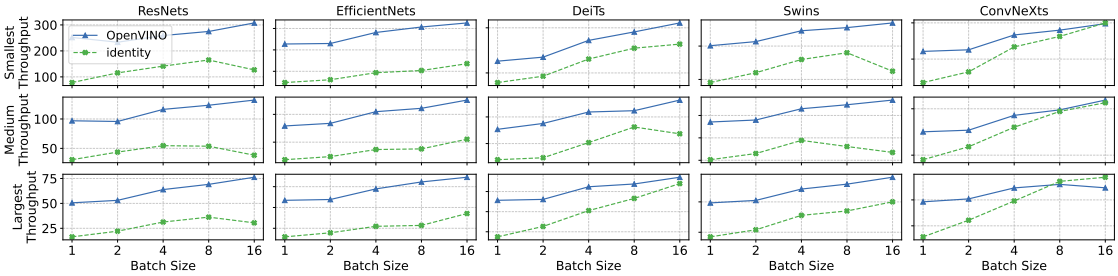
The results reveal strikingly distinct performance patterns across batch sizes and neural network architectures, demonstrating performance patterns highly dependent on batch



(a) Device = GPU



(b) Device = Orin



(c) Device = Xeon

Figure 3.5: Contrasting the absolute throughputs

size and neural architecture. Conv-based architectures see broader support, whereas for transformer-based architectures, it strongly varies. Interestingly, despite being advertised as a "convolutional architecture", ConvNeXt behaves similarly to transformers in performance across all compilers. At small batches ($\leq$2), compilers provide 2-6$\times$ speed-ups by eliminating Python dispatch overhead and enabling operation fusion. These advantages diminish as batch size increases, with only vendor-specific solutions (TensorRT for GPU, OpenVINO for CPU) maintaining consistent performance advantages at batch size 16. Architecture significantly influences compiler efficacy: traditional convolutional networks benefit substantially from all compilers at small batches, while transformer-based models show minimal improvement with TVM, moderate gains with ONNX, and substantial acceleration only with vendor-specific tools. While TVM demonstrates substantial performance gains for convolutional architectures, the performance completely tanks for transformer architectures, particularly for ConvNeXt. The weaker TVM results on ConvNeXt are consistent with a search-based optimization that must navigate a

Table 3.6: Aggregated Results by Architectural Family

Family	Compiler	Batch Size(2-4)		Batch Size(8-16)	
		Throughput [t/s] ↑	CPU [%] ↓	Throughput [t/s] ↑	CPU [%] ↓
ResNets	Identity (Orin)	104.69 ± 0.57	11.13 ± 0.78	149.63 ± 0.66	5.52 ± 2.00
	Identity (GPU)	1047.32 ± 14.09	6.24 ± 0.09	2442.30 ± 43.44	6.22 ± 0.10
	Identity (CPU)	68.08 ± 1.75	99.83 ± 0.94	75.12 ± 2.78	95.41 ± 2.60
	TensorRT (Orin)	179.01 ± 6.98	3.45 ± 1.14	247.99 ± 1.28	1.58 ± 0.77
	TensorRT (GPU)	1740.11 ± 14.18	6.26 ± 0.14	2673.77 ± 19.56	6.24 ± 0.03
	OpenVINO (CPU)	136.94 ± 5.80	97.98 ± 0.84	163.90 ± 6.21	96.99 ± 1.40
	TVM (GPU)	1678.78 ± 29.22	6.32 ± 0.17	1823.83 ± 374.62	6.26 ± 0.05
	TVM (CPU)	46.55 ± 0.69	49.99 ± 0.72	60.27 ± 0.99	49.88 ± 0.83
EfficientNets	Identity (Orin)	50.33 ± 0.25	14.29 ± 0.83	64.47 ± 0.17	6.27 ± 3.75
	Identity (GPU)	294.85 ± 4.57	6.25 ± 0.10	1029.42 ± 4.12	6.24 ± 0.08
	Identity (CPU)	30.80 ± 0.92	99.05 ± 1.26	47.34 ± 0.96	97.43 ± 2.90
	TensorRT (Orin)	93.74 ± 0.48	5.31 ± 0.83	113.39 ± 0.36	2.03 ± 0.90
	TensorRT (GPU)	1114.56 ± 8.67	6.23 ± 0.14	1578.73 ± 5.53	6.24 ± 0.17
	OpenVINO (CPU)	96.04 ± 2.35	98.28 ± 0.83	120.00 ± 3.28	97.36 ± 1.32
	TVM (GPU)	1088.48 ± 32.06	6.21 ± 0.12	1261.05 ± 236.20	6.26 ± 0.11
	TVM (CPU)	17.69 ± 0.42	49.96 ± 0.80	20.66 ± 0.22	49.94 ± 0.82
DeiTs	Identity (Orin)	36.45 ± 0.26	6.14 ± 1.39	41.50 ± 0.16	1.89 ± 1.96
	Identity (GPU)	674.76 ± 5.76	6.26 ± 0.14	1015.49 ± 3.17	6.25 ± 0.13
	Identity (CPU)	37.08 ± 1.18	99.59 ± 1.58	50.81 ± 1.55	99.11 ± 1.33
	TensorRT (Orin)	95.67 ± 1.06	1.40 ± 0.84	116.26 ± 0.70	0.60 ± 0.78
	TensorRT (GPU)	1164.57 ± 9.87	6.28 ± 0.15	1531.85 ± 6.62	6.25 ± 0.11
	OpenVINO (CPU)	50.29 ± 1.52	99.01 ± 1.07	61.01 ± 2.27	97.26 ± 2.43
	TVM (GPU)	71.43 ± 4.02	6.25 ± 0.07	87.22 ± 5.54	6.25 ± 0.09
	TVM (CPU)	7.00 ± 0.06	49.92 ± 0.89	8.56 ± 0.07	49.93 ± 0.85
Swins	Identity (Orin)	22.28 ± 0.12	7.20 ± 2.80	25.66 ± 0.05	3.08 ± 4.38
	Identity (GPU)	304.23 ± 2.38	6.25 ± 0.08	591.81 ± 1.24	6.25 ± 0.09
	Identity (CPU)	21.37 ± 0.56	96.62 ± 2.52	23.81 ± 0.52	95.29 ± 2.75
	TensorRT (Orin)	68.00 ± 0.55	1.25 ± 0.74	81.62 ± 0.84	0.56 ± 0.88
	TensorRT (GPU)	918.61 ± 7.71	6.22 ± 0.23	1062.34 ± 3.32	6.25 ± 0.14
	OpenVINO (CPU)	36.75 ± 0.81	99.26 ± 0.98	42.79 ± 1.19	98.01 ± 1.90
	TVM (GPU)	149.52 ± 15.19	6.25 ± 0.09	80.34 ± 3.27	6.27 ± 0.17
	TVM (CPU)	8.01 ± 0.17	49.94 ± 0.81	9.67 ± 0.08	49.90 ± 0.85
ConvNeXts	Identity (Orin)	29.72 ± 0.19	6.51 ± 1.92	34.34 ± 0.08	2.32 ± 2.98
	Identity (GPU)	563.15 ± 5.28	6.24 ± 0.17	862.64 ± 3.24	6.24 ± 0.10
	Identity (CPU)	31.86 ± 1.99	98.31 ± 3.28	45.00 ± 1.33	99.37 ± 1.81
	TensorRT (Orin)	76.98 ± 0.57	2.21 ± 0.97	89.80 ± 0.43	0.74 ± 0.82
	TensorRT (GPU)	972.13 ± 6.93	6.25 ± 0.04	1166.44 ± 3.43	6.26 ± 0.12
	OpenVINO (CPU)	38.91 ± 0.91	99.43 ± 0.94	45.50 ± 1.29	97.13 ± 1.91
	TVM (GPU)	266.40 ± 24.91	6.26 ± 0.10	143.74 ± 17.35	6.25 ± 0.08
	TVM (CPU)	10.67 ± 0.21	49.95 ± 0.80	13.50 ± 0.12	49.91 ± 0.87

large combinatorial schedule space for hybrid blocks [CMJ+18], which is consistent with the compile-time profile in Table 3.3. The architectural hybridization of ConvNeXt exponentially expands the optimization search space, complicates identifying possible fusion patterns, and frequently results in convergence to suboptimal local minima within the computational graph. Conversely, Vendor-specific compilers maintain their advantage through extensive manual optimization targeted specifically at popular cutting-edge architectures. Since the dynamic computational graph is executed sequentially in the

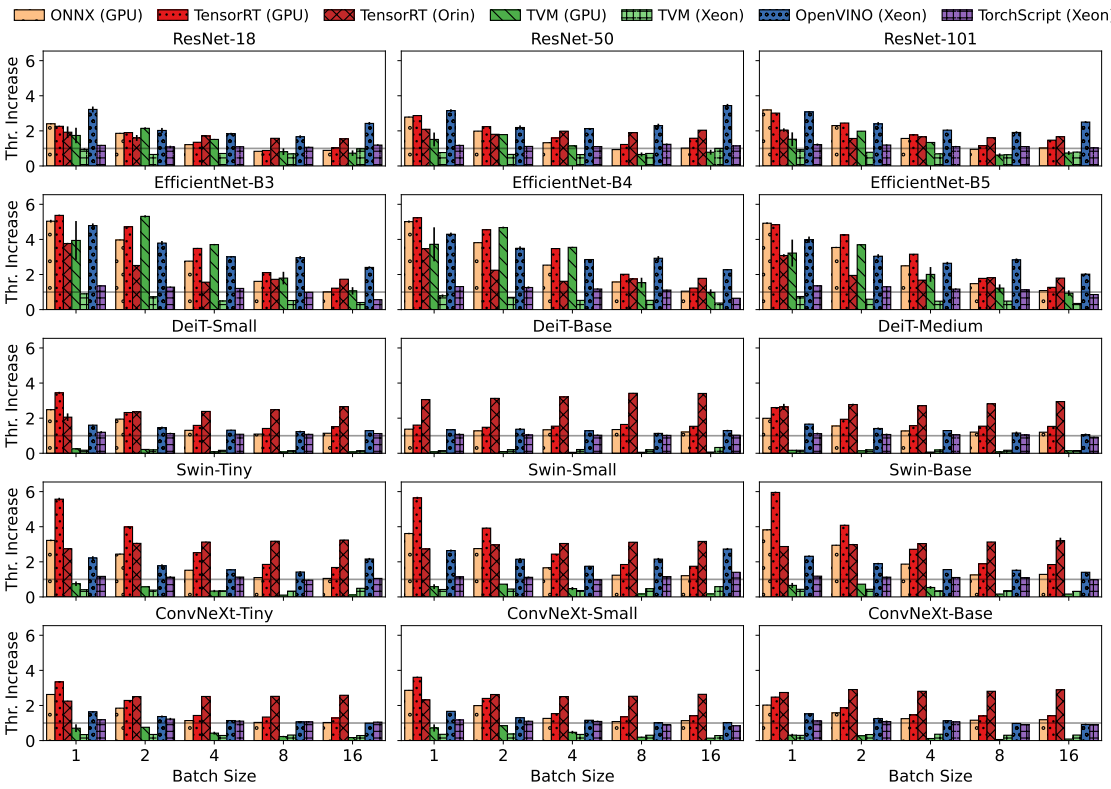


Figure 3.6: Contrasting multiplicative throughput increase relative to the baseline.

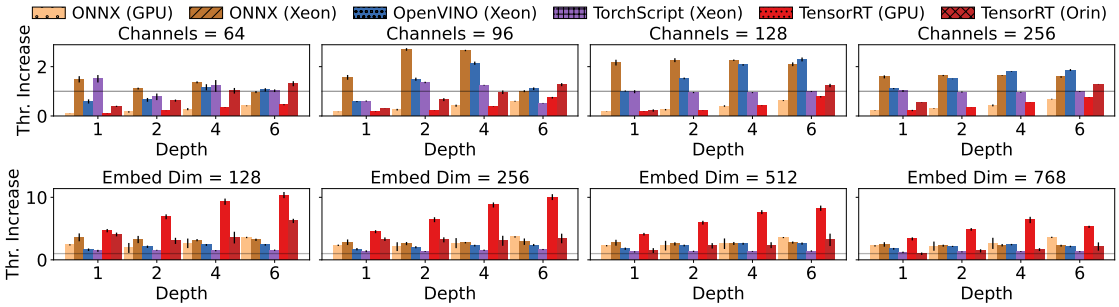
Python runtime on the Xeon CPU, we also include TorchScript as an additional baseline. TorchScript can exploit the multiple cores on the CPU with intra-op parallelism. However, compared to OpenVINO, which achieves up to 5-6 times higher throughput, TorchScript only marginally improves throughput across all configurations.

The results show strikingly distinct performance patterns across compilers, hardware platforms, and neural architectures. Convolutional-based networks benefit from all compilers, while transformer-based models see limited gains from vendor-agnostic solutions. Vendor-specific compilers maintain advantages through targeted optimization for popular architectures, while automated tuning approaches struggle with hybrid designs. Performance advantages are most pronounced at small batch sizes, with only vendor-specific solutions maintaining consistent advantages as batches grow.

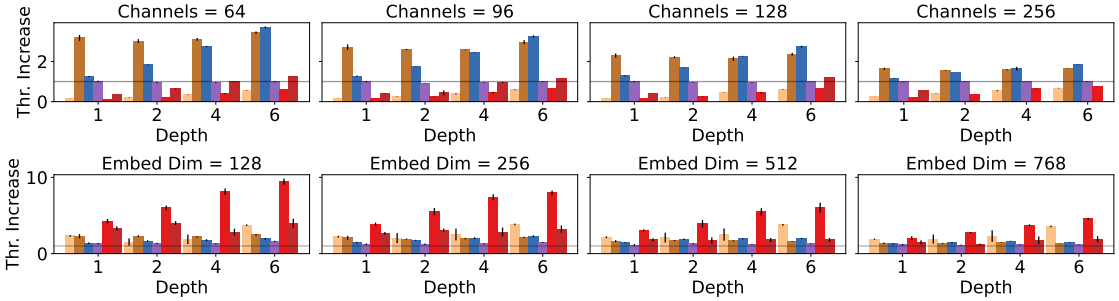
Exploiting Repeated Patterns from Depth Scaling

Figure 3.7 plots the throughput multiplier relative to the uncompiled graph for the convolutional and MHA blocks separately.

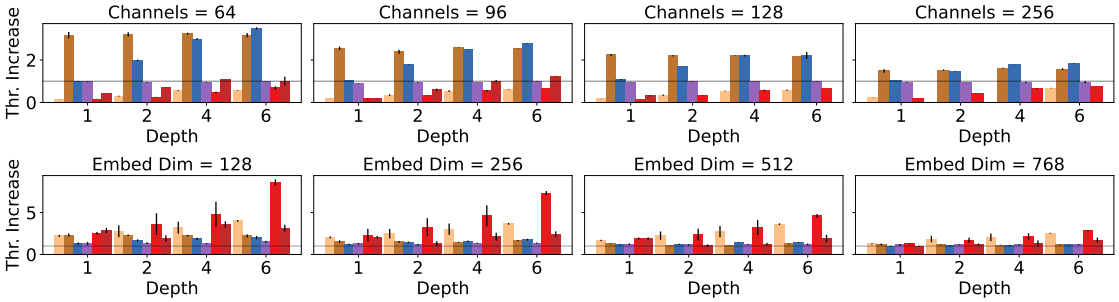
Note that the Y-axis scaling is non-uniform to accentuate the relationship between compiler-device pairs at a set batch size. It contrasts how stacking homogeneous blocks



(a) Batch Size = 2



(b) Batch Size = 8



(c) Batch Size = 16

Figure 3.7: Contrasting how stacking homogeneous blocks improves throughput.

improves throughput over the baseline. The factor diminishes at higher batch sizes, but the relative relationship between depth and the factor remains consistent. Unsurprisingly, we observe comparable behavior of convolutional blocks and MHA blocks as with convolutional-based and transformer-based architectures. However, the depth noticeably impacts the throughput relationship between the compilers. This trend remains even when performance starts to saturate due to high computational load from large batch sizes and block widths. For example, Figure 3.7b shows how the throughput for MHA blocks at batch size 8 with embed dimension 128, using the TensorRT compiler on the GPU, is approximately twice that of the baseline dynamic computational graph on a single block, but jumps to eightfold when stacking six repeating blocks. Conversely, convolutional architectures perform best using ONNX and OpenVINO on Xeon CPUs

Table 3.7: Depth Scaling of Convolutional Blocks

Device	Compiler	Width	Batch Size 8		Batch Size 16	
			Slope	Retention	Slope	Retention
GPU	TensorRT	64	0.08924	3.821	0.1032	1.187
GPU	TensorRT	96	0.1024	0.2116	0.1038	0.6593
GPU	ONNX	64	0.06953	1.153	0.09141	0.426
GPU	ONNX	96	0.07876	1.304	0.08367	0.7538
Orin	TensorRT	64	0.1695	0.7846	0.1349	0.7759
Orin	TensorRT	96	0.1285	1.126	0.212	6.553
Xeon	ONNX	64	0.04862	1.009	0.0006561	0.9756
Xeon	ONNX	96	0.0453	1.025	0.02658	0.9954
Xeon	OpenVINO	64	0.4417	0.3379	0.4775	0.7428
Xeon	OpenVINO	96	0.3553	0.8669	0.3378	0.4106

across the majority of experiment configurations. Table 3.7 and Table 3.8 quantify how varying compilers can leverage the repeating patterns. We compute the *slope* with the least-squares fit to measure how much speed-up changes as we increase the depth. A positive slope implies that each additional layer makes the compiler's advantage even larger. In contrast, a negative slope means that extra blocks diminish initial gains. The *retention* is simply the ratio between the speed-up factor of the deepest stack and the speed-up factor of a single block. A value close to 1 implies that a compiler is agnostic towards the depth parameter, i.e., it cannot leverage the repeated block patterns. Values above 1 imply that the compiler can leverage repeated blocks.

For convolutional blocks, the speed-ups of the vendor-specific compilers with hardware optimization are *amplified* by increasing the depth. The convolutional blocks are a repeated sequence of the simple `Conv →BatchNorm →ReLU` pattern. The simple design aids the hardware-based compilers. The vendor-specific compilers can leverage the repeating patterns and tile and fuse them into smaller or larger kernels. For example, this can occur by collapsing all 3N pointwise ops into one fused pass, or merging multiple 2D convolutions into one multi-stage convolution that reuses intermediate results. As we increase the depth, the *amortized* overhead per block of kernel launch, memory barriers, and descriptor setup decreases. Conversely, the software-based optimization of ONNX shows weaker improvements.

The retention values of MHA blocks are, on average, less than those of convolutional blocks, i.e., the compilers cannot leverage the repeated patterns as effectively. The dependency graph of the `Linear(QKV) →Reshape →MatMul →Softmax →MatMul →Add →LayerNorm →ReLU` is significantly more complex, such that we may get good one-block kernels, but stacking them does not result in intra-block fusion.

Table 3.8: Depth Scaling of MHA Blocks

Device	Compiler	Width	Batch Size 8		Batch Size 16	
			Slope	Retention	Slope	Retention
GPU	TensorRT	128	0.8207	1.404	1.028	0.556
GPU	TensorRT	256	0.6369	0.6935	0.8831	1.471
GPU	ONNX	128	0.3594	2.187	0.3106	0.83
GPU	ONNX	256	0.3406	1.724	0.2929	0.9108
Orin	TensorRT	128	0.05291	1.178	0.173	0.9055
Orin	TensorRT	256	0.08368	0.95	0.1488	1.629
Xeon	ONNX	128	0.04244	1.105	-0.01775	1.018
Xeon	ONNX	256	0.01867	1.031	0.0215	0.9735
Xeon	OpenVINO	128	0.1113	1.115	0.1355	0.9635
Xeon	OpenVINO	256	0.1589	1.577	0.1015	1.029

Vendor-specific compilers achieve increasing throughput with depth by exploiting repeated block structures for deeper fusion and lower launch overhead. Convolutional blocks (Conv→BatchNorm→ReLU) show superlinear scaling under TensorRT and OpenVINO, as their regular patterns enable kernel tiling and fusion. Transformer-style blocks (QKV→Softmax→Add→Norm) saturate early since inter-layer dependencies hinder intra-block fusion, producing additive rather than multiplicative gains. Depth interacts multiplicatively with batch size: GPUs favor deep, narrow models, while CPUs benefit from shallower, wider ones due to cache and scheduling limits. Repeated simple patterns yield disproportionate efficiency in constrained settings, emphasizing compiler-aware design that favors regular, fusable structures.

Batch Parallelization Scaling Efficiency

From both Figure 3.6 and Figure 3.7, it is apparent that increasing the batch size significantly influences the throughput rate, and different compilers exhibit varying behavior. Figure 3.8 contrasts between architectural styles explicitly to show how increasing the batch size decreases the scaling efficiency despite increasing the raw throughput.

As performance saturates at higher batch sizes, the throughput gain from parallelization diminishes. Apache TVM shows considerable but inconsistent scaling efficiency for convolutional-based architectures on the GPU. This is expected due to TVM's search-heuristic-based optimization, i.e., we must start a new search for each batch size. Conversely, the scaling efficiency decay of TensorRT is more predictable, as it is consistent with negligible variance.

To account for varying compute capacities, and to provide information on relative improvement over the dynamic computational graph baseline, Figure 3.9 plots the batch scaling resilience (Section 3.2.3) for each architectural style.

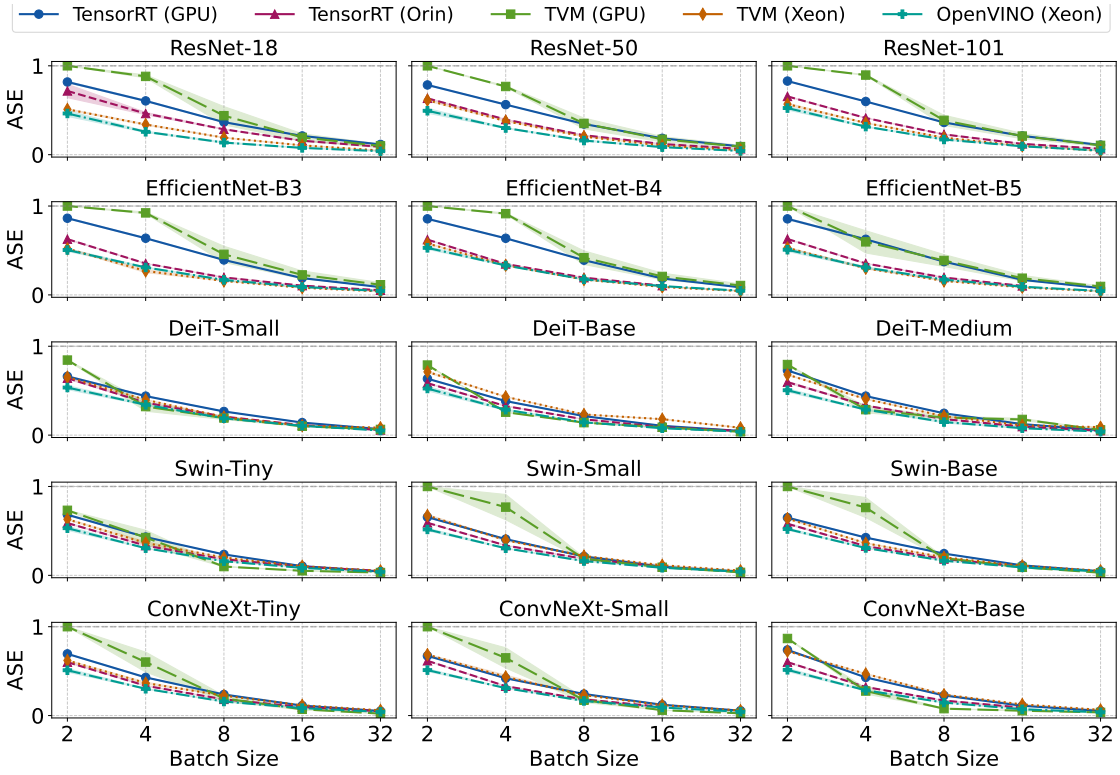


Figure 3.8: Contrasting ASE (higher is better) of architectural styles.

TensorRT has BSR ≈ 1 across most architectures, which implies consistent throughput gains over the baseline. Note that a BSR below 1 implies steeper efficiency losses relative to the baseline. We argue that a BSR below 1.0 indicates that there are potentially further opportunities for optimization that the compiler has missed. The intuition is that if the compiler has found a global maximum, the drop in scaling efficiency should be *at worst* consistent between the unoptimized dynamic and the optimized compiled computational graph. In particular, the erratic results of TVM demonstrate that specific optimizations exist that the corresponding vendor does not adequately consider, but they are challenging to find. For example, applying TVM to the mid-sized DeiT shows significantly higher resilience than OpenVINO on the Xeon CPU. Conversely, the resource-constrained Orin shows a BSR of roughly 1.0 across transformer-based architectures for all sizes while showing substantial throughput gains for the same architecture. However, especially for smaller architectures, the BSR of TensorRT on the comparatively powerful GPU is consistently below 1.0.

The results indicate a consistent relationship between compiler optimization strategy and runtime saturation behavior. When scaling efficiency decreases at larger batch sizes, it often reflects limited overlap between data movement and computation once the compiler's kernel fusion scope is reached.

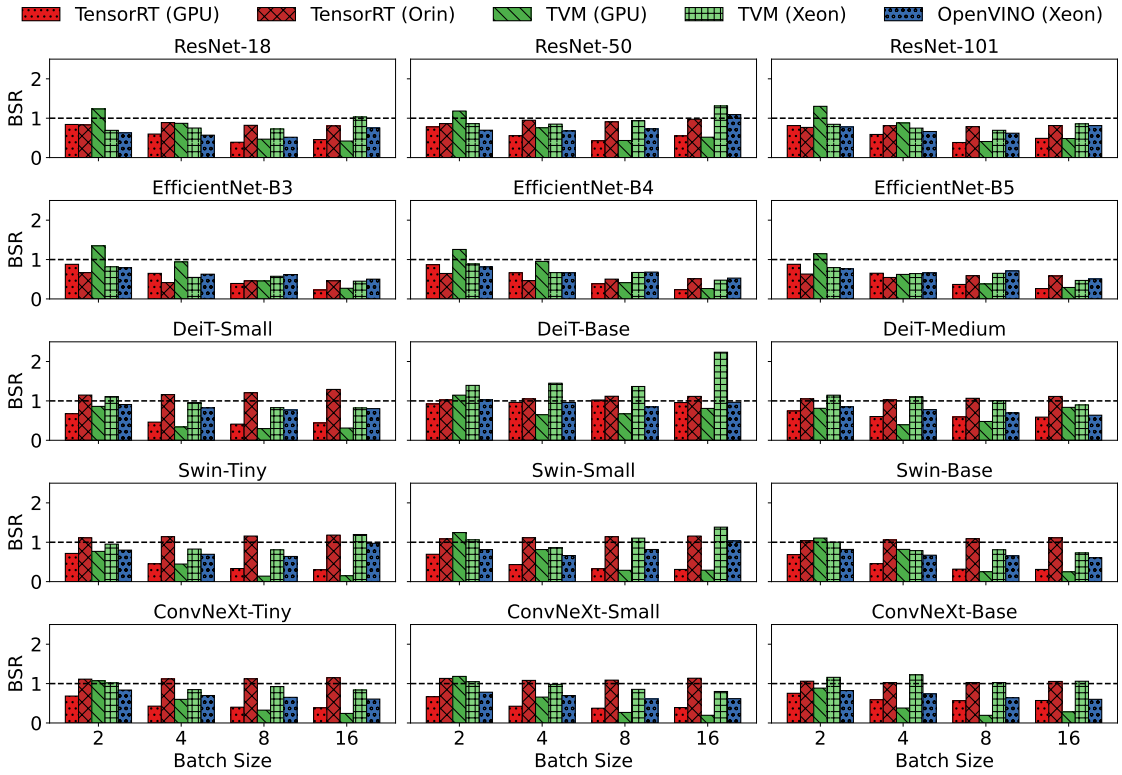


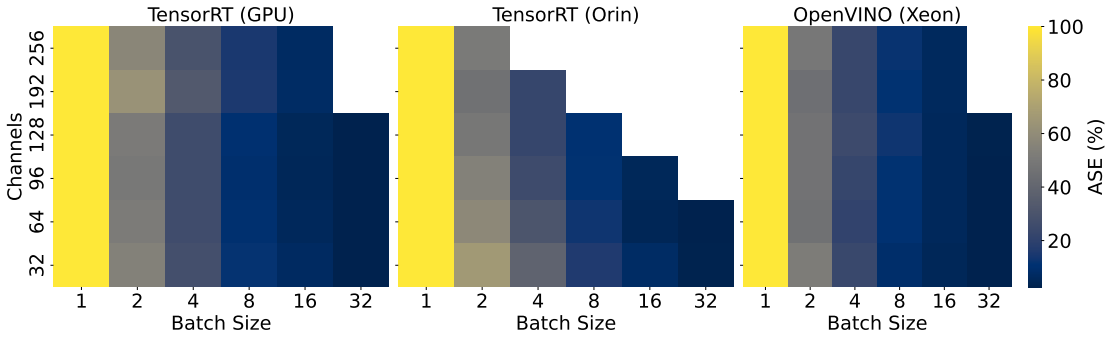
Figure 3.9: Contrasting BSR (higher is better) of architectural styles.

We conjecture that Vendor-specific compilers maintain near-constant BSR because their fusion templates are tuned for common convolutional and attention patterns. In contrast, search-based approaches such as TVM show irregular scaling since each batch configuration exposes a different optimization boundary. Hence, compiler efficiency does not grow smoothly with batch size but changes discretely with internal scheduling thresholds. For evaluation design, benchmarking a single batch configuration can overestimate scalability. Measuring ASE and BSR across several batch-width regimes provides a more accurate view of achievable throughput. Compilers with stable BSR values yield predictable latency–throughput trade-offs, which is preferable for systems that rely on dynamic batching or experience variable workload intensity.

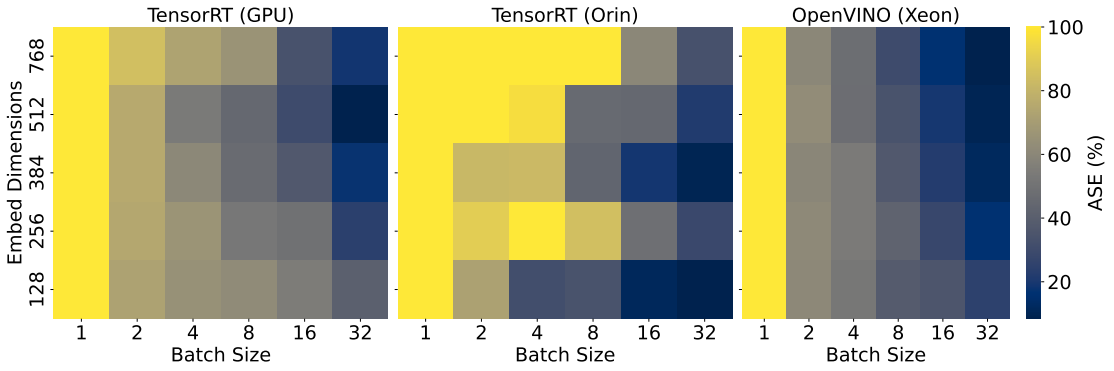
The Batch Scaling Resilience (BSR) metric uncovers compiler-specific optimization patterns, demonstrating that TensorRT achieves consistent scaling profiles (BSR ≈ 1) for most architectures while TVM shows erratic but occasionally superior resilience for specific model-hardware combinations. The results show that compilers can more easily optimize for resource efficiency when resources are scarce. When resources are abundant, BSR values below 1.0 suggest that further optimizations are possible.

Batch-Width Scaling Friction Mitigation

We investigate whether compilers can mitigate the batch-width scaling friction described in Definition 1 block-level experiments.



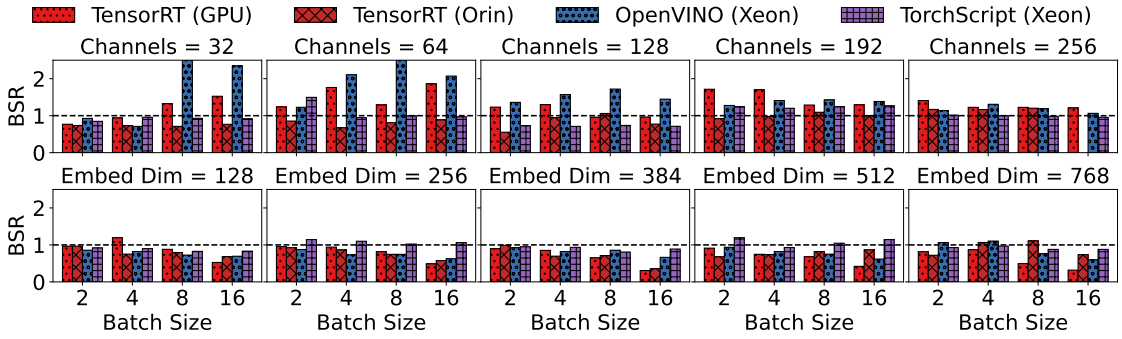
(a) Convolutional Blocks (Depth=6)



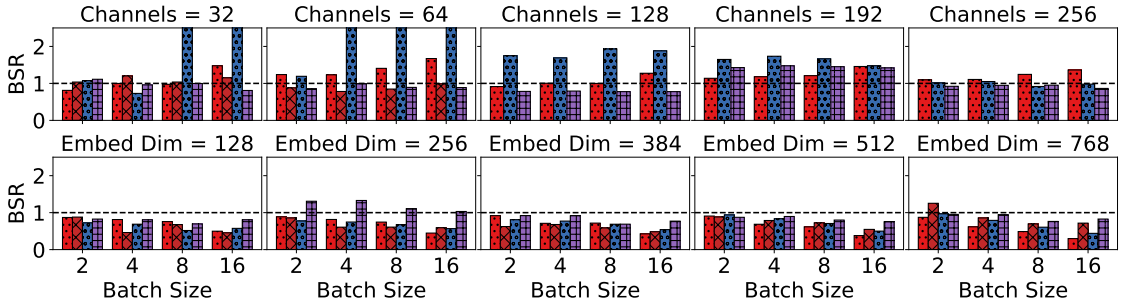
(b) MHA Blocks (Depth=6)

Figure 3.10: Heatmaps plotting the effect of width on ASE.

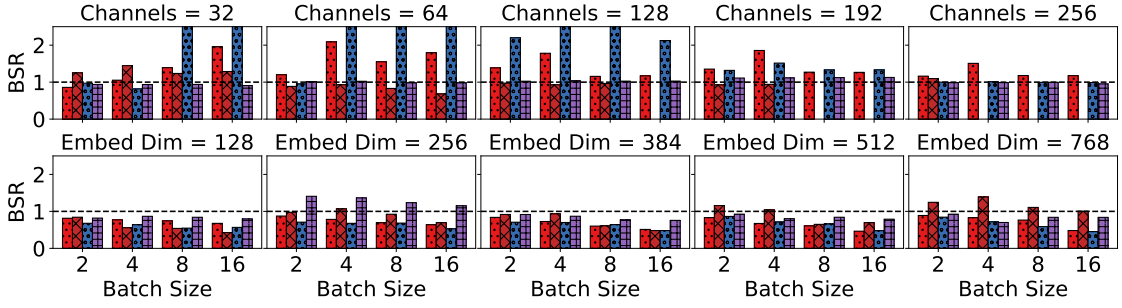
The batch-width friction is directly apparent from Figure 3.10. On all device-compiler pairs, the scaling efficiency decreases faster for wider networks, but the rate varies. As we increase the width for a block, the efficiency scaling drops considerably faster from one batch size to the next larger batch size. However, to account for hardware differences and to compare with the baseline dynamic computational graph Figure 3.11 plots the BSR for three depth configurations. Successful optimization on the CPU shows improved parallelization rates even at higher batch sizes. Increasing the depth tends to moderately improve BSR for TensoRT, arguably for the same reasons as outlined in Section 3.2.4. TorchScript slightly mitigates the scaling friction for some configurations through intra-ops parallelization, which is expected. A BSR value higher than one implies that scaling efficiency decreases more gracefully relative to the uncompiled dynamic graph. This is best seen with OpenVINO. For the convolutional blocks, it can considerably mitigate the efficiency decrease by exploiting the multiple cores. The results indicate that batch-width scaling friction primarily stems from memory-bandwidth contention



(a) Depth = 1



(b) Depth = 3



(c) Depth = 6

Figure 3.11: Contrasting BSR between convolutional and MHA blocks.

and kernel-launch serialization as tensors expand laterally. Increasing the width increases synchronization cost between fused operations, which reduces effective parallel overlap within compute units. Compilers may mitigate this by restructuring fusion boundaries and adjusting tile granularity. Vendor-specific compilers such as TensorRT rely on pre-defined fusion templates and optimized CUDA launch graphs, which benefit deep but narrow configurations where per-kernel reuse is high. Conversely, OpenVINO's thread-level parallelism distributes wide tensor partitions across cores, explaining its stronger friction mitigation for convolutional blocks. Transformer-based blocks see greater friction because attention layers may introduce irregular memory access and residual connections that hinder static scheduling. Consequently, compilers can only partially amortize the width-induced overhead through operator fusion. In practice, this implies

that widening layers should be balanced against attainable batch parallelism under the target compiler: depth scaling favors environments with GPUs, whereas moderate width scaling with multi-core exploitation is preferable on CPU-based deployments. These interactions highlight that compiler-aware architecture design can reduce scaling friction without altering model semantics, emphasizing the need to evaluate joint width-batch trade-offs during system optimization.

Batch-width scaling friction originates from increased memory synchronization and reduced kernel reuse as model width expands. Compilers differ in their ability to restructure execution to offset this loss. Vendor-specific compilers such as TensorRT mitigate friction through deep-kernel fusion and launch-graph optimization, favoring narrow but deep configurations. OpenVINO may achieve stronger mitigation for convolutional blocks by distributing wider tensor partitions across CPU cores. Transformer-style blocks are limited by irregular memory access and residual dependencies that can hinder static fusion. Overall, effective friction mitigation depends on the compiler's capacity to balance fusion depth with parallel partitioning, implying that width scaling should be aligned with the underlying compiler–hardware concurrency model.

Resource Usage Reduction

From Table 3.6 it is apparent that when compilers successfully optimize the graph to have considerable throughput gains, the CPU usage increases on the Xeon where there is no dedicated GPU. On the GPU server and the Jetson board, TensorRT can decrease the CPU usage - marginally on the powerful server and significantly on the constrained Jetson board. TensorRT on GPU leverages static-graph capture and aggressive kernel scheduling to slash CPU-side launch overhead, which on the constrained Jetson Orin's SoC shows up as CPU-usage drops. On a higher-end GPU Server, these savings are negligible.

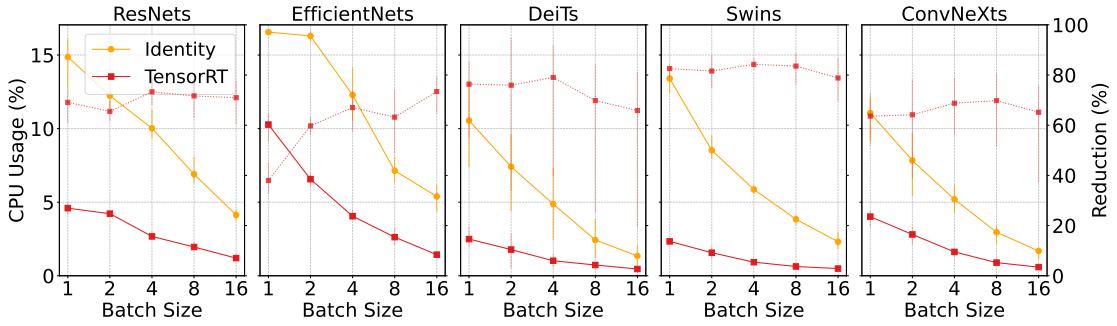


Figure 3.12: The left Y-axis (solid line) shows the absolute CPU usage. The left Y-axis (dashed line) shows the relative CPU reduction. When the batch size increases, throughput performance saturates on the GPU, such that the CPU usage reduces.

Figure 3.12 directly compares the CPU usage of TensorRT on the compiled network

and the baseline on all architectural families on the Orin device. Notice that for the transformer-based architectures, particularly for the Swin family, there is up to 80% reduction in CPU usage. This is valuable in constrained environments that perform auxiliary tasks on the CPU. For example, in [FZR+25], interference from pre- and post-processing on the CPU was negligible on smaller models but adversely affected the throughput of larger models.

Applying compilers on devices with a dedicated accelerator can significantly reduce CPU utilization (up to 80%) through static-graph capture and kernel scheduling optimizations. This reduction is particularly valuable in edge computing scenarios where horizontal scaling is limited and CPUs may handle concurrent auxiliary tasks. These findings indicate compiler selection should consider both throughput and resource utilization metrics when deploying neural networks in resource-constrained edge-cloud systems.

Discussion

Our findings are largely consistent with prior benchmarking work, though our scope differs methodologically. Earlier studies typically focus on isolated compilers or per-operation performance, whereas our evaluation integrates compiler effects into end-to-end model benchmarking. Consistent with Zhou & Yang [ZY22] and Li et al. [LLL+21], we observe that vendor-specific compilers (e.g., TensorRT, OpenVINO) achieve superior throughput due to aggressive fusion of supported operator patterns. However, unlike Xing et al. [XWW+19], who report stable performance ordering across architectures, our cross-compiler evaluation shows that compilation can invert relative throughput rankings when optimization coverage diverges. This apparent inconsistency arises from differences in experimental design, explained by our methodology using fully compiled workloads on heterogeneous edge–cloud hardware rather than operation-level microbenchmarks. Moreover, while Zhang et al. [ZLC+22, ZCC+24] attribute performance variance across libraries to framework differences, our results subsume their finding that underlying compiler optimizations are the decisive factor. In summary, our work is complementary to earlier studies by extending the benchmarking dimension from framework-level to compiler-level and providing diagnostic metrics such as BSR for future analyses.

3.3 Summary

The chapter introduced a framework for incorporating compiler effects throughout the research process for Edge-Cloud systems relying on NNs. Empirical analysis demonstrated that optimizations can completely invalidate performance expectations by systematically analyzing compiler behavior across heterogeneous platforms. The introduced Batch Scaling Resilience metric quantifies a compiler's ability to mitigate performance friction as batch size increases. Block-level experimentation confirmed that simple compositions with widely supported operations provide significant advantages in resource-constrained

environments, as compilers effectively leverage repeated patterns for disproportionate throughput gains. However, despite our comprehensive analysis, the evaluation relies on a finite heterogeneous testbed, which does not capture the full diversity of emerging accelerator architectures. Scaling the methodology to distributed settings or deployments that include model compression is left for future investigation.

CHAPTER 4

Neural Feature Compression

This chapter handles the demands of latency-sensitive and performance-critical applications by efficiently reducing transmission costs. Efficency here implies (I) the operator need not know about client tasks, (II) latency reduction from lower transmission costs, and (III) preserving enough information for (near-)lossless prediction performance. The encoder design is purposefully simple, consisting of vanilla convolutional layers, as the previous chapter has shown that this layer type sees the most consistent support across vendors.

4.1 Shallow Variational Bottleneck Injection

The section describes the core method for addressing the need for efficient bandwidth reduction from ANN inference requests. It first provides a broad overview of existing approaches and discusses their limitations.

4.1.1 Introduction

Problem domains relying on ANN inference range from Computer Vision (CV) [VDDP18] to Natural Language Processing (NLP) [OMK20]. Complementary with the advancements in mobile edge computing (MEC) [FHZ+22] and energy-efficient accelerators, visions of intelligent city-scale platforms for demanding applications, such as mobile augmented reality (MAR) [RHS+21b], disaster warning [TD22], or facilities management [XAD+22], are increasingly feasible. Progress in energy-efficient ASICs and embedded AI with model compression, such as quantization, pruning, or knowledge distillation, may permit constrained devices to host lightweight ANNs. Yet, irrespective of advancements for local computing, network providers see unprecedented growth in Machine-to-Machine (M2M) communication [Cis20] from mobile clients. The same methods that facilitate local computing on constrained devices are also applicable to large models, and when

prediction performance is critical, it is preferable to offload requests to the most accurate model available. Moreover, hosting foundational models is convenient as it gives operators more control over optimizing inference [FD22] and reduces the maintenance overhead as a single model may accommodate various tasks [JT21, ANK+25]. Besides leaving local resources idle, the downside to offloading is that by constantly streaming high-dimensional visual data, the limited bandwidth will inevitably lead to network congestion, resulting in erratic response delays. As discussed in Section 1.2 and Section 2.2.7, Split Computing (SC) emerged as an alternative to alleviate inefficient resource utilization and to facilitate latency-sensitive and performance-critical inference. To recap, the idea is to partition an ANN to process the shallow layers with the client and send a processed representation to the remaining deeper layers deployed on a server. SC can draw resources from the entire edge-cloud compute continuum, but its applications are limited. They assume extremely constrained conditions, are tailored toward specific neural network architectures, and break server-side transparency. We identify two critical design flaws of SC that render them impractical in MEC. First is forcing the requirement for reducing server-side workload when there is significant resource asymmetry. Second is the black-box approach that treats ANN inference as a special workload.

4.1.2 Related Work

Neural Data Compression

Image Compression Lossy image compression minimizes bitrates given a constraint on the distortion or perceptual quality of the reconstruction [BM19, Sha59]. Transform coding is a basic framework of lossy compression, which divides the compression task into decorrelation and quantization [Goy01]. Decorrelation reduces the statistical dependencies of the pixels, allowing for more effective entropy coding, while quantization represents the values as a finite set of integers. The core difference between handcrafted and learned methods is that the former relies on linear transformations based on expert knowledge. Contrarily, the latter is data-driven with nonlinear transformations learned by neural networks [BCM+20].

Ballé et al. introduced the Factorized Prior (FP) entropy model and formulated the neural compression problem by finding a representation with minimal entropy [BLS17]. An encoder network transforms the original input to a latent variable, capturing the input's statistical dependencies. In follow-up work, Ballé et al. [BMS+18] and Minnen et al. [MBT18] extend the FP entropy model by including a hyperprior as side information for the prior. Minnen et al. [MBT18] introduce the joint hierarchical priors and autoregressive entropy model (JHAP), which adds a context model to the existing scale hyperprior latent variable models. Typically, context models are lightweight, i.e., they add a negligible number of parameters, but their sequential processing increases the end-to-end latency by orders of magnitude. The empirical analysis will describe and include more recent methods as baselines.

Feature Compression Singh et al. demonstrate a practical method for the Information Bottleneck principle in a compression framework by introducing the bottleneck in the penultimate layer and replacing the distortion loss with the cross-entropy for image classification [AMT+17]. Dubois et al. generalized the VIB for multiple downstream tasks and were the first to describe the feature compression task formally [DBRUM21]. However, their encoder-only CLIP compressor has over 87 million parameters. Both Dubois and Singh et al. consider feature compression for mass storage, i.e., they assume the data is already present at the target server. In contrast, we consider how resource-constrained clients must first compress the high-dimensional visual data before sending it over a network. Closest to our work is the Entropic Student (ES) proposed by Matsubara et al. [MYLM23, MYLM22], as we follow the same objective of real-time inference with feature compression. However, the ES exhibits the same limitations as other bottleneck injection methods. We carefully examine the problem domain of resource-conscious feature compression to identify underlying issues with current methods, allowing us to derive training objectives and decoder designs with significantly better performance.

Split Computing

We focus on Split Computing (SC) and distinguish between two orthogonal approaches. SC corresponds to depthwise partitioning discussed in Section 2.2.7. Widthwise partitioning is omitted, as the objective is to minimize request latency and bandwidth requirements when offloading requests to foundational models running on powerful hardware.

Split Runtimes are characterized by performing no or minimal modifications on off-the-shelf ANNs. The objective is to dynamically determine split points according to the available resources, network conditions, and intrinsic model properties. Hence, split runtimes primarily focus on profilers and adaptive schedulers. Kang et al. performed extensive computational cost and feature size analysis on the layer-level characterizations of ANNs and introduced the first split runtime system [KHG+17]. Their study has shown that split runtimes are only sensible for ANNs with an early natural bottleneck, i.e., models performing aggressive dimensionality reduction within the shallow layers. However, most modern ANNs increase feature dimensions until the last layers for better representation. Consequently, follow-up work focuses on feature tensor manipulation [LHJ+18, LVA+20, ALV+22]. We argue against split runtimes since they introduce considerable complexity and force operators to tune the system towards external conditions, with extensive profiling and careful calibration. Additionally, runtimes raise overhead and another point of failure by hosting a network-spanning system. Notably, even the most sophisticated methods still rely on a natural bottleneck, evidenced by how state-of-the-art split runtimes still report results on superseded ANNs with an early bottleneck [LZLG22, BMZ+23].

Artificial Bottleneck Injection retains the simplicity of offloading by shifting the effort towards modifying and re-training an existing base model (backbone) to replace the shallow layers with an artificial bottleneck. Eshratifar et al. replace the shallow

layers of ResNet-50 with a deterministic autoencoder network [EEP19]. A follow-up work by Jiawei Shao and Jun Zhang further considers noisy communication channels [SZ20]. Matsubara et al. [MBC+19], and Sbai et al. [SSTM21] propose a more general network agnostic knowledge distillation (KD) method for embedding autoencoders, where the output of the split point from the unmodified backbone serves as a teacher.

4.1.3 The Case for Neural Data Compression

We assume an asymmetric resource allocation between mobile devices and servers, where the latter has considerably higher computational capacity. Additionally, we consider large models for state-of-the-art performance of non-trivial discriminative tasks unsuitable for mobile clients.

Limitations of depthwise partitioning in Mobile Edge Computing

Figure 4.1 illustrates generic on/offloading and split runtimes that illustrate how SC may conditionally draw resources from two computational tiers when binary on- or offloading decision mechanisms will leave either client or server-side resources idle.

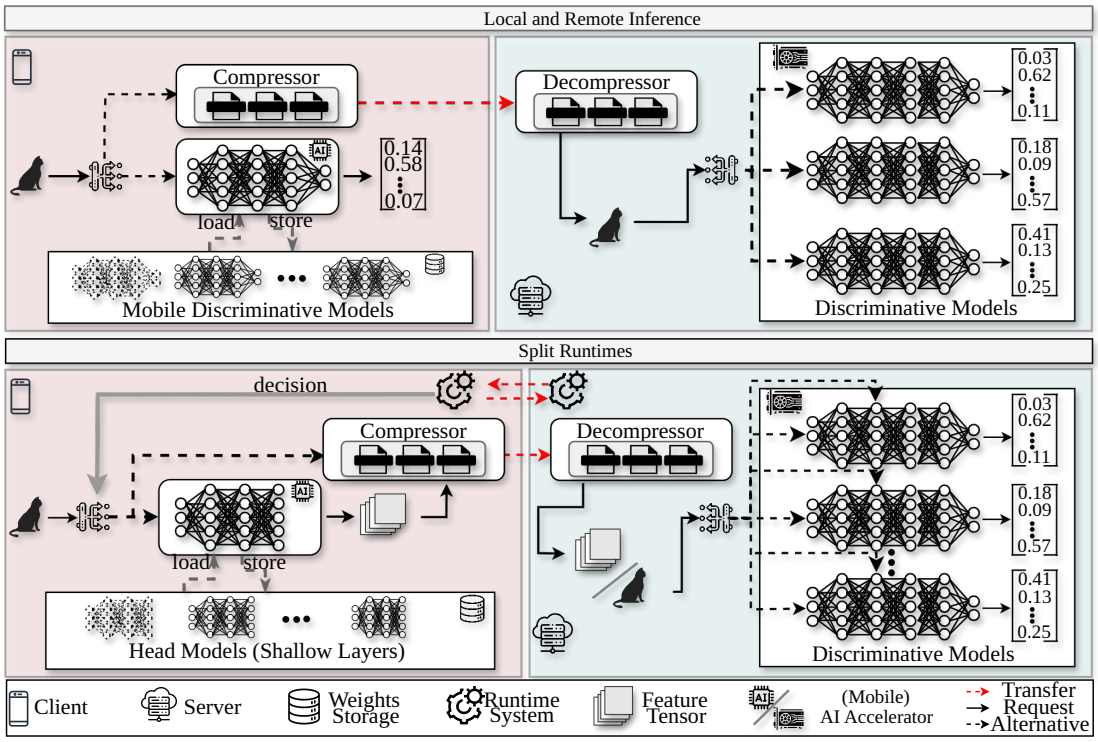


Figure 4.1: Prediction with on/offloading and split runtimes

The caveat is that both SC approaches discussed in Section 4.1.2 are only conditionally applicable. In particular, split runtimes reduce server-side computation for inference

tasks with off-the-shelf models by onloading and executing shallow layers at the client. This approach introduces two major limitations.

First, when the latency is crucial, this is only sensible if the time for client-side execution, transferring the features, and remotely executing the remaining layers is less than the time of directly offloading the task. More recent work [LZLG22, ALV+22, BMZ+23] relies on carefully calibrated dynamic decision mechanisms. A runtime component periodically measures (e.g., network bandwidth) and internal conditions (e.g., client load) to measure ideal split points or whether direct offloading is preferable. Second, since the shallow layers must match the deeper layers, split runtimes cannot accommodate applications with complex requirements, which is a common justification for MEC (e.g., MAR). Constrained clients would need to swap weights from the storage in memory each time the prediction model changes. Worse, the layers must match even for models predicting the same classes with closely related architectures. Hence, it is particularly challenging to integrate split runtimes into systems that can increase the resource efficiency of servers by adapting to shifting and fluctuating environments [RLYK21, ZZC+22]. For example, when a client specifies a target accuracy and a tolerable lower bound, the system could select a ResNet-101 that can hit the target accuracy but may temporarily fall back to a ResNet-50 to ease the load when necessary.

Execution Times with Resource Asymmetry. Table 4.1 summarizes the results of a simple experiment to demonstrate limitations incurred by resource asymmetry. The client is an Nvidia Jetson NX2 equipped with a low-powered accelerator, and the server hosts an RTX 3090 (see Section 4.2.2 for details on hardware configurations). We measure the execution times of ResNet variants, classifying a single $3 \times 224 \times 224$ tensor at two split points.

Table 4.1: Execution Times of Split Models

Model	Split Index	Head [NX2] (ms)	Head [3090] (ms)	Tail [3090] (ms)	Rel. Exec. [NX2] (%)	Contribution [NX2] (%)
ResNet-50	Stem	1.5055	0.1024	4.9687	23.25	0.037
	Stage 1	8.2628	0.9074	4.0224	67.26	0.882
ResNet-101	Stem	1.5055	0.1024	9.8735	13.23	0.021
	Stage 1	8.2628	0.9074	8.9846	47.91	0.506
ResNet-152	Stem	1.5055	0.1024	14.8862	9.18	0.015
	Stage 1	8.2628	0.9074	13.8687	37.34	0.374

Similar to other widespread architectural families, ResNets organize their layers into four top-level layers, and the top-grained ones recursively consist of finer-grained ones. While the terminology differs for architectures, we will uniformly refer to top-level layers as *stages* and the coarse-grained layers as *blocks*.

Split point *stem* assigns the first preliminary block as the head model. It consists of a convolutional layer with batch normalization [IS15] and ReLU activation, followed

by max pooling. Split point *Stage 1* additionally assigns the first stage to the head. Notice how the shallow layers barely constitute the overall computation, even when the client takes more time to execute the head than the server for the entire model. Further, compare the percentage of total computation time and relate them to the number of parameters. At best, the client contributes to 0.02% of the model execution when taking 9% of the total computation time and may only contribute 0.9% when taking 67% off the computation time.

Despite a powerful accelerator, it is evident that utilizing client-side resources to aid a server is inefficient. Consequently, SC methods commonly include some form of quantization and data size reduction to offset resource asymmetry. In the following, we conceive a hypothetical SC method to provide intuition behind the importance of reducing transfer costs.

Feature Tensor Dimensionality and Quantization. Studies on SC commonly start with some statistical analysis of the output layer dimensions, as illustrated in Figure 4.2. Excluding repeating blocks, the feature dimensionality is identical for ResNet-50, -101, and -152. The red line marks the cutoff where the size of the intermediate feature tensor is less than the original input. ResNets (including more modern variants [XGD+17]), among numerous recent architectures [LLY+22, HWC+22], do not have an early natural bottleneck and will only drop below the cutoff from the first block of the second stage (S3RB1-2). Since executing until S3RB1-2 is only about 0.06% of the model parameters of ResNet-152, the computational overhead may seem negligible. However, as shown in Table 4.1, even when executing 0.04% of the model, the client will make up 37% of the total computation time. More recent approaches reduce the number of layers a client must execute, quantize the feature tensor, and apply other clever methods that statically or dynamically prune channels [MLR22].

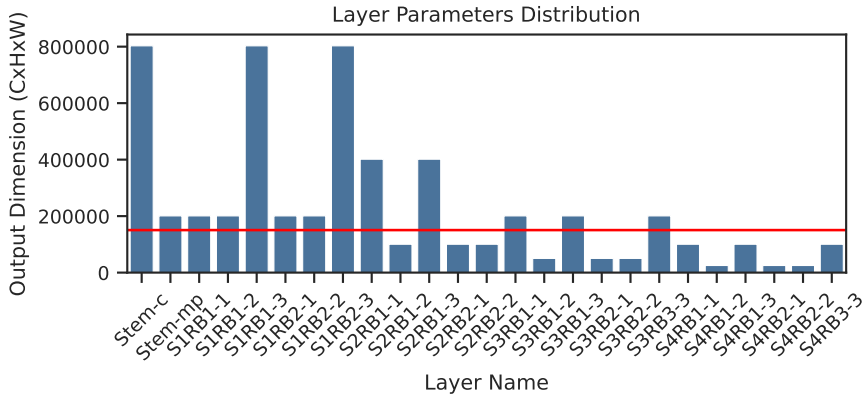


Figure 4.2: Output dimensionality distribution for ResNet

For our hypothetical method, we use the execution times from Table 4.1. We generously assume that the method applies feature tensor quantization and channel pruning to

reduce the expected data size without a loss in prediction performance. While early work has experimented with graph compilers on the local device [EEP19] to mask inefficiencies, we omit applying TensotrRT as Chapter 3 has shown how graph compilers will at least match the performance gains and may increase the resource asymmetry. Instead, we reward the client for executing deeper layers to reflect deterministic bottleneck injection methods, such that the output size of the stem and stage one are 802816 and 428168 bits. For stage one, this is roughly a 92% reduction relative to its original FP32 output size. Yet, the plots in Figure 4.3 show that offloading with PNG, let alone more modern lossless codecs, will beat SC in total request time, except when the data rate is severely constrained.

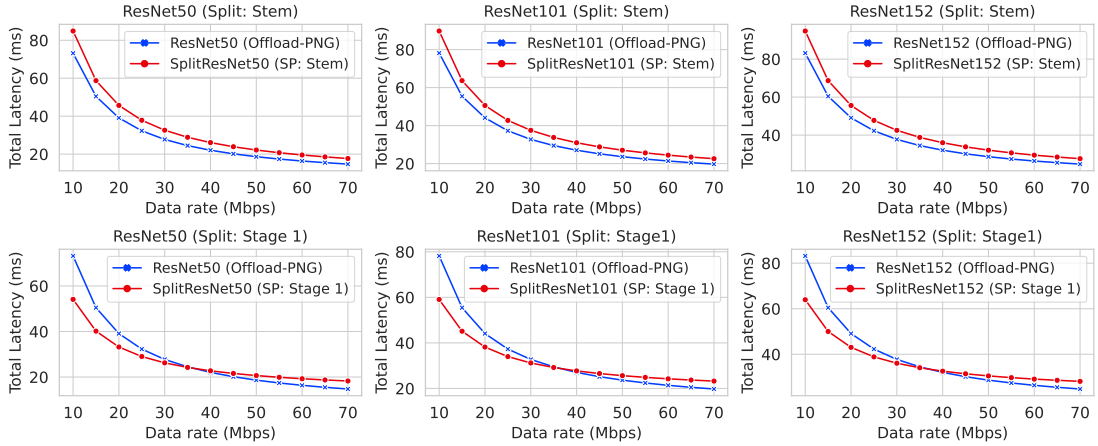


Figure 4.3: Inference latency for SC and offloading

Since partially unloading task inference locally is clearly not an efficient use of local resources, we propose an approach driven by the question: *Is it more efficient to focus the local resources exclusively on compressing the data rather than executing shallow layers of a network that would constitute a negligible amount of the total computation cost on the server?*. The following elaborates on diverging approaches to compression, its role in SC, and why applying existing methods cannot meet the requirements of MEC.

Dimensionality Reduction and Transform Coding

The most common method to reduce the transmission costs in SC literature is to crudely reduce the latent dimensions. Dimensionality reduction may seem effective on toy datasets. However, this is more due to the overparameterization of large ANNs. Precisely, for a toy dataset, we can prune most channels or inject a small autoencoder at the shallow layers that may appear to achieve unprecedented compression rates relative to the unmodified head's feature tensor size. Section 4.2.2 will show methods that work reasonably well on a toy dataset, with performance collapsing on more challenging datasets. Dimensionality reduction may approximate compression, but is not equivalent to it [MBT18]. The output

dimensionality of a layer is an uninformed measure of transmission costs [SZT17]. A trivial counterexample is a high-dimensional but extremely sparse tensor. Conversely, compression reduces the entropy of the latent under a prior shared between the sender and the receiver.

Transform coding is the underlying framework of principled approaches to lossy compression, consisting of decorrelation and quantization [Goy01]. Decorrelation reduces the statistical dependencies of the pixels, while quantization represents the values as a finite set of integers. The difference between handcrafted and learned methods is that the latter is data-driven with nonlinear transformations [BCM+20]. The nonlinear transformations are typically ANNs, hence, Neural Image Compression (NIC). Notably, larger models can learn more powerful representations, so *increasing the dimensions* of encoder latents may *reduce* transmission costs, which we will empirically show in Section 4.2.2. Despite outperforming handcrafted codecs [YMT23], we will empirically show that NIC is unsuitable for real-time inference in MEC since it consists of large models and other complex mechanisms. Moreover, they are designed with an objective that either maximizes PSNR with a reconstruction loss or with a perceptual loss, leaving considerable room for improving efficiency.

The following derives the objective that addresses the limitations of methods not intended to minimize request latency *through* compression.

4.1.4 Semantic Rate-Distortion

Using Shannon's rate-distortion (r-d) theory [Sha59], we seek a mapping bound by a distortion constraint from a random variable (r.v.) X to an r.v. U, minimizing the bitrate of the outcomes of X. More formally, given a distortion measure $\mathcal{D}$ and a distortion constraint D_c, the minimal bitrate is:

$$\min_{P_{U|X}} I(X;U) \text{ s.t. } \mathcal{D}(X,U) \leq D_c \tag{4.1}$$

where $I(X;U)$ is the mutual information and is defined as

$$\mathrm{I}(X;U) = \int\int p(x,u) \log \left(\frac{p(x,u)}{p(x)p(u)} \right) dxdu \tag{4.2}$$

In lossy image compression, U is typically the reconstruction of $\tilde{X}$ of the original input, and the distortion measure is some sum of squared errors $d(x,\tilde{x})$. We may use Equation (4.1) to implement an objective that enforces the integrity of *any* task, irrespective of whether it is image reconstruction or prediction performance on a particular dataset. With semantically lossless, we refer to the solution quality that meets the client-application requirements. It poses a threshold, so we seek the lowest possible rate during optimization.

From Deep to Shallow Bottlenecks

When the task is to predict the ground-truth labels Y from a joint distribution $P_{X,Y}$, the r-d objective is essentially the information bottleneck principle [TPB00]. By relaxing

the Equation (4.1) with a lagrangian multiplier, the objective is to maximize:

$$I(Z;Y) - \beta I(Z;X) \tag{4.3}$$

Specifically, an encoding Z should be a minimal sufficient statistic of X respective Y, i.e., we want Z to contain relevant information regarding Y while discarding irrelevant information from X. Practical implementations differ by the target task and how they approximate Equation (4.3). For example, an approximation of $I(Z;Y)$ for an arbitrary classification task, the conditional cross entropy (CE) [YMT23]:

$$\mathcal{D} = H(P_Y, P_{\tilde{Y}|Z}) \tag{4.4}$$

where $\tilde{Y}$ is the prediction based on the compressed representation Z. Using Equation (4.4) for estimating $I(Z;Y)$ to end-to-end optimize a neural compression model is not a novel idea (Section 4.1.2). However, in such work, the latent variable is typically the final representation of a large backbone, which we refer to as *Deep Variational Information Bottleneck Injection (DVBI)*. Conversely, we work with resource-constrained clients. To design lightweight encoders, we shift the bottleneck to the shallow layers, which we refer to as *Shallow Variational Bottleneck Injection (SVBI)*.

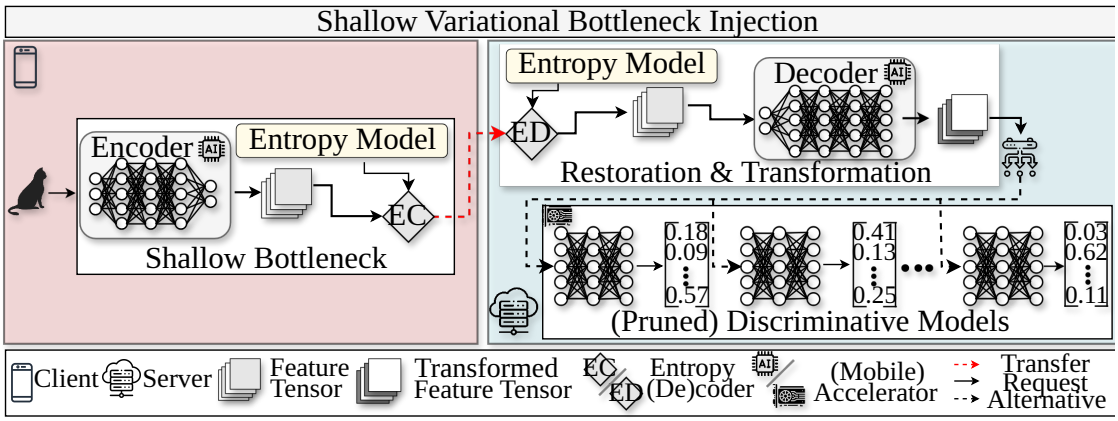


Figure 4.4: Prediction with Variational Bottleneck Injection

Figure 4.4 sketches the proposed approach at a high level. There are two elemental distinctions to SC methods. First, the prediction model is not split between the client and the server. Instead, SVBI separates the concern for learning a representation with sufficient information and learning to map the representation to a client-requested model architecture. A decoder restores and transforms the compressed representation to a backbone that may accommodate multiple tasks. Hence, operators can deploy a single lightweight encoder. Server-side decoders replace the shallow layers of a backbone, so the prediction model is split within the server. The encoder is decoupled from a particular task and the decoder-backbone pair. Second, compared to split runtimes, the decision to apply the compression model may only depend on *internal conditions*. It can decouple

the client from any external component, and applying the encoder should always be preferable if a mobile device has the minimal required resources. Since our method does not fine-tune weights, we do not need to maintain additional models to accommodate clients who cannot apply the encoder and can simply route the image tensor to the input layer instead of skipping them.

The properties permit server-side transparency and minimize the operator overhead, as demanded in Section 1.2, Section 4.2.1 will elaborate on the technical details of how this separation permits one encoder instance to accommodate multiple decoder-backbone pairs. We first derive the optimization objective of SVBI.

Shallow Bottlenecks with Head Distillation

The existing objectives for DVBI could generalize to SVBI, for example, by estimating the distortion term with Equation (4.4) as done in [AMT+17]. The downside is that it assumes access to the original labeled dataset. We may substitute hard labels Y distillation, using the soft labels $Y_{\mathcal{T}}$ (dark knowledge [HVD15]) of the prediction model we inject the bottleneck in. Still, we will demonstrate empirically in Section 4.2.2 that neither hard labels nor distillation with soft labels yield a meaningful reduction in rate.

SVBI requires an objective function that is simple enough for low-capacity encoders to learn while significantly reducing latent entropy without adverse effects for downstream tasks. We achieve this by training the compression model to predict the output of the *shallow layers* H of the prediction model instead of hard or soft labels, so the objective becomes maximizing

$$I(Z;H) - \beta\, I(Z;X). \tag{4.5}$$

Loss functions using the output of the shallow layers as the reconstruction target, optimize for a Z that is a minimal sufficient statistic of X with respect to H.

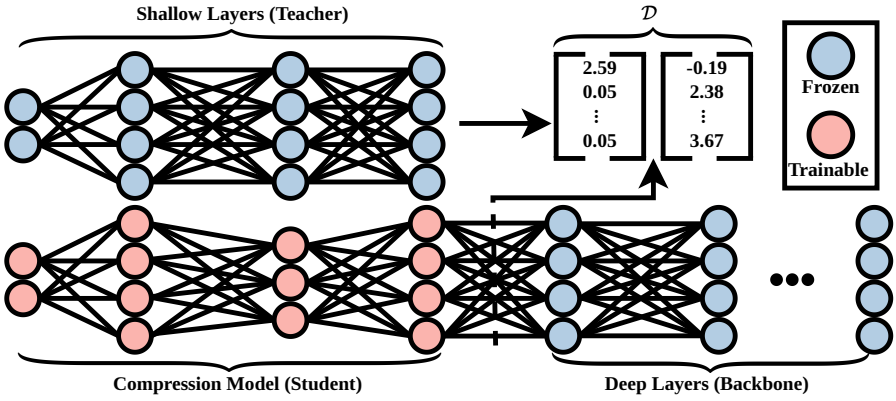


Figure 4.5: Head Distillation

Figure 4.5 illustrate the idea. With faithful replication of H, the partially modified ANN has an information path that approximates the unmodified version accurately. A sufficient

statistic retains the information necessary to replicate the input for a deterministic tail, so the final prediction does not change.

Lastly, we emphasize that any rate reduction for a set distortion constraint does *not* come from targeting the ANN inference instead of human perception. It is the *specificity* of the task that determines the lower-bound on a distortion constraint that is semantically lossless. Shallow features represent less specific information than deep features, and therefore, there is less potential for reducing the rate of semantically lossless prediction. In return, a compressed representation of shallow features will generalize more seamlessly. Under reasonable conditions, there is no meaningful relationship between how the data is represented and the rate-distortion performance. Hence, we are *not* gaining any efficiency with an objective that uses the syntax of a particular architectural style's latent tensor, instead of pixel values aligned in a two-dimensional grid. Chapter 5 will elaborate on the seemingly pedantic distinction when discussing image reconstruction from feature tensors. We first design a practical method for MEC using SVBI to empirically demonstrate that it can meet the demands set in Section 1.2.

4.2 Neural Feature Compression for Mobile Edge Computing

This section describes an implementation of an SVBI, which we named *FrankenSplit*. FrankenSplit is intended for MEC, meaning that it should reduce request latency by reducing transmission costs. Operators may deploy *a single* encoder for devices of various applications, and support arbitrary ANN architectures for high-level vision tasks. We extensively evaluate FrankenSplit and compare it against a wide range of competitive baselines. At the time of writing, the core methodology of FrankenSplit may be considered the state-of-the-art in artificial bottleneck injection for MEC, with some recent work starting to outperform it, but only in isolated use cases [ZC25].

4.2.1 Solution Approach

While Head Distillation has been proposed in the context of Split Computing [MBC+19, MYLM22, SSTM21], such methods have the same limitations discussed in Section 4.1.3. Their loss relies on signals from deeper layers for convergences, so the shallow layers only serve as hints [RBK+15, WY22]. Additionally, they require a second training stage that fine-tunes the deeper layers of the target model. This results in unstable training, modest rate reductions, and their adoption would incur the operational overhead that should be avoided.

Besides showing Section 4.2.2 that pure HD can significantly outperform existing approaches *without* fine-tuning deeper layers with an adequate decoder design, we show how to distill "softer signals" from deeper layers. Unlike in previous approaches, such signals are not crutches required for the bottleneck to converge, but further improve rate-distortion performance.

Loss Function for End-to-end Optimization

The compression algorithm follows the NTC framework [BCM+20], embeds a variational autoencoder, and assumes a factorized prior (FP) as in [BLS17]. For an image vector x, we have a parametric analysis transform $g_a(x;\phi_g)$ that maps x to a latent vector z. Then, a quantizer Q discretizes z to $\bar{z}$, such that an entropy coder can use the entropy model to losslessly compress $\bar{z}$ to a sequence of bits. Different from NIC, a parametric synthesis transform parametric synthesis transforms $g_s(\bar{z};\theta_g)$ does not map $\bar{z}$ to a reconstruction of the input $\tilde{x}$, but to an approximation of shallow representation $\tilde{h}$ distilled from a teacher.

Analogous to variational inference, we approximate the intractable posterior $p(\tilde{z}|x)$ with a parametric variational density $q(\tilde{z}|x)$, excluding constants, as follows:

$$\mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{x}}} D_{\mathrm{KL}}\left[q \| p_{\tilde{\boldsymbol{z}}|\boldsymbol{x}}\right] = \mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{x}}}\mathbb{E}_{\tilde{\boldsymbol{z}}\sim q}[\underbrace{-\log p(x|\tilde{z})}_{\text{distortion}} - \overbrace{\log p(\tilde{z})}^{\text{weighted rate}}\ ] \tag{4.6}$$

The distortion term is given by

$$P_{x|\tilde{z}}(x \mid \tilde{z}, \theta_g) = \mathcal{N}(x \mid g_s(\tilde{z};\theta_g),\ \mathbf{1}) \tag{4.7}$$

which we implement as the square sum of differences between h and $\tilde{h}$ as our distortion loss.

The rate term describes the cost of compressing $\tilde{z}$. We apply uniform quantization $\bar{z} = \lceil \tilde{z} \rfloor$. Since discretization leads to problems with the gradient flow, we apply a continuous relaxation by adding uniform noise $\eta \sim \mathcal{U}(-\frac{1}{2}, \frac{1}{2})$. Combining the rate and distortion term, we derive the loss function for estimating objective Equation (4.5) as:

$$\mathcal{L} = \|\mathcal{P}_h(x) \text{ - } (g_s(g_a(x;\phi_g) + \eta;\theta_g)\|_2^2 + \beta \log(g_a(x;\theta_g) + \eta) \tag{4.8}$$

Note that FP is a strong assumption, and there is no shortage of sophisticated methods that can further improve rate-distortion performance, such as with side information, considered in Chapter 5. We chose the simplest model since the empirical analysis focuses on gains from SVBI, and replacing FP with any more recent methods is straightforward. The following shows how to improve the semantic rate-distortion performance without introducing additional components native to NIC.

Saliency Guided Distortion

As discussed in Section 4.1.3, we may trade off task specificity with rate. Sufficiently approximating shallow layers as a distortion constraint may generalize, but it may be needlessly strict. The challenge is to distill knowledge from deeper layers, but only as soft signals for regularization that reduce the penalty of not accurately predicting non-salient pixels. For each sample, we require a vector $\mathcal{S}$, where each $s_i \in \mathcal{S}$ is a weight term for a spatial location salient about the conditional probability distributions of the remaining

tail layers. Then, we should be able to improve the r-d performance by regularizing the distortion term in Equation (4.8) with

$$\mathcal{L}_{\text{distortion}} = \gamma_2 \cdot \mathcal{L}_1 + \gamma_2 \cdot s_i \cdot \frac{1}{N} \sum_i (h_i - \tilde{h}_i)^2 \tag{4.9}$$

Where $\mathcal{L}_1$ is the distortion term from Equation (4.8), and γ_1, γ_2 are nonnegative real numbers summing to 1, and may be used to tune the regularization from the saliency guidance. We default to $\gamma_1 = \gamma_2 = \frac{1}{2}$ in our experiments, as we refrain from optimizing performance through dataset-specific adjustments. Figure 4.6 describes our final training setup. Note

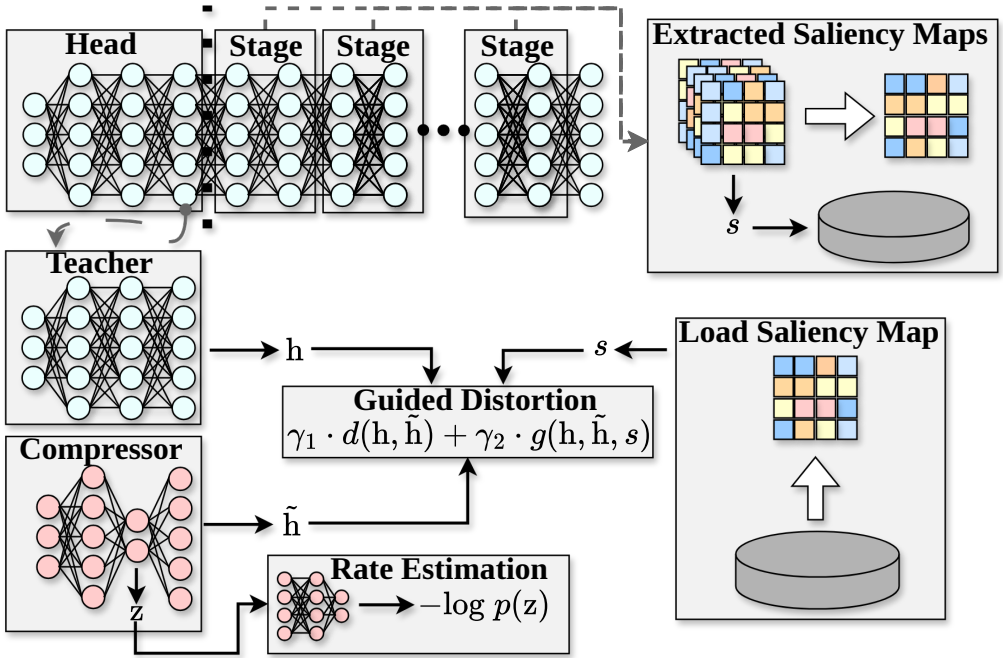


Figure 4.6: Training setup

that we only require computing the saliency maps once, and they are architecturally agnostic towards the encoder. We compute saliency maps using *class activation mapping (CAM)* [ZKL+16]. Their intended purpose to improve the explainability of ANNs is to summarize salient pixel locations, which we repurpose to regularize the reconstruction loss. Specifically, we use Grad-CAM [SCD+19] to measure a spatial location's importance at any stage.

The advantage of Grad-CAM is its architecture-agnosticism and computational efficiency. Mixing with guided backpropagation [SDBR15] could refine the resulting saliency maps with finer-grained feature importance scaling. However, guided backpropagation relies on specific properties of the activation function and requires adjustments for each architectural family. Figure 4.7 illustrates some examples of saliency maps when averaged over the deeper backbone stages.

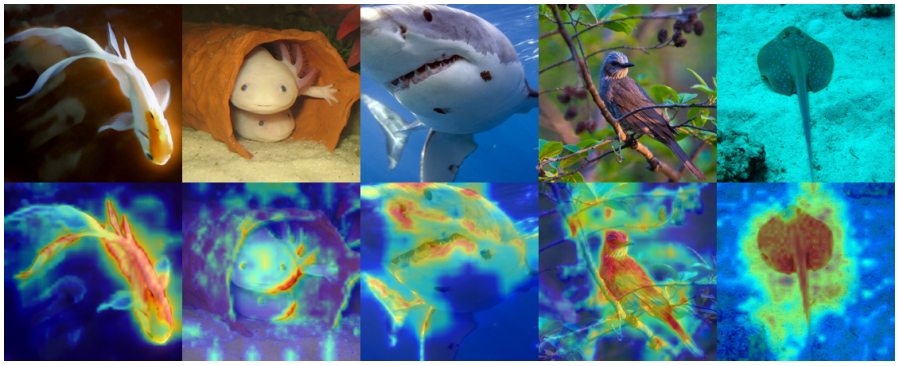


Figure 4.7: Extracted saliency maps using Grad-CAM

Network Architecture

The beginning of this section broke down our aim into three problems. We addressed the first with SVBI and proposed a novel training method for low-capacity compression models. A generalizable resource-asymmetry-aware autoencoder design remains. Additionally, the encoder should be reusable for several backbones. To not inflate the significance of our contribution, we refrain from including components based on existing work in efficient neural network design.

Model Taxonomy We introduce a minimal taxonomy described in Figure 4.8 for our approach. The top-level, *Archtype*, reflects the primary inductive bias of the model. *Architectural families* describe variants (e.g., ResNets such as ResNet [HZRS16], Wide ResNet [ZK16], ResNeXt [XGD+17], etc.). *Directly related* refers to the same architecture of different sizes (e.g., Swin-T, Swin-S, Swin-B, etc.). The challenge is to conceive a

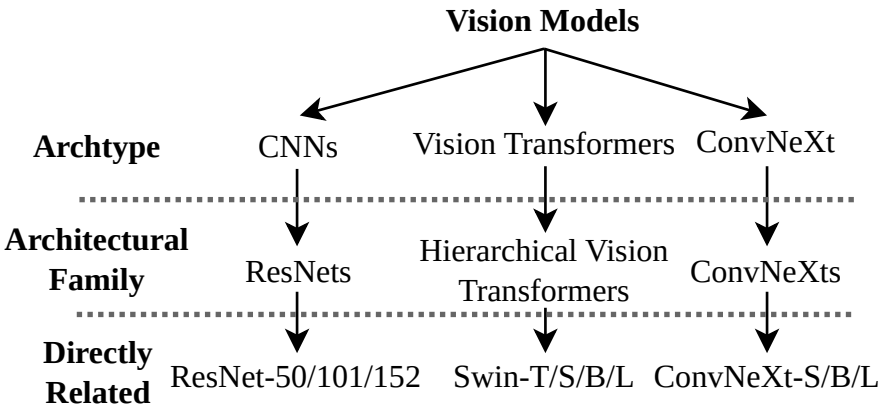


Figure 4.8: Simple taxonomy with minimal example

design heuristic that can exploit the available server resources to aid the lightweight encoder with minimal overhead on the prediction task. First, we concretize shallow

features by describing how to locate the layers for bottleneck placement. Then, we derive the heuristic to conceive decoder models for arbitrary architectural families and how to account for client-server resource asymmetry.

Lastly, we describe how to share trained compressor components among directly related architectures.

Bottleneck Location by Stage Depth Consider how most modern ANNs consist of an initial embedding followed by a few stages (Described in Section 4.1.3). Within directly related architectures, the individual components are identical. The difference between variants is primarily the embed dimensions or the block ratio of the deepest stage. For example, the block ratio of ResNet-50 is 3:4:**6**:3, while the block ratio of ResNet-101 is 3:4:**23**:3. Consequently, the stage-wise organization of models defines a natural interface for SVBI. For the remainder of this work, we refer to the *shallow layers* as the layers before the deepest stage (i.e., the initial embedding and the first two stages).

Decoder Blueprints A key characteristic distinguishing archetypes is the inductive bias introduced by basic building blocks (e.g., convolutions versus attention layers). To consider the varying representations among non-related architectures, we should not disregard architecture-induced bias by directly repurposing neural compression models for SC. For example, a scaled-down version of Ballé et al.'s [BLS17] convolutional neural compression model can yield strong r-d performance for bottlenecks reconstructing a convolutional layer [MYLM23]. However, we will show that this does not generalize to other architectural families, such as hierarchical vision transformers [LLC+21].

One potential solution is to use identical components for the compression model from a target network. While this may be inconsequential for server-side decoders, it is inadequate for encoders due to the heterogeneity of edge devices. Vendors have varying support for the basic building blocks of a ANN, and particular operations may be prohibitively expensive for the client. Hence, in FrankenSplit, the encoder is fixed, but the decoder is adaptable. Regardless of the decoder architecture, we account for the heterogeneity with a uniform encoder architecture composed of three downsampling residual blocks of two stacked 3×3 convolutions with ReLU non-linearity, totaling around $140,000$ parameters. We handle the varying representations by introducing *decoder blueprints* tailored towards an architectural family, i.e., one blueprint corresponds to all directly related architectures.

Figure 4.9 illustrates a reference implementation of FrankenSplit post-training with two blueprints applied to two variants. Creating blueprints is required only once for an architectural family. Boxes within the gray areas are separate instances (i.e., only one encoder), and boxes with the same name share an architecture. The rounded boxes outside organize layer views from coarse to fine-grained. We elaborate on how a single encoder can accommodate multiple decoder-backbone pairs in Section 4.2.1. The numbers in the parentheses refer to stage depth. Since the backbones are foundational models

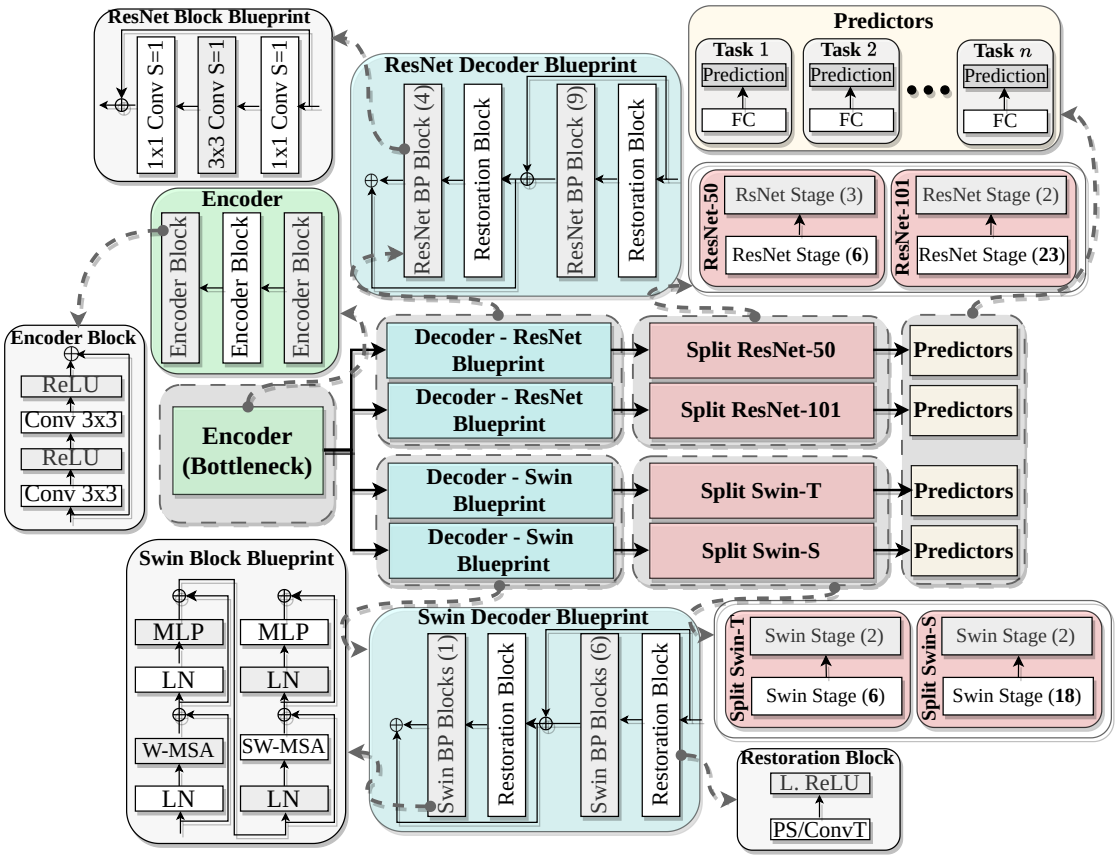


Figure 4.9: Reference implementation of FrankenSplit

extensively trained on large datasets, we can naturally accommodate several downstream tasks by attaching separately trained predictors.

Blueprint instances replace a backbone's first two stages (i.e., the shallow layers) with two blueprint stages, taking a compressed representation as input instead of the original sample. The work by Liang et al. [LCS+21] inspires our approach to treat decoding as a restoration problem. Each stage comprises a restoration block and several blueprint (transformation) blocks, followed by a residual connection. The idea is to separate restoration (i.e., upsampling, ''smoothing" quantized features) from transformation (i.e., matching the target representation regardless of encoder architecture). The restoration block is agnostic regarding the target architecture and optionally upsamples. The blueprint blocks induce the same bias as the target architectural family.

Two distinctions exist between the original blocks and their corresponding blueprint (transformation). First, the latter modifies operations not to reduce the latent spatial dimensions. Second, the embedding layer dimensions and stage depths may differ to reflect the resource asymmetry commonly found in MEC.

Although we should consider the resource asymmetry between the client and the server (i.e., by allocating more parameters to the decoder), there are limitations. Learning a function that can accurately retain necessary information is limited by the encoder's capacity (Section 4.1.4). Still, when end-to-end optimizing the compression model, it can benefit from increasing the decoder's capacity for restoration with diminishing returns.

Intuitively, we implement blueprints that result in decoder instances with, at most, the same execution time as the head of a target backbone. As a reminder, unlike most work in SC, we advocate keeping the execution time roughly equal on the server rather than reducing it. The encoder's responsibility is not to minimize the server load by executing shallow backbone layers. FrankenSplit treats the encoder entirely separate from the backbone. Besides dedicating the encoder exclusively to reducing transfer size, this separation of concern is necessary to accommodate several backbones with a single encoder instance.

Encoder Re-Usability We argue that the representation of shallow layers generalizes well enough that it is possible to reuse compressor components. Consider the experiment

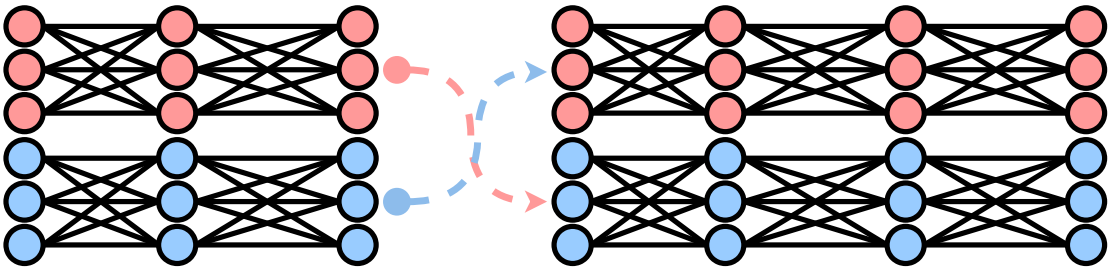


Figure 4.10: Routing head outputs to different tails

illustrated in Figure 4.10, where we split several backbones into head and tail models. The backbones are off-the-shelf models from *torch image models* (timm) [Wig19] and pre-trained on the ImageNet [RDS+15] dataset. The head models consist of the initial embedding and shallow layers, i.e., the first two stages. The remaining layers comprise the substantially larger tails (roughly $2-5\%$ of total model parameters).

Then, we freeze the tail parameters and route the head output to all non-corresponding tails (e.g., ConvNeXt-T to Swin-T/S/B) and measure the accuracy every few iterations with a batch size of 128 as we finetune the head parameters using cross entropy loss. Each head-tail pair is a separate model built by attaching a copy of the head from one architecture to the tail of another. Where dimensions between head and tail pairs do not match, we add a single 1×1 convolutional layer.

Figure 4.11 shows how rerouting the input between head models first (0 iterations) results in near 0% accuracy across all head-tail pairs. However, the concatenated models quickly converge near their original accuracy (roughly $80-83\%$) within just a few iterations (10100 iterations with 128 samples corresponding to one epoch on the ImagNet dataset).

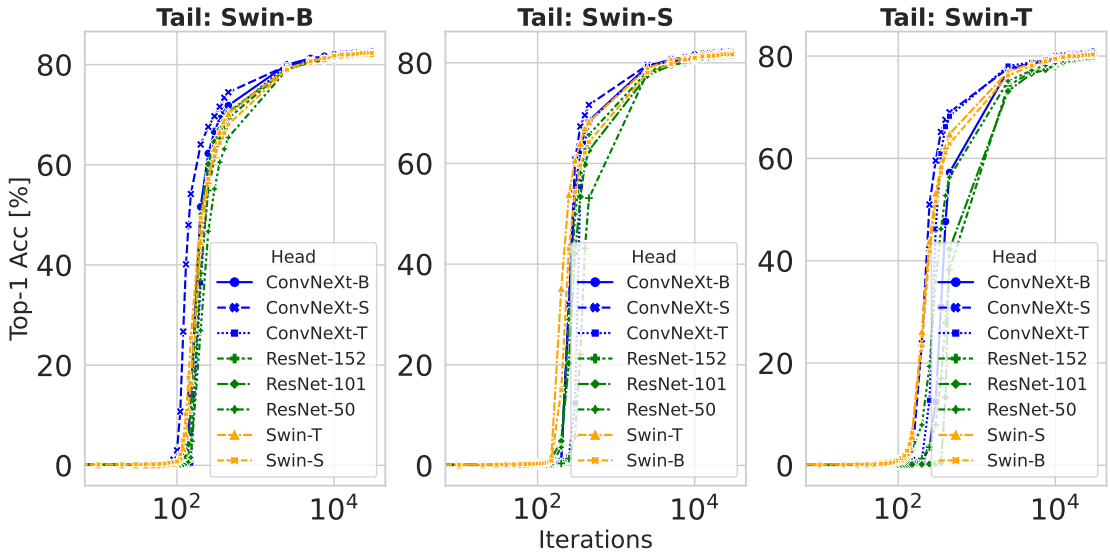


Figure 4.11: Recovering Top-1 accuracy of rerouted heads

Notice that this holds regardless of whether the head-tail pairs are directly related to the modified network.

Therefore, if a compressor can sufficiently approximate the representation of just one head (i.e., the shallow layers of a network), it should be possible to accommodate arbitrary tails (i.e., the deeper layers of a network).

Crucially, applying the distortion measure in Equation (4.8) or Equation (4.9) does not result in an inherently different encoder behavior. Like training the compression model with a distortion measure from NIC, the purpose of the encoder is to reduce uncertainty by decorrelating the data and discarding information. The distortion measure only controls what information an encoder should prioritize. Regardless of the target backbone's architecture, the encoder should decorrelate the input to reduce uncertainty. Conversely, the decoder seeks a mapping to the backbone's representation. In other words, if we can map the latent to one representation, we can map it to any other with comparable information content. We can freeze the encoder and train various decoders to support arbitrary architectures once we train one compression model with a particular teacher as described in Figure 4.6. The blueprints facilitate an efficient transformation from the encoder's compressed representation to an input suitable for a particular backbone.

In summary, the high-level procedure is to take a copy of $P_{\mathcal{T}}$. Mark the location of the bottleneck by separating the copy into a head $\mathcal{P}_h$ and a tail $\mathcal{P}_t$. Since both parts are deterministic, or every realization of r.v. X there is a representation $\mathcal{P}_h(x) = h$ such that $\mathcal{P}_{\mathcal{T}}(x) = \mathcal{P}_h(\mathcal{P}_t(x))$. Then, replace the head with an autoencoder and a parametric entropy model. The encoder is deployed at the sender, the decoder at the receiver, and the entropy model is shared. Instantiate decoders using the corresponding blueprint, and train a separate instance for each variant. Notice that this method keeps the encoder

parameters frozen, permitting us to deploy a single set of weights across all clients. After deployment, splitting is replaced with rerouting the input to a layer index (Section 4.2.1) since the original weights remain unmodified.

4.2.2 Evaluation

Training & Implementation Details

We optimize our compression models initially on the 1.28 million ImageNet [RDS+15] training samples for 15 epochs, as described in section 4.2.1 and section 4.2.1, with some slight practical modifications for stable training. We aim to minimize bitrate without sacrificing predictive strength. Hence, we first seek the lowest β resulting in lossless prediction. We use Adam optimization [KB14] with a batch size of 16 and start with an initial learning rate of $1 \cdot 10^{-3}$, then gradually lower it to $1 \cdot 10^{-6}$ with an exponential scheduler. Methods are implemented using PyTorch [AYH+24], CompressAI [BRFP20] for entropy estimation and entropy coding, and pre-trained backbones from timm [Wig19]. All baseline implementations and weights were either taken from CompressAI or the official repository of a baseline. To compute the saliency maps, we use a modified *XGradCAM* method from the library in [Gc21] and include necessary patches in the accompanying repository. Lastly, we use torchdistill [Mat21] to configure experiments and for reproducibility.

Experiment Setting

The experiments aim to empirically demonstrate that SVBI can enable latency-sensitive and performance-critical applications. The basic scenario is that a mobile client requires access to a remotely deployed ANN. The metrics measure the semantic rate-distortion performance, i.e., whether it can achieve a substantially lower rate without loss in prediction quality, and request times in various channels. As in Section 3.2.3, since it is not feasible to exhaustively evaluate all existing ANN architectures, we focus on three well-known architectural representatives and some of their variants.

ResNet [HZRS16] for classic residual CNNs, Swin Transformer [LLC+21] for hierarchical vision transformers, which are receiving increasing adoption, and ConvNeXt [LMW+22] for modernized state-of-the-art CNNs. Table 4.2 summarizes the relevant characteristics of the unmodified backbones subject to our experiments.

Baselines Since our work aligns closest to learned image compression, we extensively compare FrankenSplit with learned and handcrafted codecs applied to the input images, i.e., the input to the backbone is the distorted output. Comparing task-specific methods to general-purpose image compression methods may seem unfair. However, FrankenSplit's universal encoder has up to 260x less trainable parameters and further reduces overhead by not including side information or a sequential context model.

Table 4.2: Overview of Backbone Performance on Server

Backbone	Ratios	Params	Inference (ms)	Top-1 Acc. (%)
Swin-T	2:2:6:2	28.33M	4.77	81.93
Swin-S	2:2:18:2	49.74M	8.95	83.46
Swin-B	2:2:30:2	71.13M	13.14	83.88
ConvNeXt-T	3:3:9:3	28.59M	5.12	82.70
ConvNeXt-S	3:3:27:3	50.22M	5.65	83.71
ConvNeXt-B	3:3:27:3	88.59M	6.09	84.43
ResNet-50	3:4:6:3	25.56M	5.17	80.10
ResNet-101	3:4:23:3	44.55M	10.17	81.91
ResNet-152	3:8:36:3	60.19M	15.18	82.54

The naming convention for the learned baselines is the first author's name, followed by the entropy model. Specifically, we choose the work by Ballé et al. [BLS17, BMS$^+$18] and Minnen et al. [MBT18] for NIC methods since they represent foundational milestones. Complementary, we include the work by Cheng et al. [CSTK20] to demonstrate improvements with architectural enhancement.

As the representative for disregarding autoencoder size to achieve state-of-the-art r-d performance in LIC, we chose the work by Chen et al. [CLM$^+$21] Their method differs from other LIC baselines by using a partially parallelizable context model, which trades off compression rate with execution time according to the configurable block size. We refer to such context models as Blocked Joint Hierarchical Priors and Autoregressive (BJHAP). Due to the large autoencoder, we found evaluating the inference time on constrained devices impractical when the context model is purely sequential, and set the block size to 64x64. Additionally, we include the work by Lu et al. [LGS$^+$22] as a milestone of the recent effort on efficient LIC with reduced autoencoders, but only for latency-related experiments since we do not have access to the trained weights.

As a baseline for the state-of-the-art SC, we include the Entropic Student (ES) [MYLM23, MYLM22]. The ES demonstrates the performance of directly applying a minimally adjusted LIC method for feature compression. One caveat is that we intend to show how FrankenSplit generalizes beyond CNN backbones, despite the encoder's simplistic CNN architecture. Although Matsubara et al. evaluate the ES on a wide range of backbones, most have no lossless configurations. Nevertheless, comparing bottleneck injection methods using different backbones is fair, as we found that the choice does not significantly impact the r-d performance (Section 4.2.2). Therefore, for an intuitive comparison, we choose ES with ResNet-50 using the same factorized prior entropy model as FrankenSplit. We separate the experiments into two categories to assess whether our proposed method addresses the abovementioned problems.

Criteria rate-distortion performance We measure the bitrate in bits per pixel (bpp) because it permits directly comparing models with different input sizes. Choosing a distortion measure to draw meaningful and honest comparisons is challenging for feature compression.

Unlike evaluating reconstruction fidelity for image compression, PSNR or MS-SSIM does not provide intuitive results regarding predictive strength. Similarly, reporting absolute values (e.g., top-1 accuracy) gives an unfair advantage to experiments conducted on higher capacity backbones and veils the efficacy of a proposed method.

Hence, for a transparent evaluation, we determine the adversarial effects of codecs with image classification since it provides an unambiguous performance metric with established benchmark datasets. Specifically, we evaluate the distortion with the relative measure *predictive loss*, i.e., the drop in top-1 accuracy incurred by codecs. In particular, for SVBI methods, (near) lossless prediction implies that the reconstruction is a sufficient approximation for shallow features of an arbitrary feature extractor.

To ensure a fair comparison, we give the LIC and handcrafted baselines a grace threshold of 1.0% top-1 accuracy, to account for mitigating predictive loss incurred by codec artifacts [LTY+21]. For FrankenSplit, we set the threshold at 0.4%, reflecting the configuration with the lowest predictive loss of the ES. Note that, unlike the ES, FrankenSplit does not rely on fine-tuning the tail parameters of a backbone to improve r-d performance.

Measuring latency and overhead To account for the resource asymmetry in MEC, we use NVIDIA Jetson boards[1] for representing capable but resource-constrained mobile clients. Contrastingly, the server hosts a powerful GPU. Table 4.3 summarizes the hardware we use in our experiments.

Table 4.3: Clients and Server Hardware Configuration

Device	Arch	CPU	GPU
Server	x86	16x Ryzen @ 3.4 GHz	RTX 3090
Client (TX2)	arm64x8	4x Cortex @ 2 GHz	Vol. 48 TC
Client (NX)	arm64x8	4x Cortex @ 2 GHz	Pas. 256 CC

Rate-Distortion Performance

We measure the predictive loss by the drop in top-1 accuracy from Table 4.2 using the ImageNet validation set for the standard classification task with 1000 categories. Analogously, we measure filesizes of the entropy-coded binaries to calculate the average bpp. To demonstrate that we can accommodate a non-CNN backbone with a CNN encoder, we start with a Swin-B implementation of FrankenSplit. Figure 4.12 shows r-d curves with the Swin-B backbone. The architecture of FrankenSplit-FP (FS-FP)

[1] nvidia.com/en-gb/autonomous-machines/embedded-systems/

and FrankenSplit-SGFP (FS-SGFP) are identical. We train both models with the loss functions derived in Section 4.2.1. The difference is that FS-SGFP is saliency-guided, i.e., FS-FP represents the pure HD training method and is an ablation to the saliency-guided distortion.

Effect of Saliency Guidance Although FS-FP performs better than almost all other models, it is trained with the suboptimal objective discussed in Section 4.1.4. We identified the issue as overly skewing the objective needlessly towards the distortion term. Consequently, we proposed regularizing the distortion term by applying extracted saliency maps in Section 4.2.1 to improve the r-d performance. We favor Grad-CAM to compute the saliency maps over comparable methods for two reasons. First, it is generically applicable to arbitrary vision models. Second, it does not introduce additional tunable hyperparameters. The suboptimality of the unregularized objective is demonstrated by FS-SGFP outperforming FS-FP. By simply guiding the distortion loss with saliency maps, we achieve a 25% lower bitrate without impacting predictive strength or additional runtime overhead.

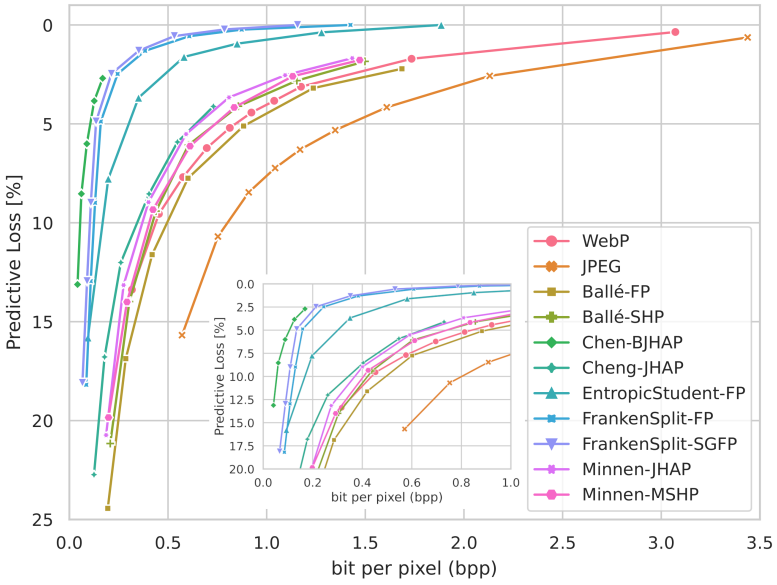


Figure 4.12: Rate-distortion curve for ImageNet

Comparison to the ES Even without saliency guidance, FS-FP consistently outperforms ES by a large margin. Specifically, FS-FP and FS-SGFP achieve 32% and 63% lower bitrates for the lossless configuration.

We ensured that our bottleneck injection incurs comparable overhead for a direct comparison to the ES. Moreover, the ES has an advantage due to fine-tuning tail parameters in an auxiliary training stage. Therefore, we attribute the performance gain to the more sophisticated architectural design decisions described in Section 4.2.1.

Comparison to Image Codecs For almost all lossy codec baselines, Figure 4.12 illustrates that FS-(SG)FP has a significantly better r-d performance. Comparing FS-FP to Ballé-FP demonstrates the r-d gain of task-specific compression over general-purpose image compression. Although the encoder of FrankenSplit has 25x fewer parameters, both codecs use an FP entropy model with encoders consisting of convolutional layers. Yet, the average file size of FS-FP with a predictive loss of around 5% is 7x less than the average file size of Ballé-FP with comparable predictive loss.

FrankenSplit also beats modern general-purpose LIC without including any of their heavyweight components. The only baseline FrankenSplit does not convincingly outperform is Chen-BJHAP. Nevertheless, in Section 4.1.3, we demonstrate that the incurred overhead offsets the compression gain disproportionately.

Image Codec Incurred Predictive Loss For clarity, we separately evaluate r-d performance on the other backbones listed in Table 4.2 for FrankenSplit and baseline codecs.

Earlier, we argued that measuring PSNR is unsuitable for assessing effects on downstream prediction. Since the image codecs are entirely decoupled from the predictive task, the bitrate is identical regardless of the backbone. We use this opportunity to plot PSNR instead of bpp against predictive loss in Figure 4.13.

Considering that compression models aggressively discard information, it is intuitive that the predictive loss is comparable across backbones. While some models handle distorted samples better, the difference in predictive loss is at most 3-5%. Still, the discrepancy demonstrates that PSNR is not a suitable measure for downstream tasks even within the same codec. More importantly, the discrepancy across baselines is considerably wider. For example, it is around 10% between Minnen-MSHP and Chen-BJHAP for lower PSNR levels.

Blueprints Generalization to Arbitrary Backbones We now evaluate the r-d performance of other implementations of FrankenSplit to determine whether the blueprint heuristics generalize to arbitrary architectures. We create a decoder blueprint (Section 4.2.1) for each of the three architectural families (Swin, ResNet, and ConvNeXt). Then, we perform bottleneck injection at the layers before the deepest stage (Section 4.2.1), Figure 4.14 plots r-d performance of directly related architectures sharing the corresponding blueprint but with separately trained compressors. All models are trained as described in Figure 4.6. Across all architectural families, we observe similar r-d performance. The (near) lossless configurations of the largest backbones (Swin-B, ConvNeXt-B, ResNet-152)

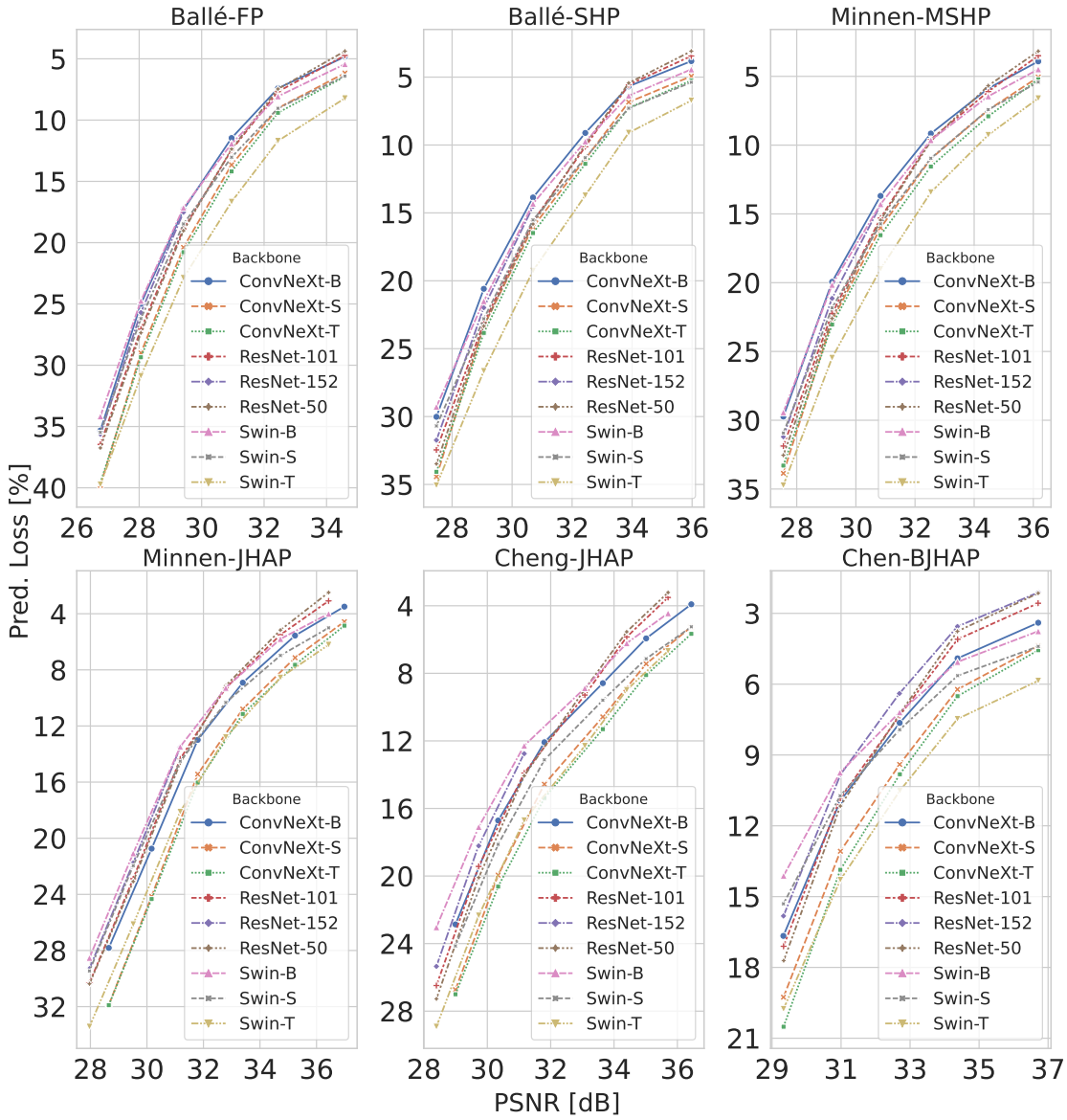


Figure 4.13: Predictive Loss of baselines on multiple Backbones

require around the same bpp, whereas smaller models tend to require 3-4% more bpp for comparable predictive loss.

Next, we conduct experiments to determine the importance of finding an adequate blueprint but assigning mismatching instances to a backbone. Table 4.4 summarizes the results for the largest backbones with varying decoder sizes. The Swin blueprint for the Swin-B decoder results in the FrankenSplit implementation from FS-FP from Figure 4.12. With 1% overhead in parameters, the compressor achieves 5.08 kB for 0.40% predictive

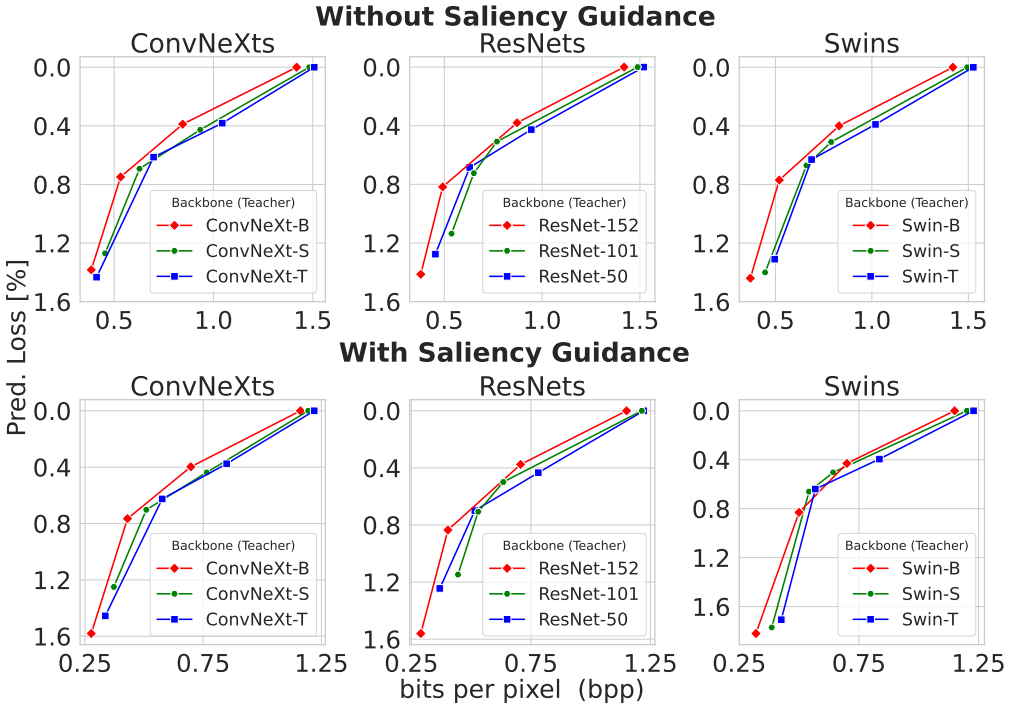


Figure 4.14: Rate-distortion curve for various backbones

loss. However, once we train compressors with ResNet or ConvNeXt restoration blocks, the r-d performance for the Swin-B is significantly worse when overhead is roughly equal. A blueprint that performs well for its intended target architecture results in substantially

Table 4.4: Effect of Mismatching Blueprints

Blueprint-Backbone	Params Overhead (%)	File Size (kB)	Pred. Loss (%)
ConvNext-Swin	0.96	19.05	2.49
		15.07	3.00
	2.86	14.46	2.53
		11.37	2.74
	5.89	12.53	1.36
		10.08	2.09
ResNet-Swim	1.03	22.54	0.82
		18.19	0.99
	2.73	16.32	0.81
		12.68	0.98
	5.25	13.89	0.79
		10.01	0.98

worse r-d performance for other architecture. Only increasing the decoder size brings the r-d performance closer to configurations that apply the appropriate blueprint.

From our findings, we can draw several conclusions. The r-d performance regarding the backbone network is near-agnostic. The implication is that the information content of the teachers (i.e., shallow layers) of varying architectures is comparable. We explain this by considering that we select the shallow layers as all layers preceding the deepest stage, which have comparable parameters across varying architectures.

Additionally, choosing a decoder architecture with the correct inductive bias (i.e., a blueprint) can transform compressed features significantly more efficiently.

Single Encoder with Multiple Backbones We conduct a similar experiment as head rerouting from Section 4.2.1. However, we finetune the decoders instead of the head models.

We first select the compressors with (a near) lossless prediction from Figure 4.14 for each architectural family. Then, we choose the encoder from one of the compressors corresponding to the largest variants. Finally, we attach the decoders from the other compressors and finetune their parameters. We use unweighted head distillation and cross entropy (between the backbone classifier outputs and the hard labels) as the loss function. Analogous to the experiment in Section 4.2.1, we set the batch size as 128 and use PyTorch's Adam optimizer with a learning rate of $7 \cdot 10^{-5}$. To demonstrate the

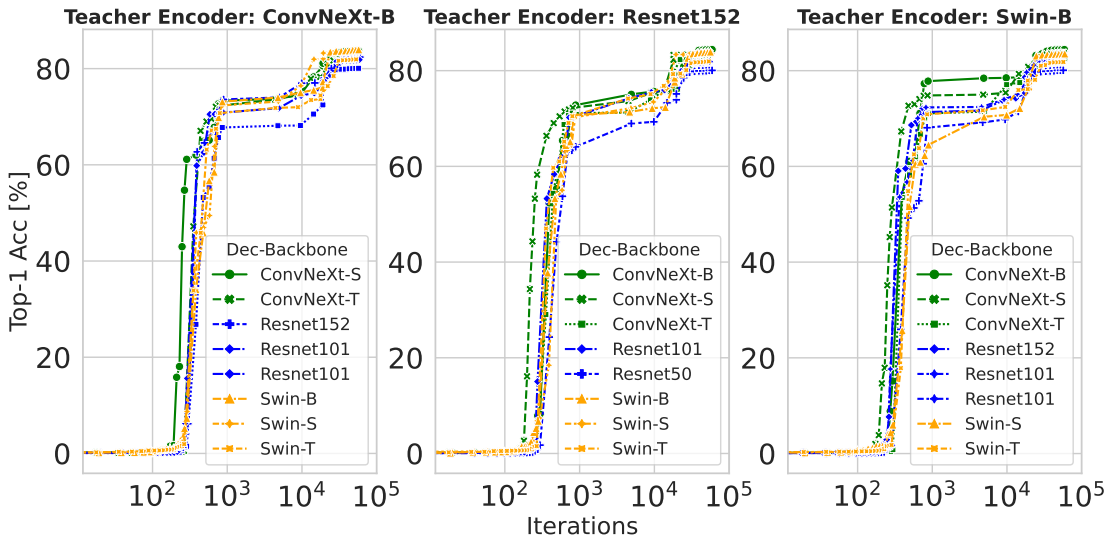


Figure 4.15: Iterations to recover accuracy with decoder

limited importance of the initial teacher, we repeat this process for each of the three encoders separately and summarize the results in Figure 4.15. Note that the bitrate does not change due to freezing the encoder parameters. Hence, we report iterations until accuracy is restored to exemplify the similarity to the rerouting experiment in Section 4.2.1. We consider an accuracy restored if it is within $0.25 \pm 0.25\%$ of its original accuracy.

Besides requiring more iterations for convergence, the results are unsurprisingly similar to the head routing experiment outlined in Figure 4.11. Since we can infer from the earlier results that decoders can sufficiently approximate the head output, fine-tuning the decoder is near-equivalent to fine-tuning a head.

Generalization to multiple Downstream Tasks Arguably, SVBI naturally generalizes to multiple downstream tasks due to approximating shallow features. We provide empirical evidence by evaluating the r-d performance of the compressors from Figure 4.12 without retraining the weights on different datasets.

Specifically, attach separate classifiers to the Swin-B backbone (as illustrated in Figure 4.9). Using PyTorch's Adam optimizer, we train each classifier for five epochs with no augmentation, a learning rate of $5 \cdot 10^{-5}$. A classifier refers to the last layers of a network.

For FrankenSplit-(SG)FP, we applied none or only rudimentary augmentation to evaluate how our method handles a type of noise it did not encounter during training. Hence, we include the Food-101 [BGVG14] dataset since it contains noise in high pixel intensities. Additionally, we include CIFAR-100 [KH+09]. Lastly, we include Flower-102 [NZ08] datasets to contrast more challenging tasks. The classifiers achieve an 87.73%, 88.01%, and 89.00% top-1 accuracy, respectively. Figure 4.16 summarizes the r-d curves for each task. Our method still demonstrates clear r-d performance gains over the baselines.

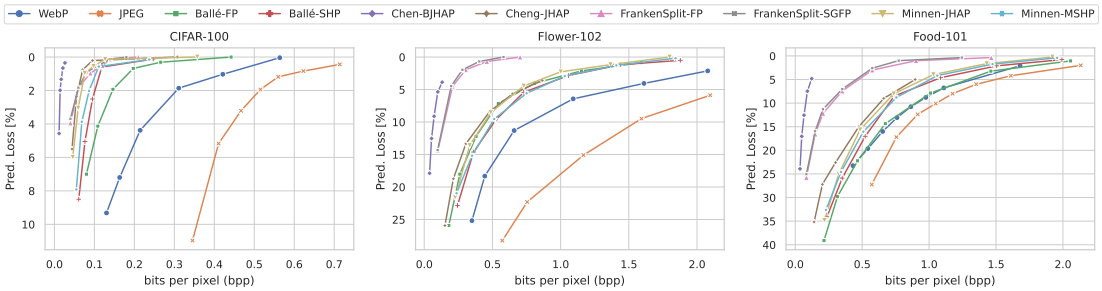


Figure 4.16: Rate-distortion curve for multiple downstream tasks

More importantly, notice how FS-SGFP outperforms FS-FP on the r-d curve for the Food-101 dataset, with a comparable margin to the ImageNet dataset. Contrarily, on the Flower-102 datasets, there is less performance difference. Presumably, on simple datasets, the suboptimality of HD is less significant. Considering how easier tasks require less model capacity, the diminishing efficacy saliency guidance is consistent with our claims.

Effect of Tensor Dimensionality on R-D Performance Section 4.1.3 argues that measuring tensor dimensionality is inadequate to assess whether partial execution on the client is worthwhile.

To verify, we implement and train additional instances of FrankenSplit with the Swin-B backbone and show results in Figure 4.17 FS-SGFP(S) is the model with a small

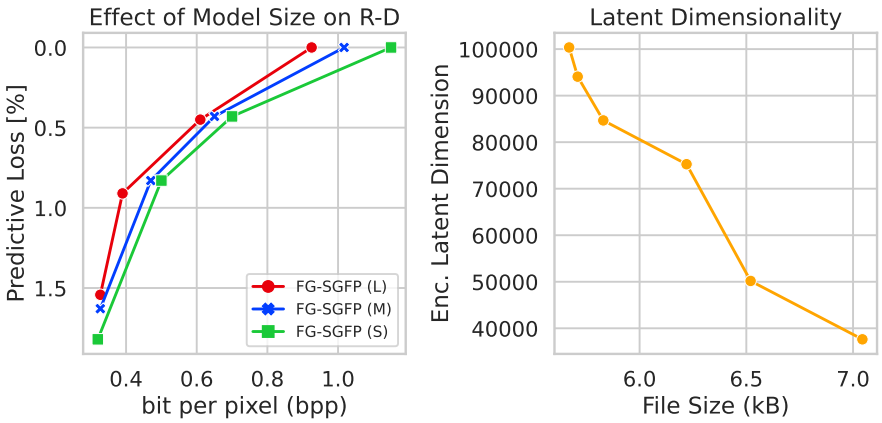


Figure 4.17: Comparing effects on sizes

encoder (∼140′000 parameters) we have used for our previous results. FS-SGFP(M) and FS-SGFP(L) are medium and large models where we increased the (output) channels $C = 48$ to 96 and 128, respectively. Besides the number of channels, we've trained the medium and large models using the same configurations. On the left, we plot the r-d curves showing that increasing encode capacity naturally results in lower bitrates without additional predictive loss. For the plot on the right, we train further models with $C = \{48, 64, 96, 108, 120, 128\}$ using the configuration resulting in lossless prediction. Notice how increasing output channels will result in higher dimensional latent tensors $C \times 28 \times 28$ but inversely correlates to compressed file size. Arguably, increasing the encoder capacity will yield more powerful transforms to decorrelate the input.

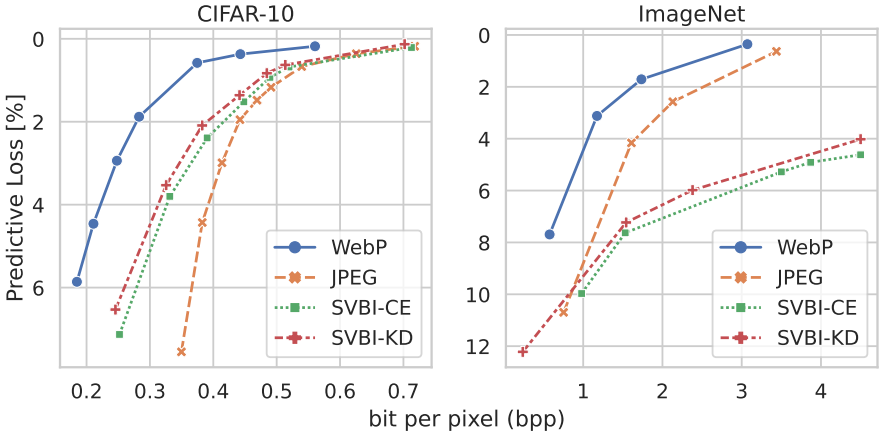


Figure 4.18: Contrasting the r-d performance

The Limitations of Direct Optimization for SVBI Section 4.2.1 mentioned that direct optimization does not work for SVBI as it does for DVBI, where the bottleneck is

at the penultimate layer. Specifically, it performs incomparably worse than HD despite the latter's inherent suboptimality. We demonstrate this by applying the SVBI-CE and SVBI-KD objective on the CIFAR-10 [KH+09] and ImageNet dataset. All models are identical and trained with the setup in Section 4.2.2, except we train for more epochs to account for slower convergence.

Figure 4.18 summarizes the results the results. On CIFAR-10, SVBI-CE and SVBI-KD yield moderate performance gain over JPEG. Yet, they perform substantially worse on ImageNet. Sufficiency as a necessary precondition may explain why the objective in (4.3) does not yield good results when the bottleneck is at a shallow layer, as the mutual information $I(Y;\tilde{Y})$ is not adequately high. Since the representation of the last hidden and shallow layer are so far apart in the information path, there is insufficient information to minimize $\mathcal{D}(H;\tilde{H})$. The compression model approximates the intermediate representation for a simple classification task to minimize predictive loss by incurring higher bitrates. Consequently, for the challenging ImageNet classification task, the same method incurs significant predictive loss even when skewing the r-d objective heavily towards high bitrates.

Prediction Latency and Overhead

We exclude entropy coding from our measurement, since not all baselines use the same entropy coder. For brevity, the results implicitly assume the Swin-B backbone for the remainder of this section. Inference times with other backbones for FrankenSplit can be derived from Table 4.5. Analogously, the inference times of applying LIC models for

Table 4.5: Execution Times of FS (S) with Various Backbones

Backbone	Overhead Prams (%)	Inf. Server+NX (m/s)	Inf. Server+TX2 (m/s)
Swin-T	2.51	7.83	9.75
Swin-S	1.41	11.99	13.91
Swin-B	1.00	16.12	18.04
ConvNeXt-T	3.46	6.83	8.75
ConvNeXt-S	1.97	8.50	10.41
ConvNeXt-B	0.90	9.70	11.62
ResNet-50	3.50	13.16	10.05
ResNet-101	2.01	8.13	15.08
ResNet-152	1.48	18.86	20.78

different unmodified backbones can be derived using Table 4.2. Notably, the relative overhead decreases the larger the tail is, which is favorable since we target inference from more accurate predictors.

Computational Overhead We first disregard network conditions to get an overview of the computational overhead of applying compression models. Table 4.6 summarizes

Table 4.6: Inference Pipeline Components Execution Times

Model	Prams Enc./Dec.	Enc. [NX/TX2] (ms)	Dec. (ms)	Full [NX/TX2] (ms)
FrankenSplit	**0.14M/ 2.06M**	**2.92/ 4.87**	2.00	**16.34/ 18.29**
Ballé-FP	3.51M/ 3.51M	27.27/ 48.93	**1.30**	41.71/ 63.37
Ballé-SHP	8.30M/ 5.90M	28.16/ 50.89	1.51	42.81/ 65.54
Minnen-MSHP	14.04M/ 11.65M	29.51/ 52.39	1.52	44.17/ 67.05
Minnen-JHAP	21.99M/ 19.59M	4128.17/ 4789.89	275.18	4416.7/ 5078.2
Cheng-JHAP	16.35M/ 22.27M	2167.34/ 4153.95	277.26	2457.7/ 4444.3
Lu-JHAP	5.28M/ 4.37M	2090.88/ 5011.56	352.85	2456.8/ 5377.8
Chen-BJHAP	36.73M/ 28.08M	3111.01/ 5837.38	43.16	3167.3/ 5893.6

the execution times of the prediction pipeline's components. Enc. NX/TX2 refers to the encoding time on the respective client device. Analogously, dec. refers to the decoding time at the server. Lastly, Full NX/TX2 is the total execution time of encoding at the respective client plus decoding and the prediction task at the server. Lu-JHAP demonstrates how LIC models without a sequential context component are noticeably faster but are still 9.3x-9.6x slower than FrankenSplit despite a considerably worse r-d performance. Notice that the computational load of FrankenSplit is near evenly distributed between the client and the server. The significance of considering resource asymmetry is emphasized by how the partially parallelized context model of Chen-BJHAP leads to faster decoding on the server. Nevertheless, it is slower than other JHAP baselines due to the overhead of the increased encoder size outweighing the performance gain of the blocked context model on constrained hardware.

Competing against Offloading The average compressed filesize gives the transfer size from the ImageNet validation set. Using the transfer size, we evaluate transfer time on a broad range of standards. Since we did not include the execution time of entropy coding for learned methods, the encoding and decoding time for the handcrafted codecs is set to 0.

The setting favors the baselines because both rely on sequential CPU-bound transforms. Table 4.7 summarizes how our method performs in various standards. Due to space constraints, we only include LIC models with the lowest request latency (Minnen-MSHP) or the lowest compression rate (Chen-BJHAP). Still, with Table 4.6 and the previous

Table 4.7: Total Latency with Various Wireless Standards

Standard/ Data Rate (Mbps)	codec	Transfer (ms)	Total [TX2] (ms)	Total [NX] (ms)
BLE/ 0.27	FS-SGFP (0.23)	142.59	160.48	158.53
	FS-SGFP (LL)	209.89	227.78	225.83
	Minnen-MSHP	348.85	415.89	393.01
	Chen-BJHAP	40.0	6167.79	3441.41
	WebP	865.92	879.06	879.06
	PNG	2532.58	2545.72	2545.72
4G/ 12.0	FS-SGFP (0.23)	3.21	21.09	19.15
	FS-SGFP (LL)	4.72	22.61	20.66
	Minnen-MSHP	7.85	74.89	52.01
	Chen-BJHAP	0.9	6128.69	3402.31
	WebP	19.48	32.63	32.63
	PNG	56.98	70.13	70.13
Wi-Fi/ 54.0	FS-SGFP (0.23)	0.71	18.6	16.65
	FS-SGFP (LL)	1.05	18.93	16.99
	Minnen-MSHP	1.74	68.78	45.9
	Chen-BJHAP	0.2	6127.99	3401.61
	WebP	4.33	17.47	17.47
	PNG	12.66	25.81	25.81
5G/ 66.9	FS-SGFP (0.23)	0.58	18.46	16.51
	FS-SGFP (LL)	0.85	18.73	16.78
	Minnen-MSHP	1.41	68.44	45.56
	Chen-BJHAP	0.16	6127.95	3401.57
	WebP	3.49	16.64	16.64
	PNG	10.22	23.36	23.36

results, we can infer that the LIC baselines have considerably higher latency than FrankenSplit. Generally, the more constrained the network is the more we can benefit from reducing the transfer size. In particular, FrankenSplit is up to 16x faster in highly constrained networks, such as BLE. Conversely, offloading with fast handcrafted codecs may be preferable in high-bandwidth environments. Yet, FrankenSplit is significantly better than offloading with PNG, even for 5G. Figure 4.19 plots the inference latencies against handcrafted codecs using the NX client. For stronger connections, such as 4G LTE, it is still 3.3x faster than using PNG. Nevertheless, compared to WebP, offloading seems more favorable when bandwidth is high. Still, this assumes that the rates do not fluctuate and that the network can seamlessly scale for an arbitrary number of client connections. Moreover, we did not apply any optimizations to the encoder.

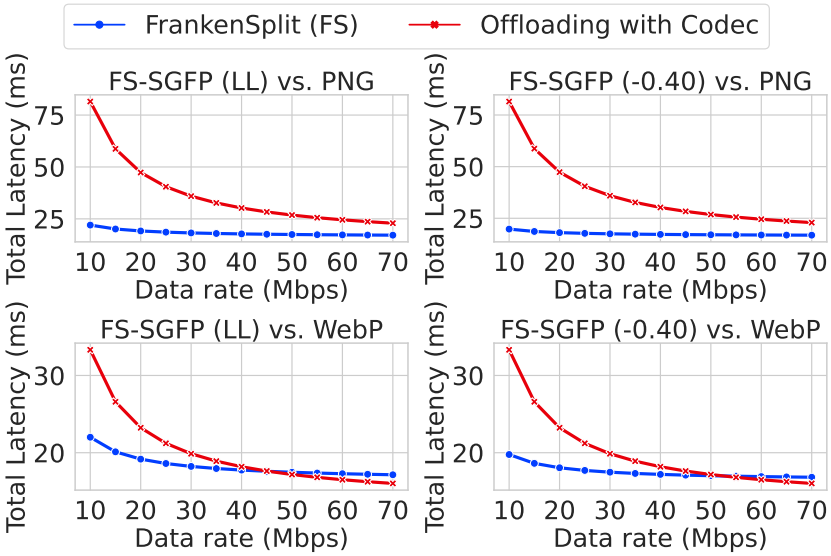


Figure 4.19: Comparing effects on sizes

4.3 Summary

This chapter has introduced the core method concerning bandwidth requirements from request payloads in urban environments. It has shown the difficulty of exploiting local resources due to resource asymmetry and introduced a method for task-agnostic feature compression with (near-) lossless prediction. An informed decoder design made relying on hard signals from deeper layers obsolete, improving the (semantic) rate-distortion performance while enabling the encoder to be reusable across backbones with varying architectural styles. A novel distillation approach further improves performance by distilling soft signals from deeper layers. Extensive evaluation has shown the method to outperform numerous competitive baselines convincingly.

CHAPTER 5

Downlink Bottlenecks in Remote Sensing

Previously, we have assumed urban environments with near-permanent and high-quality network availability with Wi-Fi, 5G base stations, etc. This chapter explores the problem of efficiently transmitting and recovering salient information at the opposite extreme of available networking infrastructure. It extends SVBI to include the peculiarity of intermittently available network connections in Low Earth Orbit and addresses challenges when recovering trustworthy human image interpretable images from low-bitrate lossy compression.

5.1 Satellite Edge Computing

5.1.1 Introduction

The development of commercial ground stations [TPK21] and the advancement in aerospace technology has enabled the emergence of nanosatellite constellations [LBC+18] in low earth orbit (LEO) as a novel mobile platform. The standardization of small form factors, such as CubeSat [LHT+09], reduces launch costs, allowing for frequent updates and deployments. Manufacturers typically equip satellites with sensors to capture large geographic regions. The downlinked satellite imagery enables Earth observation (EO) services with socially beneficial applications, such as agriculture [SRS20] and disaster warning [TD22]. Nonetheless, most constellations follow a "bent pipe" architecture where satellites downlink raw sensor data for processing in terrestrial data centers. Notably, given the constraints of orbital dynamics, satellites may only establish a connection for a few minutes. For example, the Dove High-Speed Downlink (HSD) system [DKL+17] provides segments with volumes as low as 12 GB during a single ground station pass.

As constellation sizes and sensor resolutions increase, downlink bandwidth cannot keep up with the accumulating data volume [VSC21, TMGV23]. Additional ground station equipment may prevent link saturation. However, building and maintaining them, including licensing the necessary frequencies, is a significant cost factor for satellite operation. As an alternative, Orbital Edge Computing (OEC) proposes processing data at the source [DL20, FMD+20, WLX+23, GFM+22, WL23]. Recent work on reducing bandwidth requirements in OEC is roughly categorizable in aggressive (task-oriented) filtering and compression [WLX+23]. The former relies on subjective value measures that restrict their practicability to coarse-grained tasks, such as de-duplication or cloud filtering. The latter constrains entire missions to particular tasks or prediction models. We argue that the limitations of existing compression or other data reduction approaches are particularly adverse to OEC.

First, the CubeSat design is intended for short-duration missions [LHT+09] (typically up to 3-5 years), and despite waning prices, launching sensor networks in space is still associated with substantial logistical, administrative, and monetary costs. Therefore, it seems undesirable to designate entire constellations to a small subset of tasks and Deep Neural Network (ANN) architectures. More pressingly, irrespective of whether current codecs can prevent bottlenecks, they may undermine the effectiveness of entire missions. Precisely, the assumption that prediction models only require a subset of information for image reconstruction may lead to false confidence in a codec to reliably discern the salient signals. We argue the opposite holds, i.e., when the objective is to accommodate *arbitrary* downstream tasks with prediction models instead of human experts, there is *less* potential for rate reductions. Intuitively, two seemingly visually identical images may have subtle differences in pixel intensities, which a prediction model could leverage to overcome physiological restrictions.

In summary, three conflicting objectives aggravate the challenges for OEC: (i) maximizing downlinking captures, (ii) ensuring the value of the captures by relying on as few assumptions on downstream tasks as possible, and (iii) minimizing the risk from unpredictable adverse effects on current and future prediction models. To this end, we propose drawing from recent work on neural feature compression with Shallow Variational Bottleneck Injection (SVBI) [FRD24]. The idea of SVBI is to reduce discarding information necessary for arbitrary, practically relevant tasks by targeting the shallow representation of foundational models as a reconstruction target in the rate-distortion objective. In other words, rate reductions come from constraining the solution space with abstract high-level criteria rather than reifying target tasks with an explicit definition of value or expert-crafted labels. We investigate whether the SVBI framework is suitable for EO from a compression perspective and identify lower-level system considerations given the oppressive constraints of OEC. Then, we apply our insights to introduce a **T**ile **H**olistic **E**fficient **F**eatured **O**riented **O**rbital **L**earned (THE FOOL) compression method, which we will refer to as FOOL for short. FOOL alleviates the challenges of OEC by generalizing SVBI to improve compression performance while introducing more specific methods that aid in meeting the requirements of OEC and EO tasks. FOOL comprises a profiler, a

neural feature codec with a separate reconstruction model, and a simple pipeline. The profiler identifies configurations that maximize data size reduction, factoring in intermittently available downlinks and the trade-off between processing throughput and lowering bitrate from more powerful but costlier transforms. The neural codec's architecture includes task-agnostic context and synergizes with the profiler's objective to maximize throughput with batch parallelization by exploiting inter-tile spatial dependencies. The pipeline minimizes overhead by CPU-bound pre- and post-processing with concurrent task execution. We perform in-depth experiments to scrutinize our approach with a wide range of evaluation measures by emulating conditions on a testbed with several edge devices. Our results show that FOOL is viable on CubeSat nanosatellites and increases the downlinkable data volume by two orders of magnitude relative to bent pipes at no loss on performance for EO. Unlike a typical task-oriented compression method, it does not rely on prior information on the tasks. Additionally, FOOL exceeds existing SVBI methods with an up to 2.1× bitrate reduction. Lastly, the reconstruction model can map features from the compressed shallow feature space to the human-interpretable input space. The resulting images compete with state-of-the-art learned image compression (LIC) models using mid-to-high quality configurations on PSNR, MS-SSIM, and LPIPS [ZIE+18] with up to 77% lower bitrates. Like all core methods introduced in the thesis, we open-source the core compression algorithm[1]. Crucially, it proposes a solution approach that assumes reconstruction for human interpretability as a subset of objectives that prioritize maintaining the integrity of model predictions.

5.1.2 Related Work

Collaborative Inference and Data Compression

The Deep Learning aspect of our method draws from recent advancements in collaborative inference [MLR22] and data compression [YMT23]. The underlying compression algorithm and objective functions are derived and extended from our previous work [FRD24], which re-formulizes the distortion term from lossy compression methods [BCM+20] and deploys lightweight models suitable for resource-constrained mobile devices. Besides introducing novel components to further lower transfer costs, FOOL considers the diverging requirements due to intrinsic differences between terrestrial and orbital remote sensing.

Preventing Link Saturation with Orbital Inference

The system aspect of our method aligns best with work focusing on getting the data to the ground for further processing instead of performing inference on board [GDdG+20, GFM+22, DL20, ZYX+24]. We emphasize the high variability among fundamental design principles for OEC, as it is an emerging field, and a comprehensive literature review is not within the scope of this work. In summary, we found that current approaches focus on designing complex systems tailored to specific conditions and rely on strong assumptions limiting their applicability. Moreover, they may adequately model the system conditions

[1] https://github.com/rezafuru/the-fool

but run experiments on toy tasks or on low-resolution images. Contrastingly, FOOL is a holistic approach to the downlink problem that considers satellite systems and imagery properties. The following discusses the approaches we find most promising as representatives in their general direction.

Gadre et al. introduce *Vista* [GKM22], a Joint Source Channel Coding (JSCC) system for LoRa-enabled CubeSats designed to enhance low-latency downlink communication of satellite imagery and ANN inference. It shows significant improvements in image quality and classification performance through LoRa-channel-aware image encoding. Moreover, the evaluation assumes simple tasks that are not representative of practical EO. In contrast, FOOL decouples image recovery from the initial compression objective and ensures task-agnostic preservation of information.

Lu et al. introduce *STCOD* [LCH+23], a JSCC system for efficient data transmission and object detection in optical remote sensing. STCOD integrates satellite computing to process images in space, distinguishing between regions of interest (ROIs) and backgrounds. It shows promising results with a block-based adaptive sampling method, prioritizing transmitting valuable image blocks using fountain code [Mac05]. The caveat is that ROI detectors that can reliably prevent predictive loss for downstream tasks require strong biases regarding sensor and task properties. FOOL includes task-agnostic context, with significantly less overhead and more robustness towards varying conditions than an ROI detector. Furthermore, it is end-to-end optimized with the other compression model components without relying on the same biases or expert-crafted labels.

Thematically, our work resembles Kodan by Denby et al. [DCC+23] the closest. Like FOOL, Kodan treats channel conditions as an orthogonal problem and primarily focuses on source coding to address the downlink and computational bottlenecks. Kodan uses a reference application for satellite data analysis and a representative dataset to create specialized small models. Once in orbit, it dynamically selects the best models for each data sample to maximize the value of data transmitted within computational limitations. Kodan's excellent system design is promising but relies on assumptions that hinder practicability and the potential for meaningful rate reductions. Unlike Kodan, we follow a different design philosophy by treating the downlink bottleneck primarily as a compression problem. Further, we do not treat the computational deadline as a hard temporal constraint to decouple the method to a particular system design, as reflected by FOOL's profiler measuring key performance indicators on the *pixel level*. Given hardware limitations, the aim is to reduce transfer costs by balancing the lower bitrate of more powerful encoders and the gain in processing throughput of more lightweight encoders.

5.1.3 Background

The Downlink Bottleneck

Downlink bottlenecks occur when the data volume exceeds the bandwidth within a downlink segment during a single pass. We formalize a model sufficient for our purposes by considering link conditions and sensor properties of satellites belonging to a constellation.

A constellation is defined as $\mathcal{C} = (L, \mathcal{S}, I, f)$ where L is a link to communicate with a ground station and $\mathcal{S}$ is a set of satellites. The link is determined by its *expected* downlink rate, measured in megabits per second (Mbps). The function $f : S \to I$ maps each satellite $s \in S$ to an interval where it passes the downlink segment into disjoint subsets $\mathcal{G} = \{\mathcal{G}_i | \mathcal{G}_i = \{s \in \mathcal{S} | f(s) = i\}, i \in I\}$, such that $\bigcup \mathcal{G}_i = \mathcal{S}$ and $\bigcap \mathcal{G}_i = \varnothing$. The link capacity V_{link} is the bandwidth available per pass and is determined by the link rate and the interval range. Satellites $S = (R_{\text{orbit}}, S_{\text{rate}}, S_{\text{spatial}}, S_{\text{bands}}, S_{\text{radio}}, S_{\text{fov}})$ are equipped with a sensor, and its properties determine the volume per capture.

$$V_{\text{capture}} := \overbrace{\frac{R_{\text{orbit}}^2 \cdot \tan^2(S_{\text{fov}})}{S_w \cdot S_h}}^{\text{Total Pixels}} \cdot \underbrace{S_{\text{bands}} \cdot S_{\text{radio}}}_{\text{Bits per Pixel}} \tag{5.1}$$

The radiometric resolution S_{radio} and number of bands S_{bands} determine the downlink cost per pixel in bits. The orbit R_{orbit}, sensor spatial resolution $S_{\text{spatial}} = S_h \times S_w$, and field of view S_{fov} determine the number of pixels per capture. The number of captures depends on the time to complete an orbit

$$T_{\text{orbit}} := 2\pi \sqrt{\frac{(R_{\text{orbit}} + R_{\text{earth}})^3}{GM}} \tag{5.2}$$

and on the capture rate S_{rate}. G is the gravitational constant and M is the earth's mass. The orbit R_{orbit} is usually around 160 to 800 kilometers for LEO satellites. For reference, R_{orbit} is 786 kilometers for Sentinel-2 [DDC+12]. Finally, the number of captures from all satellites within the segmentation group determines the total volume per pass.

$$V_{\text{pass}} := \sum_{s^{(i)} \in \mathcal{G}_j} T_{\text{Orbit}} \cdot S_{\text{rate}}^{(i)} \cdot V_{\text{capture}}^{(i)} \tag{5.3}$$

The superscript (i) denotes the costs associated with a satellite (i). For constellations with homogenous sensors V_{capture} is a static value. Notice that V_{pass} scales linearly by the constellation size and a constant factor c for overlap occurrences, i.e., $|\mathcal{G}_j| = \frac{|\mathcal{S}|}{c}$. To determine c for a constellation, we must calculate the minimum angle between satellites β^*. Assuming a single ground station at the Earth's North Pole and given the minimum communication elevation θ

$$\beta^* = 2 \times (180^\circ - (\theta + 90^\circ) - \arcsin(\frac{R_{\text{earth}} \cdot \sin(90^\circ + \theta)}{R_{\text{orbit}} + R_{\text{earth}}}) \tag{5.4}$$

For example, consider a constellation at $R_{\text{orbit}} = 790,000$ meters altitude with a minimum elevation $\theta = 25^\circ$ such that $\beta^* \approx 22.52^\circ$. Then, $c = \frac{360^\circ}{22.52^\circ} \approx 16$, i.e., to prevent any interval sharing, the constellation size may not exceed 16 satellites.

In short, the aim is to facilitate cost-efficient scaling of constellations by increasing bandwidth value and substantially reducing reliance on building additional infrastructure.

That is, we require an encoding scheme enc, such that $V_{\texttt{enc}} < V_{\text{link}}$. Note that a single satellite may experience a bottleneck even if the constellation is sparse enough to prevent interval sharing [DCC+23]. Say, each $s \in \mathcal{S}$ is equipped with a sensor using approximate Sentinel-2 configurations [DDC+12] by setting a multispectral sensor for (near-) visible light to $S_{\text{bands}} = 4$, $S_{\text{radio}} = 12$ $S_{\text{fov}} = 21°$, $S_{h \times w} = 10 \times 10$, and five captures per pass. With $|\mathcal{S}| \leq 16$, the volume for each pass is $\frac{790{,}000^2 \cdot \tan^2(10.5°)}{100} \cdot 4 \cdot 12 \cdot 5 \approx 410$ GB. To prevent a bottleneck even without sharing an interval and using a higher-end link, such as WorldView-3 [CCA+21] where $V_{\text{link}} = 90$ GB per pass, the enc needs to decrease the data volume by a factor of 4.5.

There are two overarching objectives for a codec and the system we deploy its encoder. The system's objective is to process and encode large volumes of high-dimensional data, given the physical limitations of LEO (nano-) satellites. A 3U nanosatellite following the CubeSat standard is limited to 10cm×10cm×30cm and 4kg [CHJ+22] with restricted power supply by using solar harvesting [DL19]. The compression objective is to achieve a sufficiently low bitrate while maintaining the data's integrity. The following elaborates on the challenges of conceiving a method that fulfills our criteria and the limitations of applying existing codecs.

Limitations of Codecs

Given remote image captures and a set of unknown associated object detection tasks, we seek a transformation of the captures into representations that minimize transfer costs and loss of information that may impact any detection tasks. We refer to generalizability as a measure of how well a method can minimize the predictive loss on unknown detection tasks. For example, a purely task-oriented encoding (e.g., [SAEHJ+20]) can retain information for a set of explicitly defined tasks. Still, it does not generalize as the transformed data is unusable for non-overlapping tasks Besides bent pipes, *lossless* codecs are the only approach with easily understood guarantees on generalization. Nevertheless, lossless compression cannot adequately address the downlink bottleneck due to theoretical lower bounds. Promising alternatives are *lossy* methods that relax the requirement of relying on identical reconstruction for generalization. More formally, given a distortion measure $\mathcal{D}$, a constraint D_c bounds the minimal bitrate to $\min_{P_{Y|X}} I(X;Y)$ s.t. $\mathcal{D}(X,Y) \leq D_c$. Neural Image Compression (NIC) replaces the typically linear transformation of handcrafted codecs with nonlinear ones to reduce dependencies from sources that are not jointly Gaussian [BCM+20]. The sender applies a parametric analysis transform $g_a(\mathbf{x};\theta)$ into a latent $\mathbf{y}$, which is quantized to a latent with discrete values $\hat{\mathbf{y}}$. Then, an entropy coder losslessly compresses $\hat{\mathbf{y}}$ using a shared entropy model $p_{\hat{\mathbf{y}}}$. The receiver decompresses $\hat{\mathbf{y}}$ and passes it to a parametric synthesis transform $g_s(\hat{\mathbf{y}};\phi)$ to recover a distorted approximation to the input. To capture leftover spatial dependencies of $\hat{\mathbf{y}}$, more recent work adds *side information* with a hyperprior $\mathbf{z}$ [BMS+18] and a context model [MBT18]. Including side information requires two additional parametric transforms $h_a(\hat{\mathbf{y}};\theta_h)$ and $h_s(\hat{\mathbf{z}};\psi_h)$. Despite efficient LIC methods [MCJ+24] consistently outperforming handcrafted codecs on standardized benchmarks, the results are deceptive when assessing the impact on

downstream tasks. To provide further explanation, we perform a preliminary experiment that contrasts the predictive loss with additive noise and codec distortion and summarize results in Figure 5.1.

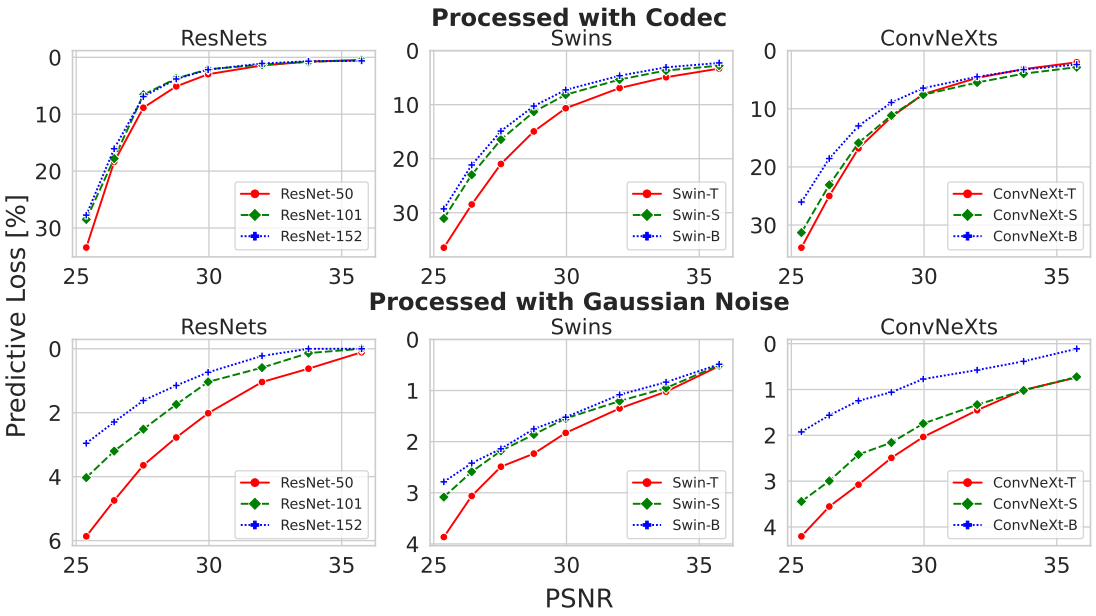


Figure 5.1: Comparing Effect of Codec and Additive Noise

We download pre-trained weights [WTJ21] for the ImageNet [RDS+15] classification task of three popular architectures[HZRS16, LLC+21, LMW+22]. First, we compute the expected Peak Signal-To-Noise Ratio (PSNR) of a popular LIC model [BLS17] for each quality level on the validation set. Then, we apply Additive White Gaussian Noise (AWGN) on the input to match the PSNR of a codec for each quality level separately. Lastly, we measure the *predictive loss* as the average difference between the accuracy of the original and processed samples. Notice that the predictive loss on the distorted input is significantly worse than the noisy input. Additive noise does not remove information; rather, it superimposes unwanted information. Conversely, lossy compression intentionally discards information from signals, and two codecs may achieve comparable rate-distortion performance despite emphasizing different information to retain. Re-training model weights on reconstructed samples may mitigate some predictive loss, but only due to adjusting prediction to input perturbations and error-prone extrapolation of lost information.

Particularly, for EO with satellite imagery that spans large geographic areas, we stress the unsuspecting danger of lossy compression, which is compounded with learned transforms [BCM+20], where it is challenging to understand behavior. The ability to differentiate between intensities beyond the capability of humans may explain why detection models can outperform domain experts. Accordingly, we should assume that lossy codecs may discard information where even experts cannot reliably verify the impact on machine

interpretability. For example, suppose a codec that reduces the rate by focusing on preserving coarser-grained structures. Then, tasks that rely on assessing the environment for fine-grained object classes will lack background information (e.g., inferring region by tree species with subtle color variations).

Current limitations of image codecs put operators in a difficult position, especially for EO. The decision falls between (a combination of) lossless codecs, applying crude filtering methods, or attempting to reduce the bitrate with lossy codecs, remaining uncertain about whether the codecs retain information necessary in real conditions. As a solution, we previously introduced SVBI, which prioritizes salient regions for (near) arbitrary high-level vision tasks. To briefly recap, SVBI trains neural codecs by replacing the distortion term of the rate-distortion objective of variational image compression models [BMS$^+$18] with head distillation (HD) [MBC$^+$19, SSTM21, MYLM22, FRD24], using the shallow representation of a pre-trained foundational model as the prediction target. We define a foundational model as a pre-trained ANN that can accommodate multiple tasks by attaching predictors or fine-tuning the deeper layers. In Knowledge Distillation (KD) terminology, the codec is referred to as the *student* and the shallow layers of a foundational model as the *teacher*. Note that KD is not this work's focus, as HD diverges from the typical KD objective. The intuition behind SVBI is that if a codec can reconstruct the representation of a foundational model, then the representation is sufficient for *at least* all tasks associated with that model.

The Effectiveness of Shallow Features

Readers may reasonably assume that using the representation for one particular network architecture instead of the input as the distortion measure is more restrictive for two reasons. First, the features are not human-readable, i.e., we cannot overlay the bounding boxes on the images. We could infer the global coordinates to present boxes overlaid on previously captured satellite imagery. This is sufficient for observing (semi-) permanent objects (e.g., landmarks) but certainly not for ephemeral or moving objects (e.g., tracking the movement of vessels). Second, even if the trend toward transfer learning with foundational models [JT21, ANK$^+$25] can accommodate various predictors, client preferences for architectures may vary.

We argue that by targeting shallow representations, both limitations can be addressed. View an n-layered feed-forward neural network as a Markov chain of successive representations R_i, R_{i+1} [TZ15]:

$$I(X;Y) \geq I\left(R_1;Y\right) \geq \ldots \geq I\left(R_n;Y\right) \geq I(\tilde{Y};Y). \tag{5.5}$$

The mutual information $I(X;R_i)$ will likely decrease relative to the distance between the input and a representation. This loss stems from layers applying operations that progressively restrict the solution space for a prediction, particularly for discriminative tasks. That is, the deeper the representation, the more information we are likely to lose regarding X:

$$I(X;X) \geq I(R_1;X) \geq \cdots \geq I(R_{n-1};X) \geq I(R_n;X). \tag{5.6}$$

Figure 5.2 visualizes how information is gradually discarded in the processing path.

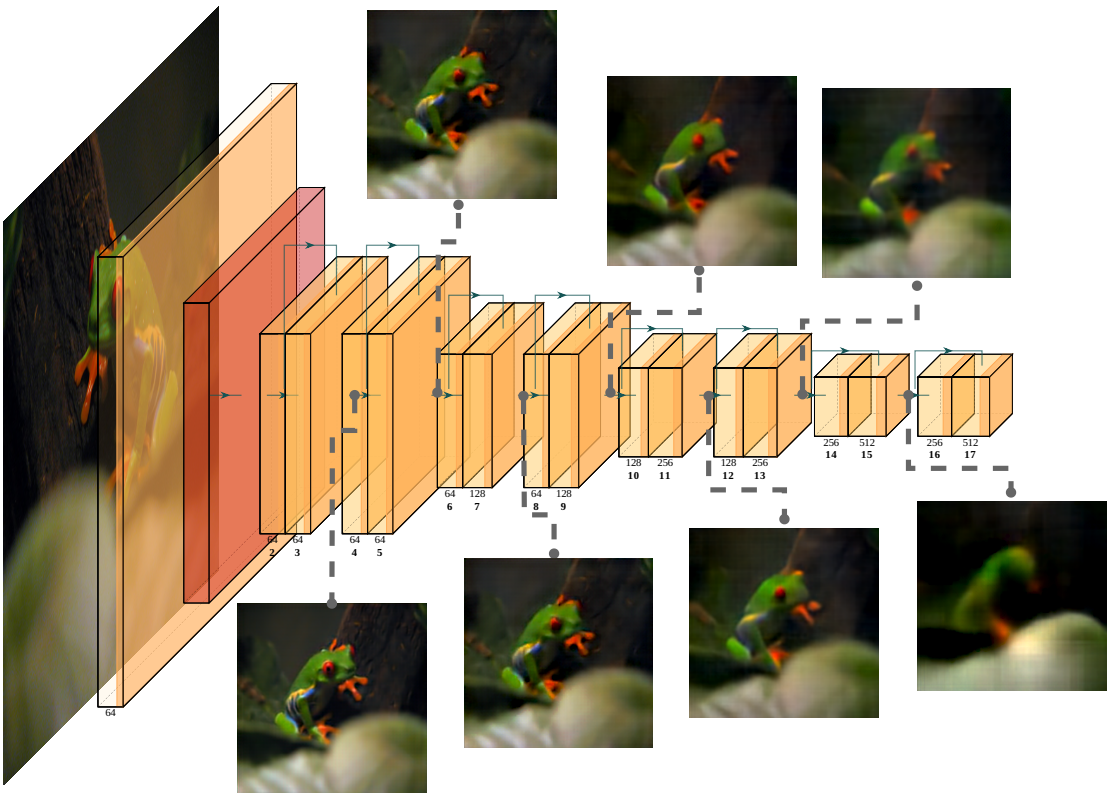


Figure 5.2: Data Processing Inequality visualized.

We extract the features from a ResNet network with weights trained on the ImageNet [RDS+15] classification task and recover the original image by training separate reconstruction models for each marked location. Section 5.2.1 elaborates on the reconstruction. Notice that models at shallow layers can recover the input with high similarity, but the recovery progressively worsens as the path distance increases. Now assume a discriminative model, such as a ResNet for image classification, $\mathcal{M} = (\mathcal{H}, \mathcal{T})$ with separate shallow $\mathcal{H}$ from deeper layers $\mathcal{T}$ as disjoint subsets, such that $\mathcal{H}(X) = H$ (i.e, mapping input to shallow features) and $\mathcal{M}(X) = \mathcal{T}(\mathcal{H}(X))$. Further, given a codec $\mathrm{c} = (\mathrm{enc}, \mathrm{dec})$ where $\mathrm{dec}(\mathrm{enc}(X)) = \tilde{H}$ is an approximation of $\mathcal{H}(X)$. Then, $\tilde{H}$ is a sufficient approximation of H if $\mathcal{T}(\tilde{H})$ results in lossless prediction, i.e., no drop in prediction performance relative to $\mathcal{T}(H)$. In other words, a sufficient representation in the shallow latent space results in high similarity in the deep latent space between $\mathcal{T}(H)$ and $\mathcal{T}(\tilde{H})$. Consider that similarity in the deep latent space coincides with high similarity for human perception in the input space [ZIE+18]. Therefore, the encoder output $\texttt{enc}(X)$ should retain sufficient information to reconstruct X with quality comparable to $\mathcal{H}(X)$. Finally, since $\texttt{enc}(X)$ sufficiently approximates H, it should be possible to sufficiently approximate the shallow representation of any model $\mathcal{M}' = (\mathcal{H}', \mathcal{T}')$ if $I(\mathcal{H}(X), X) \approx I(\mathcal{H}'(X), X)$.

Rate Reductions by Task Specificity

The idea of task-oriented communication is that messages for model prediction may require less information than human domain experts, so it should be possible to reduce bitrate by not (exclusively) using input reconstruction as the distortion measure. We argue that this assumption contradicts empirical evidence demonstrating models outperforming human experts in various image-related tasks, since machines can detect signals and patterns that humans physiologically or intellectually cannot. Rather, the opposite should hold. When compressing for quality using domain experts as judges, we should see more potential for rate savings, not less. The claim is consistent with the results in Figure 5.1 where codecs with high reconstruction quality result in images that are deceptively similar to the input (details in Section 5.2.4). As claimed in Section 4.1.4, rate reductions are from *task specificity* of the distortion measure, irrespective of the input interface, whether a particular layer of an ANN architecture, human receptors, or textual encoding. Note that this holds even when limiting measures to discriminative task objectives without image reconstruction. Besides visualizing Equation (5.6), the input image illustrates a practical example. The frog subset of ImageNet distinguishes between *Tree Frogs*, *Bullfrogs*, and *Tailed Frogs*. Since these frog species have distinct figures and dominant colors, the more delicate characteristics of a tree frog are redundant for ImageNet classification. The network gradually discards information regarding the fine-grained blue-yellow colored patterns, permitting only the recovery of general shape and environment from the deep features. The deeper the features, the less structure and detail are present, which may be redundant for the task. Now, suppose training a codec where the encoder retains the minimal information necessary to reconstruct the output of the deepest layers for a classification task (e.g., similar to Vista [GKM22]). Then, assuming uniform class distribution, we can reduce the transfer cost to as low as $\log_2(\#\text{labels})$ without predictive loss. However, we may lack the information for other tasks, i.e., there is a trade-off between generalization and the lower bound on the bitrate. In contrast, targeting shallow features for compression may strike a balance between aiming to retain information for all possible downstream tasks and only emphasizing the salient regions for the tasks associated with a foundational model. Arguably, the limitation is negligible, as maintainers will train models with useful tasks in mind.

5.2 A Complete Neural Compression Pipeline for Satellite Computing

5.2.1 The FOOL's Compression Method

We design the compression method based on three criteria. First, it should synergize with the profiling strategy (Section 5.2.2). Second, it should embed context for feature compression without favoring a particular downstream task. Third, it should prioritize the integrity of downstream tasks but allow recovering human interpretable images without increasing the bitrate.

Model Building Blocks

For a focused evaluation and transparent discussion on the efficacy of our contributions in Section 5.2.3, we restrict FOOL to basic layer types and exclude methods from work on efficient neural network design (e.g., dilated convolutions to increase the receptive field). Moreover, basic layer types ensure widespread support across hardware vendors [RMJ+22]. Figure 5.3 illustrates the building blocks of the codec architecture we will introduce in Section 5.2.1 and how it organizes the primary networks for transform coding (Section 5.1.3).

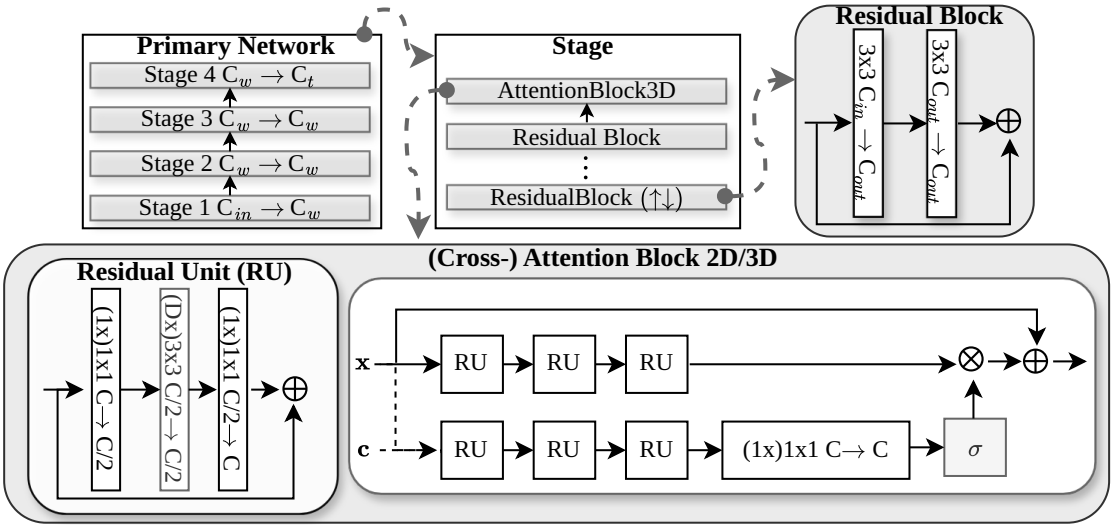


Figure 5.3: Network Organization and Components

The primary networks have four stages that control the depth and width. Each stage has at most one lightweight attention block and at least one residual block. A residual block optionally up or downsamples the spatial dimensions. A stage's width and depth parameters configure the number of channels and residual blocks within a stage.

Capturing Inter-Tile Dependencies

The input to the compression model is tiles of satellite imagery that were partitioned to maximize processing throughput (Section 5.2.2), i.e., we consider an input $\mathbf{x}$ as a list with T separate image tensors $x_t \in \mathbb{R}^{C \times H \times W}$. To decrease transfer costs further, FOOL leverages the prior knowledge from partitioning (i.e., tiles corresponding to the same image) in two ways. The first is via weight-sharing with 2D Residual Blocks by reshaping the tensor to $T \cdot B \times C \times H \times W$. This way, we include further inductive bias during training by forward passing T similar tensors before each backpropagation. The second is with an inter-tile attention mechanism. Since self-attention from transformer architectures is prohibitively expensive for our purposes, even when applying it on downsampled representations as proposed in [ERO21]. Therefore, we modify and extend the lightweight convolutional attention layer from [CSTK20]. The layer stacks residual units to increase

the receptive field that primarily emphasizes *local* interactions, i.e., it is not a drop-in replacement to capture global dependencies. We conjecture that FOOL partially replaces the need for global operations by assuming tiles to have "pseudo-temporal" dependencies. The intuition is that partitioning single captures spanning large geographic areas, which typically have recurring patterns, may be similar to moving a video feed with large strides. In particular, tiles within the same regions or biomes have global dependencies. In the 3D version of the attention block from Figure 5.3, a residual unit consists of two $1 \times 1 \times 1$ convolution and a $D \times 3 \times 3$ convolution in between. The kernel size for the temporal dimension of attention layers (Section 5.2.1) is set as $D = 3$ for $T < 5$ and $D = 5$ for $T \geq 5$. The advantage of 3D layers over concatenating channels and applying 2D convolutional operations (e.g., with channel attention [GXL+22]) is that it considerably reduces width. For example, given a 3×3 2D convolution with $T \times C$ in and out channels. Then for $T = 5$ image tensors, with dimensions $C = 64, H = 128, W = 128$, would require $5 \cdot 64 \cdot (3 \cdot 3 \cdot 5 \cdot 64 + 1) = 921,920$ parameters. Conversely, for a $5 \times 3 \times 3$ 3D convolution with the same in and out channels, it would result in $64 \cdot (5 \cdot 3 \cdot 3 \cdot 64 + 1) = 184,384$. Additionally, we reduce the number of multiply-and-accumulates from approximately 15 million to 9 million. Besides lowering memory requirements, this allows scaling model capacity with less friction against processing throughput.

Task-Agnostic Context for Feature Compression

The leftover spatial dependencies after encoding are commonly around high-contrast areas. Consider that high-contrast areas typically correspond to edges and other regions of interest, i.e., *keypoints*. As an example, Figure 5.4 contrasts leftover pixel dependencies of $\hat{y}$ from a LIC model [MBT18] to keypoints output by a KeyNet [BLRPM19] network. It shows that we can further improve compression performance with side information by embedding keypoints as context for encoding as follows:

$$
\begin{aligned}
\hat{\boldsymbol{y}} &= Q(g_a(\boldsymbol{x}; \theta)) \\
\boldsymbol{y}_{kp} &= f_{ds}(k(\boldsymbol{x}) \odot \boldsymbol{x}; \omega_{kp}) \\
\boldsymbol{y}_{ca} &= a_c(\boldsymbol{y}_{kp}, \omega_{ca}) \\
\hat{\boldsymbol{z}} &= Q(h_a(f_{ca}(\boldsymbol{y}_{kp}, \omega_{ca}); \theta_h)) \\
p_{\hat{\boldsymbol{y}}|\hat{\boldsymbol{z}}}(\hat{\boldsymbol{y}} \mid \hat{\boldsymbol{z}}) &\leftarrow h_s\left(\hat{\boldsymbol{z}}; \phi_h\right) \\
\tilde{\boldsymbol{h}} &= g_s(\hat{\boldsymbol{y}}; \phi)
\end{aligned}
$$

where k is a keypoint extraction function $k : \mathbb{R}^{3 \times H \times W} \rightarrow \mathbb{R}^{1 \times H \times W}$, f_{ds} is a parametric downsampling function $f_{ds} : \mathbb{R}^{1 \times H \times W} \rightarrow \mathbb{R}^{C' \times \frac{H}{2^n} \times \frac{W}{2^n}}$, and a_c is a single (2D) cross-attention block (Figure 5.3). The cross-attention block takes context as an additional input for weighting the latent with attention scores. For k, we use scores from a (frozen) pre-trained and simplified KeyNet [BLRPM19] due to its robustness in diverse environments and low memory requirements (less than 6000 parameters). While this method should generalize to LIC, it complements feature compression exceptionally well

and found quantizing and compressing $\mathbf{y}_{ca}$ (i.e., $\tilde{\mathbf{h}} = g_s(Q(\mathbf{y}_{ca}), \psi)$ further lowers the bitrate without affecting task performance.

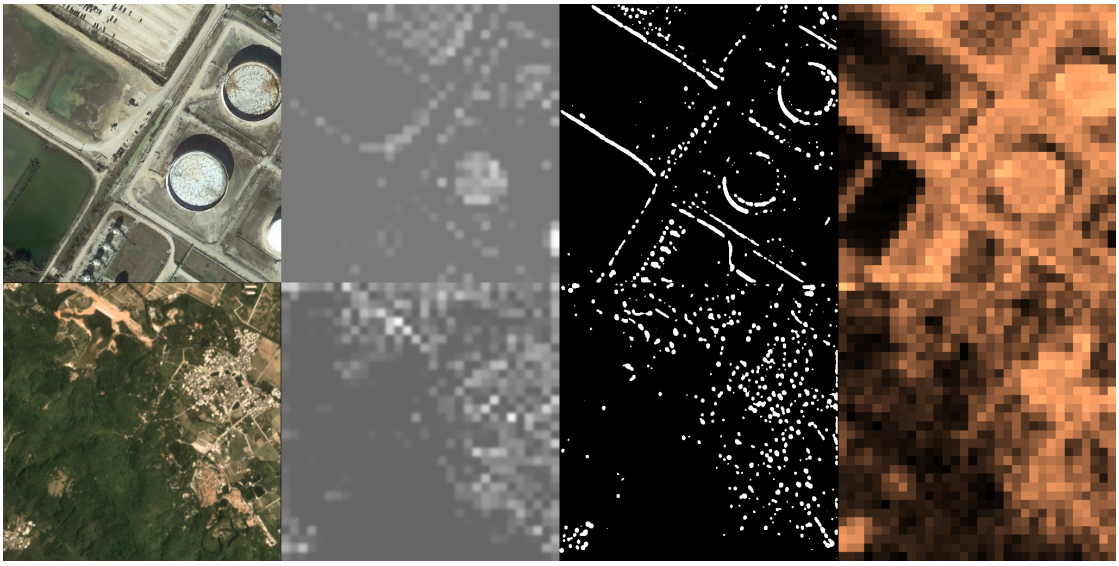


Figure 5.4: Leftover Spatial Dependencies (middle left), Keypoints (middle right), Entropy Heatmap (right)

Compression Model Architecture

Figure 5.5 illustrates the compression model's complete architecture. The dashed lines to Q indicate that we either quantize (and subsequently compress) the base latent or the cross-attention weighted latent.

For the case of passing $\mathbf{y}_{ca}$ to Q, we include an additional residual unit after attention-weighting. We skip applying the attention block to the highest input dimensions to reduce memory and computational costs. The non-linearity between the layers is ReLU to reduce vendor dependency of results from the system performance evaluation in Section 5.2.4. Residual blocks are two stacked 3×3 convolutions to increase the receptive field with fewer parameters and a residual connection for better gradient flow. For the remainder of the work, we refer to the compression model as an encoder-decoder pair enc, dec. The encoder comprises g_a, h_a, k, f_{ds}, c_a, and the entropy coder. The decoder consists of the g_s and the entropy decoder. The entropy model $p_{\hat{\mathbf{z}}}(\hat{\mathbf{z}})$ and h_s are shared. Note that, despite deploying more components on the constrained sender, the encoder has significantly fewer parameters than the decoder since we increase the width of the receiver-exclusive components.

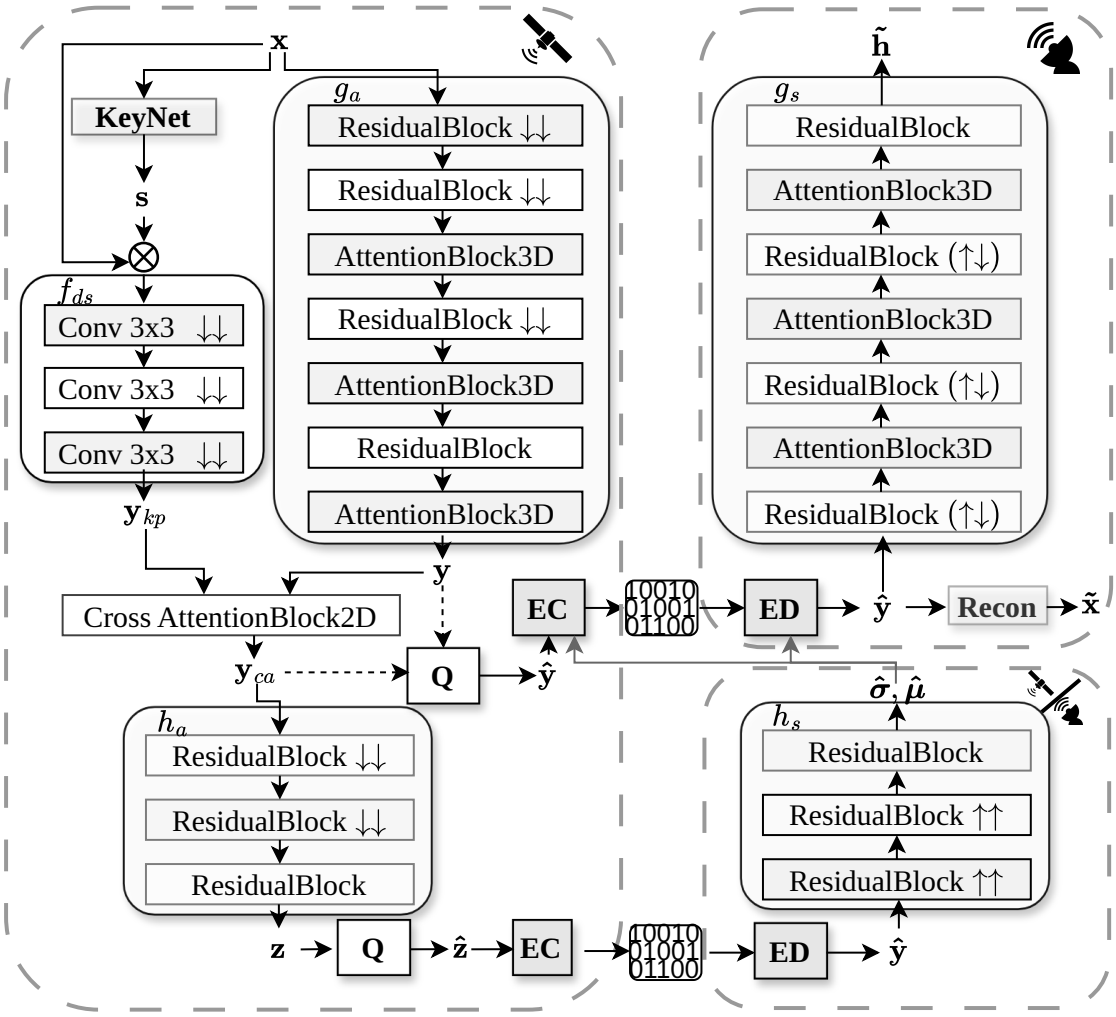


Figure 5.5: The FOOL's Compression Architecture

Single Encoder with Multiple Backbones and Tasks

Analogous to [FRD24], consider a set of n shallow and deep layers pairs of backbones (i.e., foundational models):

$$\mathcal{M}_f = (\mathcal{H}_1, \mathcal{T}_1), (\mathcal{H}_2, \mathcal{T}_2) \dots (\mathcal{H}_n, \mathcal{T}_n) \tag{5.7}$$

The shallow layers map a sample to a shallow representation, i.e., $\mathcal{H}_i(x) = h_i$. Further, associate a separate set of m predictors $\mathcal{P} = \mathcal{P}_1, \dots \mathcal{P}_m$ to the non-shallow (i.e., deep) layers of a backbone. Assume an encoder-decoder pair can sufficiently approximate the shallow layer's representation of a particular backbone (i.e., $dec(enc(\boldsymbol{x})) = \tilde{\mathrm{h}} \approx \mathrm{h_i}$). Then, inputting $\tilde{\mathrm{h}}$ to $\mathcal{T}_i$, should result in the same predictions for all m predictors associated to $\mathcal{T}_i$. Since two shallow layers output different representations (i.e., $\mathcal{H}_i(x) \neq \mathcal{H}_j$), the encoder-decoder pair cannot replace the shallow layers for any $\mathcal{H}_j$ where $i \neq j$.

Accordingly, after training an initial encoder-decoder pair, FOOL instantiates $n-1$ additional decoders, resulting in a set of separate $dec_1, dec_2 \dots dec_n$ decoders, i.e., one for each target backbone.

Image Reconstruction

FOOL trains the compression and image reconstruction models in two stages. After training the compression model and freezing encoder weights, it separately trains a reconstruction model that maps $\hat{\mathbf{y}}$ to an approximation $\tilde{\mathbf{x}}$ of the original sample $\mathbf{x}$.

Separate Training over Joint Optimization We could reduce the distortion $d(\mathbf{x}, \hat{\mathbf{x}})$ with a joint objective for training the reconstruction and compression model. While the resulting models would score higher on the sum of error benchmarks, the added distortion term will result in higher bitrates. Instead, after optimizing the encoder with the objective of SVBI, we freeze the weights (i.e., "locking" in the rate performance). Then, we leverage the high mutual information between shallow features and the input to recover presentable approximations (Section 5.1.3). Image recovery is closely related to image restoration, such as super-resolution or denoising. The component is exchangeable with the state-of-the-art, as it is orthogonal to the compression task. For this work, we select SwinIR [LCS+21] due to its relative recency, computational efficiency, and simplicity.

Reconstruction Does not Replace Decoders Approximations from decoders (Section 5.2.1) resulting in (near) lossless prediction would evidence $\hat{\mathbf{y}}$ has sufficient information to reconstruct a sample for the input layers that result in comparable task performance. Hence, after training the encoder, we could replace all decoders with the reconstruction model to approximate the input sample. Nevertheless, sufficiency may not directly result in lossless prediction since artifacts perturb the reconstructed samples. We could account for the perturbation by fine-tuning for a relatively small number of iterations [FRD24]. The downside is that operators must maintain, store, and serve different versions of the otherwise identical backbones for each client separately. Additionally, they may need to retrain the predictors of the various downstream tasks for each backbone. Instead, we train small decoders that directly map the low-dimensional encoder output to an adequate representation, i.e., FOOL does *not* pass the reconstruction to prediction models for downstream tasks. There are two advantages to introducing multiple decoders over multiple backbone weights. First, the number of additional weights operators must maintain only scales with supported backbones and not the number of backbone-task pairs. Second, the small decoder weights incur considerably less training and storage overhead than the weights of massive backbones.

Loss Functions

FOOL's training algorithm starts with extracting the shallow layers of a particular detection model (teacher). Then, it freezes the encoder and trains newly initialized

decoders $dec_1, dec_2, \ldots dec_n$ using the corresponding teacher models $\mathcal{H}_1, \mathcal{H}_2, \ldots \mathcal{H}_m$ (i.e., target shallow layers).

Rate-Distortion Loss Function for SVBI To simplify the loss expression, we treat the components related to keypoints as part of h_a if we exclusively use it as a hint for the side-information network. Alternatively, it may be used as the final block of g_a before quantizing and entropy coding the latent. Analogous to the SVBI training objective [FRD24, MYLM22], we have a parametric analysis transform $g_a(x;\theta)$ that maps x to a latent vector z. Then, a quantizer Q discretizes z to $\hat{z}$ for lossless entropy coding. Since we rely on HD (Figure 4.5) as a distortion function, the parametric synthesis transforms $g_s(\hat{x};\phi)$ that maps $\hat{y}$ to an approximation of a representation $\tilde{h}$. As introduced in [BLS17], we apply uniform quantization Q, but replace Q with continuous relaxation by adding uniform noise $\eta \sim \mathcal{U}(-\frac{1}{2}, \frac{1}{2})$ during training for gradient computation.

Without a hyperprior, the loss is:

$$\mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{x}}} D_{\mathrm{KL}}\left[q \| p_{\tilde{\mathbf{y}}|\boldsymbol{x}}\right] = \mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{x}}} \mathbb{E}_{\tilde{\mathbf{y}}\sim q} - \log p(x|\tilde{\mathbf{y}}) - \log p(\tilde{\mathbf{y}})] \tag{5.8}$$

With side information, we condition on a hyperprior, such that each element $\hat{\mathrm{y}}_i$ is now modeled as a Gaussian with its own mean and standard deviation:

$$p_{\tilde{\mathbf{y}}|\tilde{\mathbf{z}}}\left(\tilde{\mathbf{y}} \mid \tilde{\mathbf{z}}, \phi_h\right) = \prod_i \left(\mathcal{N}\left(\mu, \tilde{\sigma}_i^2\right) * \mathcal{U}\left(-\frac{1}{2}, \frac{1}{2}\right)\right)\left(\tilde{y}_i\right) \tag{5.9}$$

where $\mathbf{z} = h_a(\hat{\mathbf{y}};\theta_h)$ and $\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\sigma}} = h_s(\tilde{\mathbf{z}};\phi_h)$. The final loss function results in the following:

$$\begin{aligned} \mathbb{E}_{\mathbf{x}\sim p_{\mathbf{x}}} D_{\mathrm{KL}}\left[q \| p_{\tilde{\mathbf{y}},\tilde{\mathbf{z}}|\mathbf{x}}\right] = \mathbb{E}_{\mathbf{x}\sim p_{\mathbf{x}}} \mathbb{E}_{\tilde{\mathbf{y}},\tilde{\mathbf{z}}\sim q} [&\log q(\tilde{\mathbf{y}}, \tilde{\mathbf{z}} \mid \mathbf{x}) \\ &- \log p_{\mathbf{x}|\tilde{\mathbf{y}}}(\mathbf{x} \mid \tilde{\mathbf{y}}) - \log p_{\tilde{\mathbf{y}}|\tilde{\mathbf{z}}}(\tilde{\mathbf{y}} \mid \tilde{\mathbf{z}}) \\ &- \log p_{\tilde{\mathbf{z}}}(\tilde{\mathbf{z}})] \end{aligned} \tag{5.10}$$

For the distortion term, we use the sum of squared errors between the shallow layer (teacher) representation and the compressor (student) approximation, i.e., $\mathrm{sse}(\mathbf{h}, \tilde{\mathbf{h}})$.

Mapping Encoder Output to Target Representations After training the first enc, dec_1 pair, FOOL freezes enc weights, i.e., only applying the distortion term of the loss in Equation (5.10), for subsequent decoders $dec_2, dec_3 \ldots dec_n$ (Section 5.2.1). Lastly, FOOL treats the reconstruction model rec as a decoder and assigns the identity function as its teacher, i.e., $\mathcal{H}_{rec}(x) = x$. Unlike for other decoders, the target representation must be human-interpretable. Hence, we train the decoder for image reconstruction using the *Charbonnier Loss* [CBFAB94]

$$\mathcal{L}_{rec} = \sqrt{\|\mathbf{x} - rec(enc(\mathbf{x}))\|^2 + \epsilon^2} \tag{5.11}$$

where ϵ is a small constant we set as $2 \cdot 10^{-3}$. It is out of this work's scope to exhaustively evaluate image restoration methods. Rather, the focus is to provide empirical evidence

for the claims in Section 5.1.3. We simply found that, despite performing comparably to other sums of error losses on benchmark metrics, using Charbonnier results in more stable training.

5.2.2 The FOOL's System Design

Compression and Prediction Request Flow

Figure 5.6 illustrates a high-level view for serving requests.

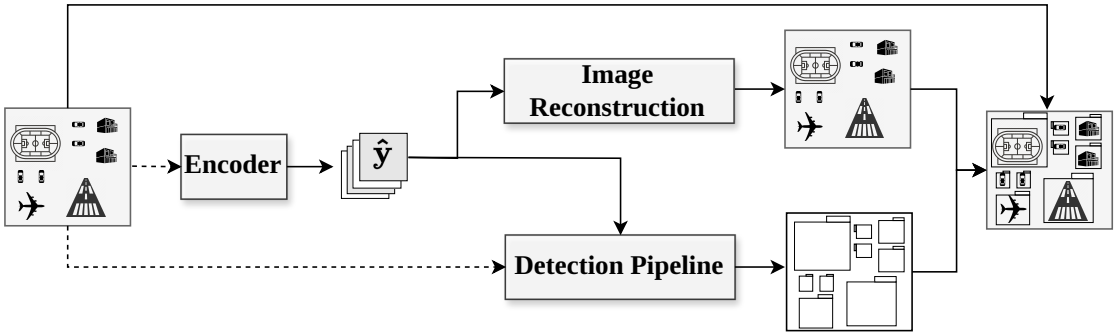


Figure 5.6: High-Level Inference Request Flow

For samples processed by FOOL, there is a single encoder. The output $\hat{\mathbf{y}}$ is forwarded to the detection pipeline, skipping the shallow layers. The detection pipeline for a single forward pass consists of a decoder, backbone, and predictor. There may be multiple backbones the client can choose from, and each backbone may have multiple predictors. A decoder transforms $\hat{\mathbf{y}}$ into an input representation for a particular backbone. A predictor outputs bounding boxes for a specified task. An image reconstruction model optionally restores the latent to a human-interpretable image to overlay the bounding boxes. Samples downlinked with bent pipe or some image codec are forwarded to the shallow layers, skipping the corresponding decoder and reconstruction model. This section focuses on the pipeline, before Section 5.2.1 introduces the compression method.

Profiling Compression Pipelines for OEC

A common challenge for operators is to determine whether reported performance regarding resource usage or throughput from the latest advancements generalizes to their target hardware. This problem stems not from a lack of rigor by authors but from the sheer heterogeneity of the AI accelerator landscape [RMJ+22]. Graph compilers and other vendor-specific optimizations discussed in Chapter 3 further complicate evaluation, with varying methods for operator fusion, graph rewriting, etc. Consequently, FOOL includes a simple profiling and evaluation strategy that operators may run before deployment. Notably, in contrast to existing work that partitions images to match the input size of a particular application, the profiler regards the importance of spatial dimensions for resource efficiency. The purpose of the profiler is to determine a configuration that

maximizes throughput. While throughput evaluation is straightforward, how to measure it (e.g., images/second) is not necessarily obvious, particularly for (neural) compression pipelines.

First, terrestrial and LEO remote sensing with constrained sensor networks demand resource-conscious methods, but in LEO, downlinks are only available within segments. Due to memory and storage constraints, devices must process samples according to a sensor rate, i.e., a prolonged interval between incoming samples. Hence, the objective in LEO is to maximize the number of pixels the accelerator can process before reaching a downlink segment, given a time constraint for a single sample (i.e., "frame deadline" [DCC+23]). For example, assume a cheaper and a costlier compression model where both models meet the frame deadline. Applying the latter results in half the bitrate but thrice the inference time. Using the former is beneficial in most network conditions for real-time terrestrial applications since it results in a lower end-to-end request latency. In contrast, applying the latter in LEO may be advantageous, as finishing earlier results in the needless idle time of resources. Second, satellite imagery has substantially higher resolution than captures from most terrestrial sensor networks. A standard method to improve throughput for high-dimensional images is parallel processing with tile partitioning. The distinction is that there is more control over the spatial dimensions and the batch size. Nonetheless, a caveat is the friction between a model's size and the input size. Increasing the width (e.g., the number of feature maps output by a convolutional layer) of a neural codec's parametric transforms may result in better compression performance but lower processing throughput. In summary, we require a measure that includes (i) the tile spatial dimensions, (ii) batch size, and (iii) the capacity-compression performance trade-off.

We can address the requirements (i) and (ii) by measuring throughput as *pixels processed per second* (PP/s). To motivate the need to expand on PP/s for (iii), we demonstrate the friction between model width, input size, and batch size using the convolutional encoder in [FRD24], consisting of three downsampling residual blocks (Section 5.2.1). Figure 5.7 summarizes the results as the average of 100 repetitions with progressively increasing width. Notice how evaluating img/s always favors smaller spatial dimensions and disregards batch size and model width. In contrast, PP/s reveals that the optimal spatial dimension is around 500×500 but will naturally favor smaller models, as it does not consider that wider models may reduce transfer costs. To alleviate the limitations of PP/s, we measure *Transfer Cost Reduction per Second* (TCR/s) as:

$$\mathrm{TCR/s} = \underbrace{\frac{\text{Image Dimension}}{\text{Seconds per Batch}}}_{\mathrm{PP/s}} \times (\mathrm{bpp}_{\mathrm{raw}} - \mathrm{bpp}_{\mathrm{codec}}) \tag{5.12}$$

The measure now includes the compression performance as the difference between the expected bits per pixel (bpp) of compressed ($\mathrm{bpp}_{\mathrm{enc}}$) and uncompressed ($\mathrm{bpp}_{\mathrm{raw}}$) sensings. The raw bpp value refers to the bit depth, i.e., the sensor's radiometric resolution and the number of bands. For example, the radiometric resolution of Sentinel-2 is 12 bits [DDC+12], so for three bands $\mathrm{bpp}_{\mathrm{raw}} = 3 \cdot 2^{12}$. The advantage of TCR/s is twofold.

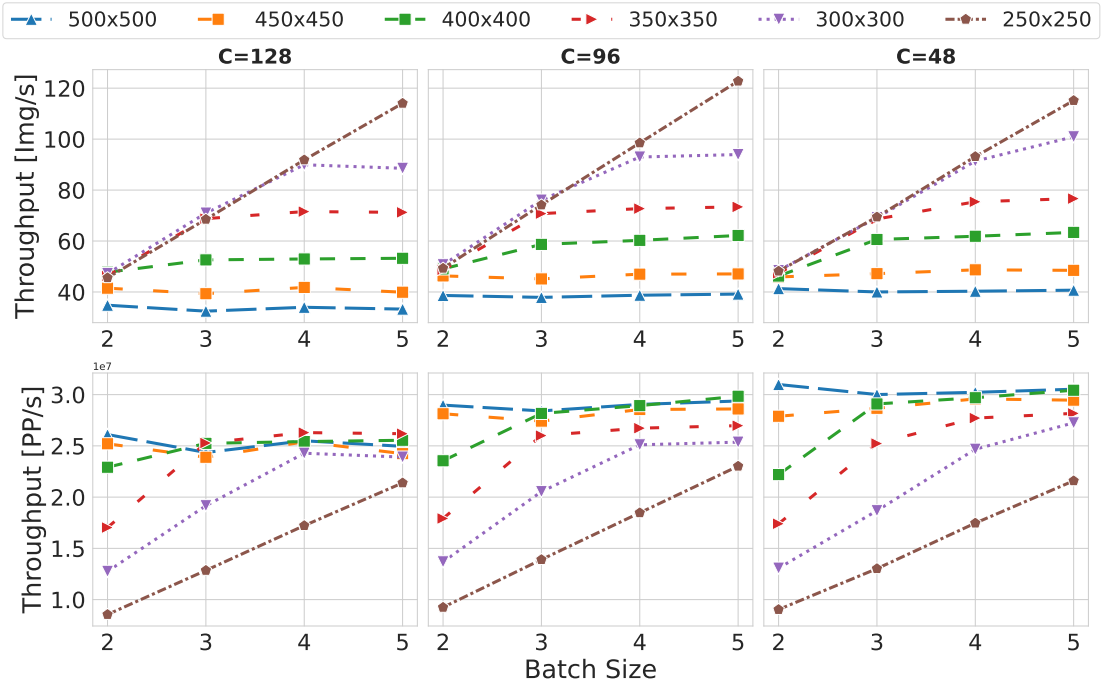


Figure 5.7: Contrasting Throughput Measures

First, decoupling it from system-specific parameters, such as sensor resolution or orbital period, permits drawing generalizable insights regarding the relative trade-off between codec overhead and bitrate reduction. Second, operators can still assess the feasibility of the pipeline on target hardware and the expected downlinkable data volume by running the profiler with configurations that reflect deployment conditions.

Concurrent Task Execution

So far, this section has solely discussed the computational cost of a codec's parametric transforms without considering pre- and post-processing. In particular, after applying the encoder transforms, it is still necessary to entropy code the output to compress the latent. Since FOOL's entropy model is input adaptive, it requires a range coder. Although more recent range coders are efficient, they incur non-negligible runtime overhead. Therefore, given the unforgiving conditions of OEC, we argue that the entropy coder cannot be neglected in the design process and evaluation of a neural codec. FOOL virtually offsets the entire runtime overhead with simple concurrent task execution. The idea is to exploit the minimal interference of processes drawn from different resource types. For three sequentially incoming samples x_{i-1}, x_i, x_{i+1}, FOOL executes CPU-bound pre-processing of x_{i+1}, accelerator-bound inference of x_i, and CPU-bound post-processing of x_{i-1}. In this work, pre-processing corresponds to tiling the samples, and post-processing to entropy coding with rANS [Dud14, Tow20]. Concurrently to inference x_{i-1} on the accelerator, a

process starts tiling x_i. After inference on x_{i-1}, $\hat{\mathbf{y}}, \hat{\mathbf{z}}, \hat{\boldsymbol{\sigma}}, \hat{\boldsymbol{\mu}}$ (Section 5.2.1) are persisted on the file system. Then, a separate process loads the data and losslessly compresses $\hat{\mathbf{y}}, \hat{\mathbf{z}}$ with an entropy coder. We expect minimal interference between the processes, resulting in virtually no PP/s decrease.

5.2.3 Evaluation

Experiment Design

Our experiments reflect our aim to determine (i) the compression performance on aerial and satellite imagery without relying on prior knowledge and (ii) the feasibility of orbital inference.

Testbed We benchmark [RRP+22] on an analytic and trace-driven [RRFD23] simulation based on results from a physical testbed with hardware summarized in Table 5.1. The power consumption is capped at 15W for the entire testbed. Our

Table 5.1: Testbed Device Specifications

Device	CPU	GPU
Ground Server	16x Ryzen @ 3.4 GHz	RTX 4090
Edge (Nano Orin)	6x Cortex @ 1.5 GHz	Amp. 512 CC 16 TC
Edge (TX2)	4x Cortex @ 2 GHz	Pas. 256 CC
Edge (Xavier NX)	4x Cortex @ 2 GHz	Vol. 384 CC 48 TC

simulation replicates a configurable CubeSat by imposing energy, memory, and bandwidth constraints. To simulate the downlink bottleneck with varying link conditions, parameterize link conditions and data volume (Section 5.1.3) using real-world missions [DKL+17, CCA+21, DBB+12, Nat24] as summarized in Table 5.2. Due to the orthogonality of compression to systems-related challenges in OEC, we argue a focused simulation yields more insight results than running a full OEC simulator (e.g., [DL20]). Our intention is for FOOL to facilitate OEC as an auxiliary method. Therefore, we demonstrate the bitrate reduction and resource usage trade-off for various configurations representing the heterogeneity of available compute resources and nanosatellite constellations.

Table 5.2: Constellation Link Conditions

Operator	Constellation	Link	Rate (Mbps)	Pass (s)	Data Per Pass (Gb)
Planet	Dove (3P/B13)	HSD 1	160	510	12.0
Maxar	WorldView	WorldView-3	1200	600	90
ESA	Copernicus	Sentinel 3A/B	560	600	40.0
NASA	Landsat	Landsat 8	440	120	39.6

Third-Party Detection Models & Target Tasks FOOL derives the basic approach to accommodate multiple backbones with a single encoder (Section 5.2.1) from FrankenSplit [FRD24]. Foundational models (i.e., feature extractors or backbones) are interchangeable third-party components in SVBI. To complement previous work (Section 5.1.2) and further show the flexibility of SVBI, we focus on modern YOLO variants [JCQ23]. Figure 5.8 illustrates the pipeline to represent third-party detectors we prepare before evaluating codecs.

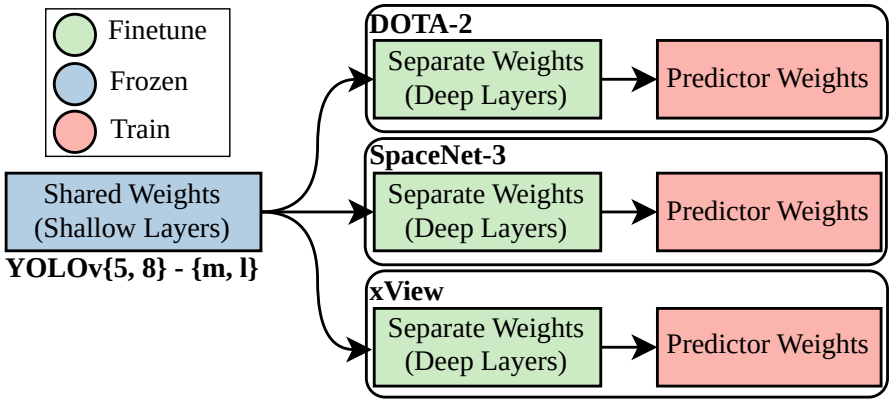


Figure 5.8: Detection Pipeline for Evaluation

While the work in [FRD24] did not explicitly evaluate object detection tasks, the support for two-stage detectors follows from the codec sufficiently approximating the representation of the feature extractor (i.e., the first stage). However, it is not apparent whether the general SVBI framework yields gains over image codecs when the targets are one-stage detectors. Therefore, to replicate a representative service for inference on aerial or satellite imagery, we apply simple transfer learning on open-source weights [JCQ23] for YOLOv5 and YOLOv8. Image codecs pass a sample $dec(enc(\mathbf{x})) = \hat{\mathbf{x}}$ to the input layers of a target model. Feature codecs (i.e., SVBI methods) skip the shallow layers and pass $dec(enc(\mathbf{x})) = \hat{\mathbf{h}}$ to the deeper layers. Detection models with the same architecture share the frozen shallow layers (i.e., layers until the first non-residual connection). We associate one task as outlined by the test labels for each of the three dataset separately. The tasks represent varying mission conditions. DOTA-2 [DXX+21] for a more coarse-grained aerial task with comparatively lower Ground Sample Distance (GSD) and larger objects. SpaceNet-3 [VELB18] for urban tasks (e.g., for traffic control) with high image resolutions. Lastly, xView [LKM+18] for disaster response systems where detection models rely on fine-grained details. Lastly, image reconstruction is treated distinctly as single task for the reconstruction model, and not the detection pipeline, by combining the images from the three test sets.

To simplify the already intricate evaluation setup and to ease reproducibility, we deliberately refrain from more refined transfer learning methods. We merely require detection models with mAP scores that are moderately high to determine whether a codec can preserve fine-grained details for EO tasks on satellite imagery. For each architecture,

we jointly finetune the deeper layers and train separate predictors that achieve around 35-65% mAP@50.

Training & Implementation Details To demonstrate that FOOL can handle detection tasks without relying on prior information (Section 5.1.3), we do not optimize the compression model with the training set of the prediction tasks (i.e., DOTA-2, SpaceNet-3, xView). Instead, we curate other aerial and satellite datasets [Cla23, BVR19, HAL+22, VEH21, SHB+20, SHVE+21, BS23, RCS+21] that cover region and sensor diversity. SVBI does not rely on labels, i.e., replacing the curation with any diverse enough dataset from satellite imagery providers (e.g., Google Earth Engine) should be possible.

We train one separate compression mode for each third-party detector using the shallow layers as teachers and verify whether the rate-distortion performance is comparable. Then, we freeze the encoder of the compression model for YOLOv5-L and discard all other encoders. Lastly, we freeze the remaining encoder's weights and (re-)train the separate decoders to demonstrate clients may request inference on variations (YOLOv5-M) or newer models as they emerge (YOLOv8).

We fix the tile resolution to 512×512 during training. We load samples as a video sequence for FOOL by grouping tiles from the same image in partitioning order with random transformations to fill any remaining spots. After training, the tensor shape (i.e., the number of tiles and the spatial dimensions) may vary for each separate sample. We use PyTorch [AYH+24], CompressAI [BRFP20], and pre-trained detection models from Ultralytics [JCQ23]. To ensure reproducibility, we use torchdistill [Mat21]. We use an Adam optimizer [KB14] with a batch size of 8 and start with an initial learning rate of $1 \cdot 10^{-3}$, then gradually lower it to $1 \cdot 10^{-6}$ with an exponential scheduler. We first seek a weight for the rate term in Equation (5.10) that results in lossless prediction with the lowest (best) bpp. Then, we progressively increase the term weight to evaluate trade-offs between rate and predictive loss.

Datasets Preparation The train sets for third-party detectors and the train sets for the compression models are strictly separated. However, we create square tiles for all datasets by partitioning the images with a configurable spatial dimension and applying 0-padding where necessary. We extract bands from samples corresponding to RGB and convert them to 8-bit images, as to the best of our knowledge, there are no widespread open-source foundational models for detection with multispectral data yet. To ease direct comparisons, we convert the network detection labels of SpaceNet-3 by transforming the polygonal chains into bounding boxes. Lastly, since there are no publicly available labels for the xView and SpaceNet test sets, we create a 9:1 split on the train set.

Compression Performance Measures To evaluate how codecs impact downstream task performance, we measure *Predictive Loss* as the drop in mean Average Precision (mAP) by inputting decoded samples. We regard a configuration to result in *lossless prediction* if there is less than 1% difference in expected mAP@50. We confirm the

Table 5.3: Summary of Codec Parameter Distribution

Codec	Pars. Total	Pars. Encoder	Pars. Decoder	Shared
FOOL-L	4.97M	1.19M	4.06M	0.28M
FOOL-M	4.33M	0.69M	3.83M	0.16M
FOOL-S	3.91M	0.35M	3.66M	0.08M
SVBI-L	4.99M	1.25M	4.39M	0.65M
SVBI-M	4.35M	0.72M	3.91M	0.29M
SVBI-S	3.95M	0.38M	3.71M	0.16M
FP	7.03M	3.51M	3.51M	0.02M
MSHP	17.56M	14.06M	11.66M	8.15M
JAHP	25.50M	21.99M	19.60M	16.10M
TinyLIC	28.34M	21.23M	19.16M	12.05M

observations from [FRD24] where the initial teacher only negligibly affects compression performance, and the predictive loss by a codec is comparable across target models (i.e., the retained information in shallow layers is similar across YOLO variations). Hence, for brevity, we aggregate the compression performance for each task separately, taking the highest predictive loss incurred on a detection model. We train the image reconstruction model using the same configurations as [LCS+21], and compare it with LIC models using common measures (PSNR, MS-SSIM, LPIPS [ZIE+18]).

Baselines We consider seminal work for image codecs as baselines with available open-source weights. Factorized Prior (FP) [BLS17] as a relatively small model without side information. (Mean-)scale hyperprior (SHP, MSHP) [BMS+18, MBT18] for drawing comparisons to side information in LIC, and Joint autoregressive and hierarchical priors (JAHP) [MBT18] that further improves compression performance with an autoregressive context model. Lastly, TinyLIC [MCJ+24] represents recent work on efficient LIC design with state-of-the-art rate-distortion performance. Table 5.3 summarizes parameter distributions between encoder and decoder components from LIC models.

To draw comparisons to existing work on SVBI, we use FrankenSplit [FRD24] *without* saliency guidance as the baseline (BSVBI). The encoder consists of stacked residual blocks (Section 5.2.1), and the decoder is instantiated from a YOLOv5+ blueprint (C3 blocks [JCQ23]). We scale the capacity of BSVBI by including side information from LIC (MSHP) and increasing the width and depth to match the various FOOL configurations. We train FOOL and BSVBI with the same dataset and training parameters (Section 5.2.3).

Rate Trade-off with Predictive Loss

We report the predictive loss as a percentage point difference using mAP@50 on foundational detection models.

Comparison to Image Codecs Figure 5.9 illustrates the trade-off between bpp (left is better) and predictive loss (top is better) for LIC models on each task separately. We

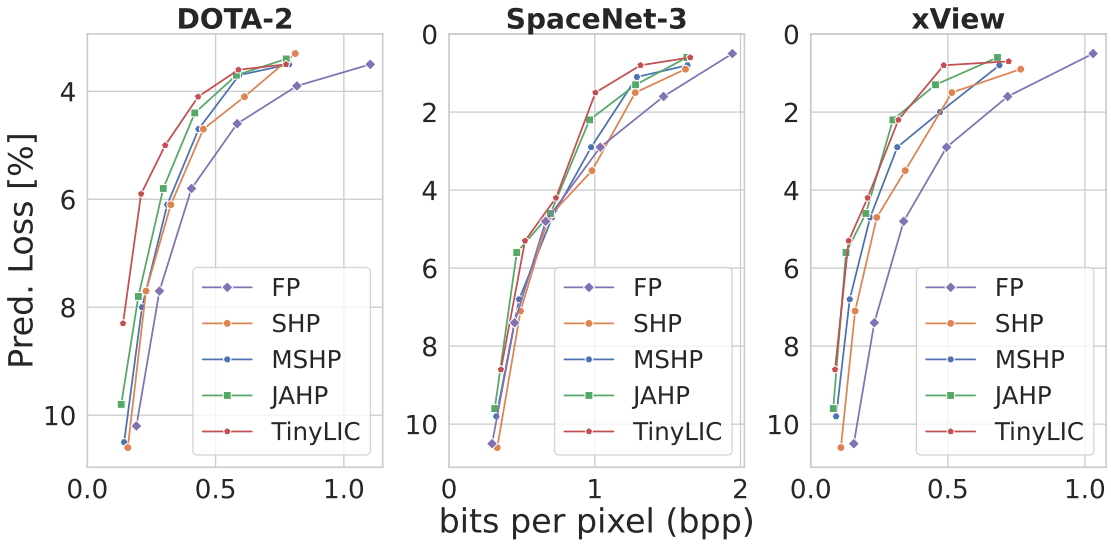


Figure 5.9: Compression Performance Image Codecs

primarily focus on how FOOL compares to existing SVBI to draw new insights from the novel additions and confirm that our results on aerial and satellite imagery datasets with one-stage detectors are consistent with previous findings on standardized terrestrial datasets in Chapter 4.

Comparison to Feature Codecs Figure 5.10 contrasts the trade-off between bpp and predictive loss for FOOL and BSVBI with progressively increasing sizes. The efficacy of compressing shallow features is best shown by comparing BSVBI and FOOL to MSHP, as they rely on the same entropy model. The highest quality MSHP model results in about 3-4% predictive loss for DOTA-2. In contrast, the highest quality BSVBI-S model has 37x fewer encoder parameters but results in half the bitrate with no predictive loss. Despite BSVBI demonstrating strong compression performance, FOOL significantly outperforms BSVBI across all configurations. FOOL-S has a 51% lower bitrate for configurations with lossless prediction than the comparatively large BSVBI-L. Relative to the FOOL model with matching capacity (FOOL-L), BSVBI-L has twice the bitrate.

Ablation Study We consider BSVBI an ablation, as FOOL extends BVSBI's architecture by placing 3D attention layers between the residual blocks and a cross-attention layer to include context. The auxiliary networks h_a and h_s (Section 5.2.1) are identical for FOOL and SVBI, i.e., three stacked residual blocks. Additionally, we perform ablation studies to assess by-component improvement and summarize the results for lossless predictions in Table 5.4.

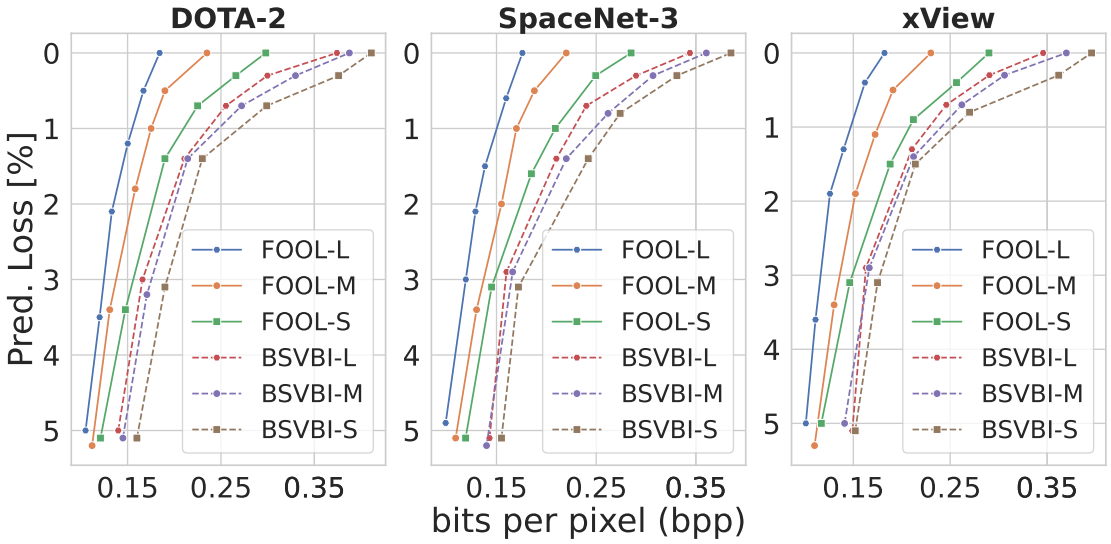


Figure 5.10: Compression Performance Feature Codecs

Table 5.4: Ablations Comparisons for lossless Prediction

Model	DOTA-2 (bpp ↓↓)	SpaceNet-3 (bpp ↓↓)	xView (bpp ↓↓)
FOOL-L	0.1843	0.1760	0.1822
FOOL-M	0.2110	0.1993	0.2032
FOOL-S	0.2389	0.223	0.2290
BSVBI-L	0.3622	0.3440	0.3452
BSVBI-M	0.3775	0.3605	0.3699
BSVBI-S	0.3889	0.3852	0.3909
NITA-L	0.2209	0.1993	0.2032
NITA-M	0.2287	0.2193	0.2205
NITA-S	0.2433	0.2386	0.2407
NKPC-L	0.2839	0.2712	0.2768
NKPC-M	0.2926	0.2855	0.2883
NKPC-S	0.3116	0.3029	0.3112

The NITA models include the keypoint context without the inter-tile attention (ITA) layers. Analogous to BSVBI, we replace attention layers with residual blocks and match corresponding model sizes by increasing the depth and width of NITA models. NKPC-Ablation drops components for embedding keypoints, i.e., it only includes the IT attention layers.

The results show that relative to BSVBI, the task-agnostic context component contributes considerably more to rate reductions than the ITA layers that leverage inter-tile spatial dependencies. Still, we argue that the NTI-layers fulfill their purpose, to synergize with the partitioning strategy that maximizes processing throughput (Section 5.2.4).

5.2.4 Image Reconstruction Quality

We aim to demonstrate the feasibility of recovering presentable images from the compressed latent space of shallow features. We average results on DOTA-2, SpaceNet-3, and xView to reduce the bloat of reporting similar values summarize the results in Table 5.5.

Table 5.5: Comparison Between Recovery and Image Codecs

Model	PSNR↑↑	MS-SSIM↑↑	LPIPS↓↓	BPP↓↓	Pred. Loss↓↓
FOOL	36.51	15.43	0.1700	0.1808	-
FOOL-FT	35.56	14.57	0.1480	0.1808	-
FP-HQ	43.22	25.07	0.0896	1.0470	1.500
FP-MQ	35.56	16.55	0.2498	0.3200	7.508
MSHP-HQ	43.90	25.20	0.0841	1.0370	1.711
MSHP-MQ	36.45	16.83	0.2361	0.2787	7.180
JAHP-HQ	43.95	25.17	0.0818	1.0297	1.504
JAHP-MQ	36.61	16.95	0.2303	0.2641	6.397
TinyLIC-HQ	44.52	25.07	0.0683	1.0473	1.602
TinyLIC-MQ	37.42	17.27	0.2102	0.2899	5.499

For transparency, we exclusively select samples from the lower quartile across all measures to qualitatively showcase the reconstruction. HQ refers to the weights with the highest available quality, and MQ refers to mid-quality weights that roughly match FOOL in PSNR. FOOL-FT finetunes the reconstruction model for an additional $2.5 \cdot 10^5$ iterations using LPIPS [ZIE+18]. Unsurprisingly, the LIC models achieve significantly better scores across all reconstruction measures (i.e., PSNR, MS-SSIM, and LPIPS). The advantage of FOOL is that it has a considerably lower bitrate with no predictive loss on tasks for which it had no prior information. Nonetheless, the results are considerably more interesting when contrasting FOOL to LIC models with mid-quality weights. Notice how FOOL matches reconstruction measures at no predictive loss and a 46-77% *lower* bitrate. Note that we did not find that the dataset significantly influences rate-distortion performance, except for a slight reduction in predictive loss (verified by training an FP model on the curation using the same setup as in [BRFP20]). Compression is a low-level vision task that generalizes well but may lack domain specificity when applying a standard rate-distortion reconstruction loss. In other words, the objective is the decisive difference between the SVBI and the NIC models. To provide some intuition to the LPIPS measure, we select an image where FOOL-FT achieves considerably lower PSNR than TinyLIC and contrast the results in Figure 5.11.

Notice that the quality differences are most visible with fine-grained details, i.e., compared to TinyLic-HQ, TinyLic-MQ has a noticeable blur with some shadows completely missing in the bottom left. FOOL-FT preserves such details, despite lower PSNR, and this increase in perceptual quality is reflected in the LPIPS score. Figure 5.12 further visualizes the potential of reliably recovering fine-grained information from the compressed representation.

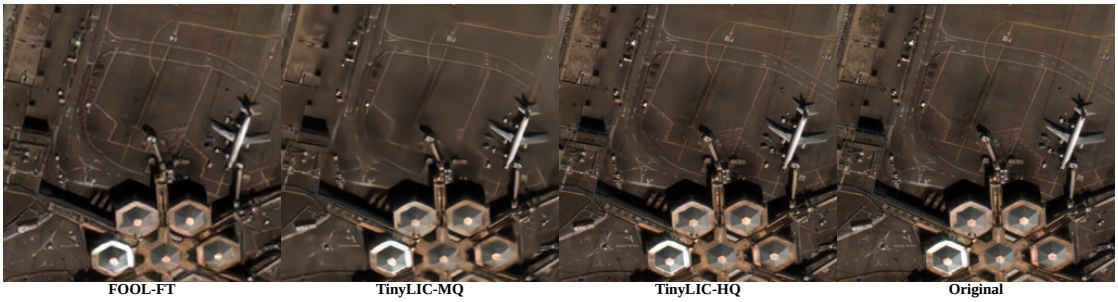


Figure 5.11: Visual Comparison between FOOL Image Recovery and a State-of-the-Art LIC model

Figure 5.12: Showcasing Potential of Recovery from Compressed Features with fine-tuning for Perceptual Quality using LPIPS

Naturally, it should be possible to fine-tune the TinyLic-MQ to improve perceptual quality analogous to FOOl-FT. However, TinyLIC is still a significantly costlier model, with a worse rate and prediction performance. More pressingly, we stress that the *reliability* of a restoration model is bound by the available signals in the compressed latent space. Accordingly, we deliberately avoid generative models that prioritize realism over structural integrity. Prioritizing realism over reliability defeats the primary purpose of image restoration, i.e., intervention by human experts in critical EO applications. A model outperforming experts does not imply that predictions may inexplicably be false. In particular, where human cost is involved, such as disaster warning or relief [TD22], it is paramount that experts can trust the codec to not include extrapolated elements in an image.

We argue that our results adequately underpin the statements in Section 5.1.3 and Section 5.1.3. In summary, if the salient regions align, compressing for model prediction requires more information than for human observation. Task specificity determines rate savings and not an entity's input interface. Targeting shallow features is minimally task-specific by relaxing the objective for lossless prediction on all possible tasks to those valuable for clients.

System Performance

The following evaluates FOOL's resource usage and how well it can address the downlink bottleneck. The methodology resembles how the system aids operators in determining the correct model size for a target device and estimating the increase in data volume relative to bent pipes. We do not apply vendor-specific optimization (e.g., TensorRT) to ensure transparent evaluation and keep the results reasonably platform agnostic. Instead, we instantiate all models dynamically with half-precision in the native PyTorch environment (`torch 1.14.0` with `CUDA 11.4.315`). Image codecs are omitted for conciseness, as even the state-of-the-art for efficient LIC design still runs considerably slower than the largest SVBI models.

Processing Throughput and Transfer Cost Reduction We manually step through parts of the profiler (Section 5.2.2) for evaluation and to show how it estimates gains in downlinkable data volume. Consider the results from measuring the friction between model sizes and input dimensions on processing throughput in Figure 5.13. Processing throughput grows with frequency and decreases with model size, showing compute-bound behavior. Each device has a batch-size-dependent optimum around 6, reflecting GPU utilization vs. memory constraints. The Jetson Orin achieves the highest throughput, TX2 the lowest, with consistent scaling across all models. The drops between even and uneven batch sizes reflect discrete transitions in GPU kernel scheduling efficiency, for example, due to occupancy saturation boundaries. Notably, the processing throughput gain of FOOL-S over FOOL-M is significantly higher than FOOL-M over FOOL-L, despite FOOL-M having a comparable size difference to both models.

Table 5.6 summarizes the configuration that maximizes profiler selection by TCR/s for all models on each device separately. Since bitrate variance is low between DOTA-2, SpaceNet-3, and xView, we average the bpp (Section 5.2.3) on the validation sets. The bold Model value indicates the adequate size of each model family on a device, i.e., the model we will deploy to measure data volume downlinking in the following experiments. The bold TCR/s marks the highest overall value for a device, i.e., we can expect applying FOOL over BSVBI to result in considerably more downlinkable data on all devices. However, due to keypoint extraction and the ITA layers, FOOL's processing throughput is slower than that of BVSBI. The overhead is particularly punishing for the most constrained device (i.e., the previous-generation TX2), where BSVBI-M has slightly higher TCR/s than FOOL-M despite the latter's significantly better compression performance. Moreover, the profiler selects FOOL-S over FOOL-M/-L for the low-end current generation Orin Nano and high-end last-gen past generation NX. Conversely, the profiler decides on the mid-sized model for BSVBI across all devices despite FOOL's compression performance scaling better.

Still, we argue that the results accentuate the findings from Section 5.2.3. Notice the contrast between TX2 and Nano Orin. One hardware generation was sufficient for the lowest-end device in the Jetson lineup to see a *threefold* increase in TCR/s on FOOL-L over the last-generation midrange device. Thus, it is reasonable to claim that FOOL

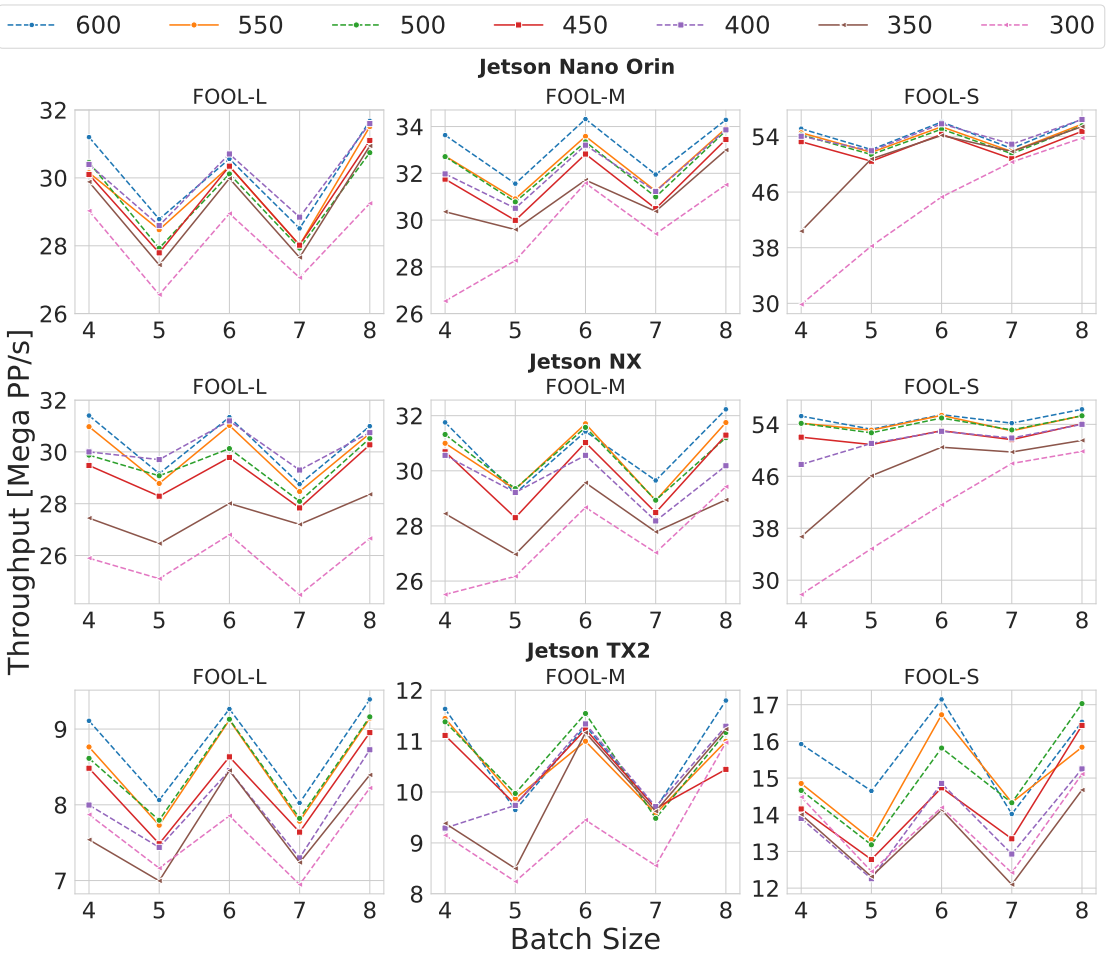


Figure 5.13: Processing Throughput by Model Size

can (i) adequately leverage the current rapid progression of energy-efficient hardware improvement (i.e., with FOOL-M, L, and potentially larger variants) and (ii) is flexible enough to be deployed on more constrained devices using the small FOOL-S that still achieve substantial rate reduction.

Model Inference with Concurrent Task Execution The following examines the claim in Section 5.2.2, i.e., whether FOOL's compression pipeline can offset the runtime overhead of entropy coding. In other words, we evaluate whether the interference between *concurrent* GPU and CPU-bound processes is negligible enough. We assume the worst case for interference, i.e., the CPU-bound processes run continuously and concurrently by keeping them busy from an additional data stream when necessary. As all three devices have multicore CPUs and a dedicated GPU, we report results on the Nano Orin for brevity.

Table 5.6: Throughput Comparison Between Feature Codecs

Device	Model	Spatial Dimensions	Batch Size	TCR/s ↑↑
Orin Nano	FOOL-L	600x600	4	$7.4794 \cdot 10^8$
	FOOL-M	600x600	8	$7.6694 \cdot 10^8$
	FOOL-S	600x600	8	$1.3386 \cdot 10^9$
NX	FOOL-L	600x600	8	$7.5456 \cdot 10^8$
	FOOL-M	600x600	6	$8.1664 \cdot 10^8$
	FOOL-S	600x600	8	$1.3422 \cdot 10^9$
TX2	FOOL-L	600x600	8	$2.3696 \cdot 10^8$
	FOOL-M	600x600	8	$2.8067 \cdot 10^8$
	FOOL-S	600x600	6	$4.0751 \cdot 10^8$
Orin Nano	BSVBI-L	600x600	8	$6.1419 \cdot 10^8$
	BSVBI-M	600x600	5	$7.2504 \cdot 10^8$
	BSVBI-S	550x550	7	$6.9919 \cdot 10^9$
NX	BSVBI-L	600x600	8	$6.0007 \cdot 10^8$
	BSVBI-M	600x600	6	$7.3717 \cdot 10^8$
	BSVBI-S	600x600	6	$7.0720 \cdot 10^9$
TX2	BSVBI-L	600x600	7	$1.7861 \cdot 10^8$
	BSVBI-M	600x600	6	$2.9978 \cdot 10^8$
	BSVBI-S	600x600	7	$2.6818 \cdot 10^8$

Table 5.7: Concurrent Entropy Coding and Effect on TCR/s

Model	TCR/s	TCR/s dec.	File Size (MB)	File/s	rANS (MB/s)
FOOL-L	$7.26 \cdot 10^8$	**2.94%**	0.616	29	37.2
FOOL-M	$7.57 \cdot 10^8$	1.28%	0.462	30	38.3
FOOL-S	$1.32 \cdot 10^9$	1.06%	0.383	52	38.8
BSVBI-L	$5.96 \cdot 10^8$	**2.88%**	0.822	37	37.6
BSVBI-M	$7.15 \cdot 10^8$	1.37%	0.617	42	39.6
BSVBI-S	$6.91 \cdot 10^8$	1.28%	0.437	58	40.1

Table 5.7 summarizes the results from running the entire compression pipeline with concurrent task execution using the configurations that maximize TCR/s from Table 5.6. The bold values in the TCR/s dec. column indicates the size with the highest decrease. A file includes all model artifacts output by the neural codec's ANN components for a single tile, i.e., the pipeline still needs to entropy code them to match the bpp in TCR/s calculations. File size refers to the storage requirements *per tile* of the encoder output tensors, i.e., the data volume the rANS process encodes. We compute file size by a worst-case upper bound by the encoder output tensor dimensionality (Section 5.2.1) without serialization formats that could exploit the sparsity of $\hat{\mathbf{y}}$ and $\hat{\mathbf{z}}$. There are two essential findings from the results. First, the rANS process can consume tasks considerably faster than the inference process can produce them, i.e., there is no risk of

backpressure within the pipeline. Second, there is only a minimal percentage decrease in TCR/s across all devices and models relative to sequential execution. Hence, we argue that the pipeline successfully offsets the runtime overhead as claimed in Section 5.2.2 even without relying on a precomputed lookup table (e.g., tANS in ZSTD [CK18]). The results are unsurprising when viewing the CPU and GPU load of ANN inference without CPU-bound concurrent tasks in Figure 5.14.

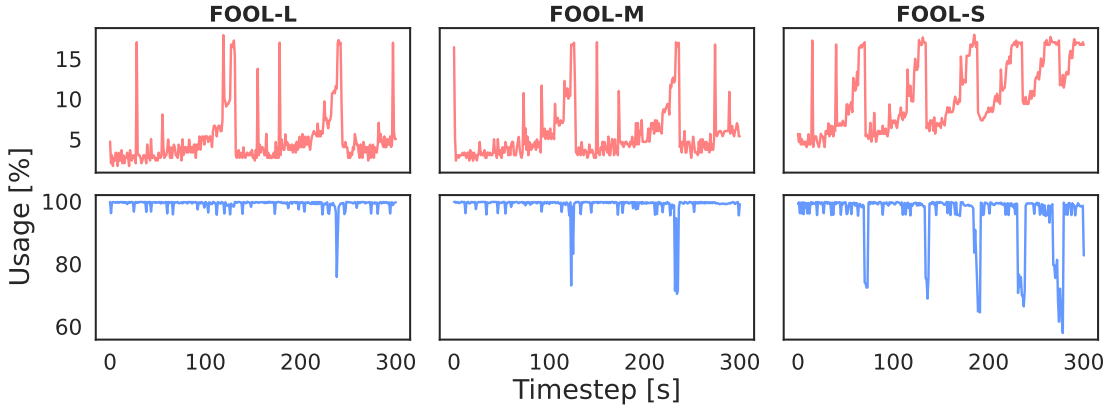


Figure 5.14: CPU (red) and GPU (blue) Usage of Encoder Network

Since inference is GPU-bound, CPU usage is low even when the GPU is under maximal load. Contrast this with the CPU and GPU usage in Figure 5.15 where we monitor [RRP+22] usage while running the entire pipeline with the two concurrent processes.

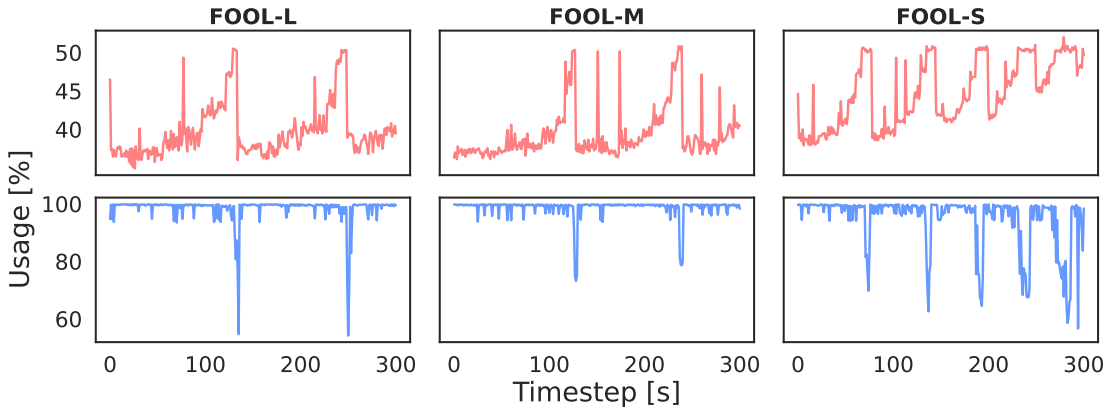


Figure 5.15: CPU (red) and GPU (blue) Usage of Concurrent Pipeline

If the CPU-bound processing task were to interfere with the ANN execution, resource usage should reveal frequent drops in GPU load. Comparing FOOL-L to S and M reveals some dependency between ANN size and CPU usage. For FOOL-L, two discernible drops

in GPU usage suggest some interference, which may explain the 2.9% decrease in TCR/s for FOOL-L and BSVBI-L. In contrast, there is no noticeable pattern difference in GPU load between Figure 5.15 and Figure 5.14 for S and M variants, explaining the negligible 1-1.5% TCR/s drop.

Downlinkable Data Volume We now compare how methods can alleviate the downlink bottleneck using the traces from previous experiments. Figure 5.16 visualizes the transferable volume per downlink pass.

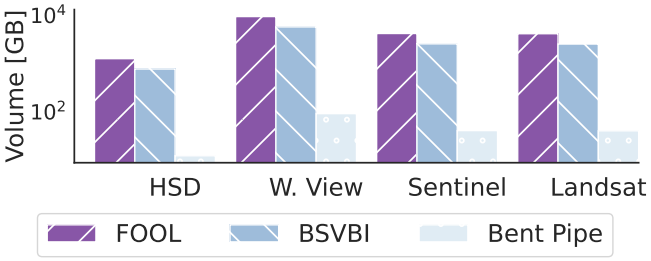


Figure 5.16: Downlinkable Data Volumes by Link

Notice the logarithmic scale, i.e., FOOL improves downlinking using bent pipes by over two orders of magnitude without relying on prior information on the downstream tasks or crude filtering methods. For example, given Maxar's WorldView-3 conditions [CCA+21], it would be possible to downlink roughly 9TB of sensor data per pass before reaching downlink saturation. As a comparison, the state-of-the-art filtering method in [DCC+23] reports a 3× improvement based on a definition of value. Note that to provide a realistic presentation of the opportunities SVBI provides, we assume that a nanosatellite processes tiles until reaching a downlink segment. Moreover, we disregard the "computational deadline", i.e., it can process all the data before reaching a ground segment. This is reasonable since there should always be enough data to process. If not produced by a single sensor, constellations may designate certain satellites as compression nodes using reliable, high-capacity local communication channels [Mit20]. Further, it is inferable that even the low-end current-generation Orin Nano without any vendor-specific optimization would barely miss the computational deadline.

Energy Consumption and Savings The following investigates the energy usage of the selected model for each device. As the GPU and CPU usage patterns are highly similar, we measure by the time it takes until a method can *double* the downlinkable data. For example, if only downlinking 40 GB is possible with the unprocessed data, then we measure energy cost until the encoded size corresponds to 80 GB of raw captures. Figure 5.17 summarizes the results.

As expected, processing on TX2 requires more energy than on NX and Orin Nano as it is slower. Somewhat interesting is that the NX consumes more energy than the Orin Nano.

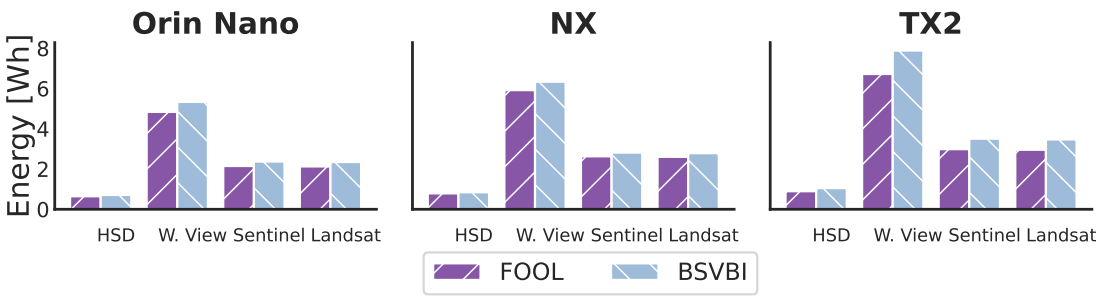


Figure 5.17: Energy Cost of Compression Pipelines

As they execute the same models with comparable processing throughput, the results suggest that the newer Jetson lineup is more energy-efficient. Lastly, we measure savings from reduced transmission time, arguably an often undervalued advantage of compression. Admittedly, satellites will downlink as bandwidth permits, i.e., the transmission energy cost does not depend on the codec performance when there is saturation. Nonetheless, to intuitively show the amount of energy large volumes might require, we contrast with bent pipes in Figure 5.18.

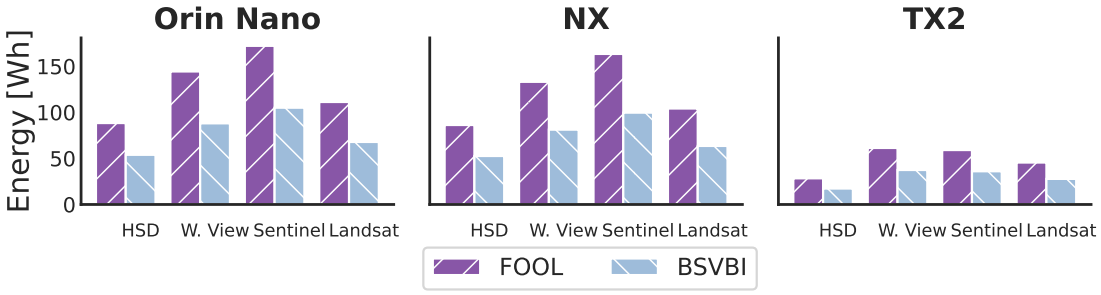


Figure 5.18: Potential Energy Savings from Transmission

Given the link conditions, we measure the difference in energy cost between transmitting until saturation and transmitting the corresponding raw volume from Figure 5.16.

5.3 Summary

This chapter introduced a novel compression method that addresses the downlink bottleneck in LEO without relying on prior knowledge of downstream tasks. A rigorous evaluation showed that FOOL increases data volume with advancements that, to the best of our knowledge, are unprecedented. The rate reductions are primarily from the task-agnostic context, focusing on low-level features. Additionally, the ITA layers further improve compression performance with an overhead that does not outweigh the processing throughput gains from batch parallelization.

A remaining limitation is not handling cases with insufficient bandwidth to downlink all processed data. In such situations, satellites must discard all remaining tiles after passing the downlink segment. Since tiles are stored as compressed latents, an informed method may quickly decide ad hoc which tiles to downlink according to how reliable they may be recovered on the ground. The intuition is to rank tiles by their uncertainty, given other tiles. We did not pursue the research due to a lack of resources for implementing and training large generative models for multispectral satellite imagery. However, due to a recent publication of a foundational model for such modalities with openly accessible weights [JYB+25], we plan to revisit the direction.

CHAPTER 6

Coding Efficient Constructions for Statistical Summaries

This chapter concerns handling data volume generated by monitoring services. While the root problem it addresses is identical to the previous two chapters, the modality requires a different methodology. The inputs are univariate data streams that must be processed within a single pass due to memory limitations and provide worst-case guarantees on arbitrary sources.

6.1 Monitoring at Scale

Statistical summaries of data distributions are a fundamental building block in modern analytics systems [CY20]. Quantile sketches are a principled approach to statistical summaries, offering mathematically sound guarantees on approximation error given strict bounds on memory consumption [GK16, WLYC13]. While the research literature on quantile sketches has extensively focused on minimizing the unserialized in-memory representation size, relatively little attention has been paid to their encoding. Sketches rarely remain local. They are transmitted repeatedly in massively distributed systems and are persisted on disk to enable fast offline analytics [TMR20].

Current approaches to sketch optimization implicitly assume that minimizing memory usage implies improved coding efficiency [CV20]. However, the assumption breaks down when multiple valid representations that are functionally equivalent for query purposes but differ significantly in their compressibility. The difficulty lies in identifying transformations that preserve the mathematical guarantees of the original algorithm. We address this optimization gap by extending the comparison-based computational model to accommodate codeword length minimization as an explicit objective. Our approach maintains strict compatibility with existing systems through receiver-side transparency,

ensuring optimized representations remain fully interoperable with legacy infrastructure. The key insight is that online construction algorithms produce representations with exploitable redundancy patterns that can be eliminated without compromising approximation guarantees *and* modifying the underlying summarization logic. This work does not progress the state-of-the-art in quantile sketches by finding new worst- or expected-case asymptotic lower bounds. Instead, it introduces a formal framework for augmenting existing sketch algorithms to improve their coding efficiencies. We first introduce measures to quantify redundancy and establish theoretical bounds on achievable efficiency gains. Then, we develop augmentation procedures that transform existing sketch instances into codeword-optimal representations while preserving their validity. Lastly, we extend the method to jointly optimize sketches when encoded in batches by exploiting inter-representation correlations post-ingestion.

We complement the theoretical analysis by empirically demonstrating the efficacy of our approach, with consistent gains for single-instance encoding. Additionally, the joint optimization further increases efficiency according to the input-stream characteristics. These gains are achieved without increasing their asymptotic runtime complexity and require no receiver-side changes. A prototype implementation and the complete evaluation code are available in a public repository[1].

The work primarily focuses on establishing the necessary theoretical framework, yet the introduced methods have direct practical implications, as they readily apply to existing systems. Hence, we occasionally break the narration to include summaries in colored boxes, making the work accessible to a broader audience, particularly readers with an engineering background.

Practitioners may only read the boxed text before proceeding to the empirical analysis to determine whether the proposed method has any value for their system. The summaries lack technical precision but convey the minimally necessary information to follow the intuition of the motivation, guarantees, and algorithms.

6.1.1 Related Work

Sketches are insert-only data structures that represent a static dataset summary and may be extended to support dynamic representations with bounded deletions [ZMW+21]. Platforms may pre-compute sketches for offline analytics or real-time monitoring in time-series databases and log stores [CS25].

Study on quantile sketches is commonly classified by the nature of their guarantees and assumptions on the input [TMR20]. Guarantees are strong when the error is bound in quantile space and require no assumptions on the input stream. The error bound may be unbiased [KLL16, GK01, ACH+13] or biased towards extreme quantiles at one tail of the empirical distribution [CKL+21, GSWY25]. No or weak guarantees are a

[1] https://github.com/rezafuru/Codeword-Optimal-Quantile-Approximation

blanket term for sketches that assume input properties and compromise on whether, what, or where the error is bound [MRL19, GDT+18, Dun21]. Randomized sketch algorithms have lower theoretical bounds, but introduce a small failure probability δ [CV20]. Deterministic algorithms can match the bound of randomized algorithms while providing guarantees in rank space, but only when assuming a known universe of bounded size [GSW24, SBAS04, ŁTV25].

Our approach shares similarities to information-theoretic work on cardinality estimation, which shows compressing mergeable sketches to their Shannon-entropy yields (near-optimal) space/error trade-off [Lan17, PW21]. However, quantile sketches must preserve the total key order. A lexicographically minimal summary has at most $\lceil 1/2\varepsilon \rceil$ keys and is trivially optimal, i.e., without additional priors, we may not further reduce the entropy of a summary with a minimal number of words. The challenge is to exploit the redundancy from *any* further source of words by finding a representation that preserves the syntax and the order at minimal entropy.

6.1.2 Background & Problem Definition

The work aims to reduce data volume by finding representations more amenable to the input of generically available compression algorithms. Accordingly, the following extends the formal computational model to accommodate the problem definition that additionally minimizes the *encoded* size of the statistical summary while maintaining compatibility with existing systems. Table 6.1 lists the introduced symbols. Function parameters are omitted when clear from context.

Table 6.1: Notations

Symbol	Description
$\mathcal{U}$	Data Universe
$\mathcal{A}$	Sketch Algorithm
$Q_{\mathcal{A}}$	Query Routine of $\mathcal{A}$
$M_{\mathcal{A}}$	Merge Routine of $\mathcal{A}$
$\mathcal{R}_{\mathcal{A}}$	Representation Space of $\mathcal{A}$
$\mathcal{V}_{\mathcal{A}}$	Valid Representations of $\mathcal{A}$
$\Leftrightarrow^*$	Semantic Equivalence
W	Wordsize function
$\mathcal{B}$	Worst-Case Bound
f	Error Bound Function
C	Encoding Function
ℓ	Codeword Length Function

Communication and Computational Model

All changes necessary to benefit from the reported gains in the evaluation are completely client-sided and only require one additional step in the serialization logic. A server may instantiate the sketch to serve queries without changes to the deserialization logic.

We refer to procedures that maintain the bound on memory requirements as *summarization* and reserve compression as a keyword for the conventional objective that reduces the entropy of a latent variable under a shared prior between the sender and receiver. The communication model assumes *separate source and channel coding* where a sender transmits values and structural information of a quantile summary. The sender and receiver only share knowledge on the syntax of the *summary structure* and established lossless compression algorithms.

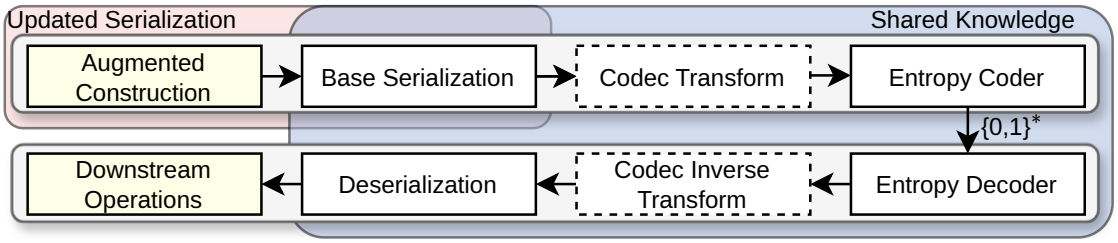


Figure 6.1: Communication model with transparent Server-side modifications.

Figure 6.1 illustrates a high-level view of the communication model. The priors shared between the sender and receiver are the structural information necessary for (De)-Serialization and the codec, i.e., the *syntax* to instanciate queriable summaries. Optionally, they may share a more powerful, but widely available, codecs that introduce additional shared priors and apply invertible (lossless) transforms, such as the Lempel–Ziv–Markov chain Algorithm (LZMA). For the theoretical analysis, we only consider entropy (de)coding, using Arithmetic Coding (AC) [WNC87], as it can compress the state of any stochastic process down to the Shannon limit with an expected 2 bits overhead. A sender-side *comparison-based* algorithm constructs a summary from an input stream with elements drawn from a totally ordered universe without further information. The algorithm is only permitted to compare and test for equality on *observed* elements, i.e., mapping elements to values not observed in a stream, such as replacing a group of elements with an average representative, is not permissible. The sender may modify the algorithm. However, the modification must provide equivalent guarantees and cannot change the representation syntax. The procedures for deserializing and querying must be identical to summaries constructed from the unmodified algorithm.

Streaming Algorithms for Quantile Approximation

Quantile sketches maintain approximate order statistics of a data stream by keeping a compressed, weighted set of representative elements. We consider the definition of

mergeability by Agarwal et al [ACH+13]. A sketch is *mergeable* if summaries of disjoint data streams can be combined to produce a result that is equivalent in accuracy guarantees to constructing a sketch over the union of the streams. A sketch is *fully mergeable* if repeated or hierarchical merging does not decrease the approximation accuracy beyond the error bounds of a sketch constructed directly over the union of the data streams. The deterministic Greenwald-Khanna (GK) sketch [GK01] keeps a sorted list of tuples that associate weight and uncertainty values with each recorded observation. On every insertion, a new tuple is instantiated and a summarization routine periodically combines adjacent tuples whose total sum of weight and uncertainty stays below $2\varepsilon n$. The GK sketch is not known to be fully mergeable.

The ACHPWY sketch is a fully mergeable randomized quantile summary that achieves $O((1/\varepsilon)\log^{3/2}(1/\varepsilon))$ space [ACH+13]. The algorithm organizes the stream into a hierarchy of compactors, where each compactor at level h holds elements with equal weight $w_h = 2^{h-1}$. A compactor stores up to k_h elements. When it becomes full, it performs a compaction step: the items are sorted, and an unbiased coin determines whether to retain the even or odd positions. The discarded items are removed, and the remaining elements have their weights doubled to $2w_h$, which effectively promotes them to the next level. Figure 6.2 illustrates this procedure.

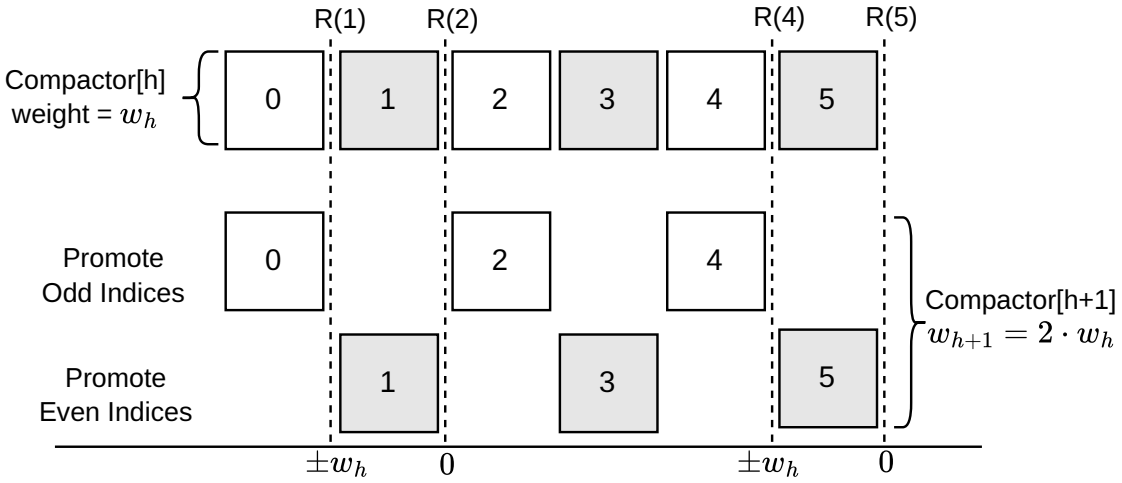


Figure 6.2: Compaction procedure.

Each compaction perturbs the rank of any query value by at most $\pm w_h$, and after m_h such operations per level, the cumulative error remains $O(\varepsilon n)$ with high probability. The merge routine concatenates compactors of two instances, level by level, and applies the compaction routine. The merged sketch has the same asymptotic guarantees as one built directly on the union of the data streams. The expected number of compactors is $O(\log(1/\varepsilon))$, giving a total space complexity of $O((1/\varepsilon)\log^{3/2}(1/\varepsilon))$ and constant-time updates. Karnin, Lang, and Liberty (KLL) improve the ACHPWY design by allowing geometrically decreasing compactor capacities, which concentrate most of the memory at

lower levels while maintaining unbiasedness [KLL16, ILL+22]. Figure 6.3 illustrates the geometric capacity scaling of KLL.

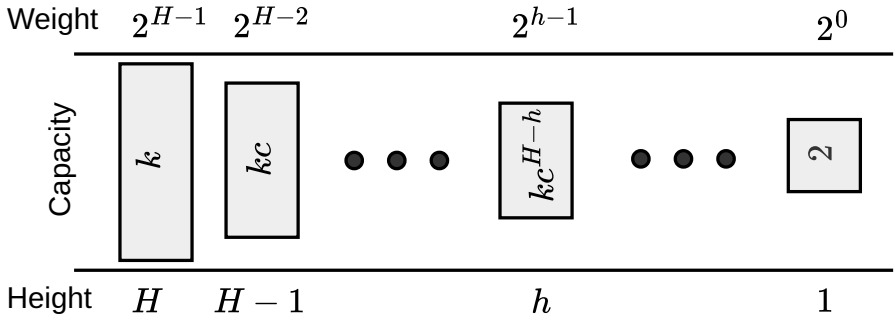


Figure 6.3: Geometric capacity scaling of KLL.

Let k_H denote the capacity of the topmost compactor and H the number of levels. The algorithm enforces $k_h \geq kc^{H-h}$, $c \in (0.5, 1)$, so the compactor capacities shrink exponentially with level. When a level exceeds its capacity, it performs the same random even-or-odd compaction and promotes the retained elements with doubled weights. This geometric scaling ensures that the lower levels, which handle lighter elements, compact more aggressively, while the upper levels maintain higher precision. The refinement removes the redundant logarithmic factor in the space bound of ACHPWY. The expected number of retained elements becomes $(O((1/\varepsilon)\sqrt{\log(1/\delta)}))$ for single-quantile estimation with failure probability (δ), and $(O((1/\varepsilon)\log\log(1/\delta)))$ for the all-quantiles case. The sketch is fully mergeable because each level's buffer corresponds exactly to that produced by running the algorithm on the concatenated stream. Conceptually, the ACHPWY sketch can be viewed as a merge-and-reduce hierarchy with uniform compactor capacities. The KLL sketch replaces this uniform design with a geometrically weighted hierarchy that smoothly transitions between randomized sampling at the bottom and deterministic Greenwald-Khanna behavior at the top. Both maintain a sorted sequence of weighted representatives whose rank errors accumulate additively and remain unbiased. Our framework builds directly on this property by introducing transformations that minimize the encoded representation length while preserving the syntax and mergeability of the original sketch.

Representations for Valid Quantile Summaries

Let $\mathcal{U}$ be a totally ordered universe and $\pi = (x_1, x_2, \ldots, x_n) \in \mathcal{U}^n$ a data stream. For any $u \in \mathcal{U}$, let $R(u; \pi) = |\{i \in 1, \ldots, n\} | x_i \leq u|$ be the rank of u in stream π. Let $\mathcal{A}$ be a *comparison-based* quantile sketch associated with a representation space $\mathcal{R}_\mathcal{A}$, query routine $Q_\mathcal{A}$, and a merge routine $\mathcal{M}_\mathcal{A}$. The representation determines the *syntax* of a sketch, i.e., the logic to create servable instances. Each representation is described by key-weight tuples $r = (x_i, w_i), x_i \in \mathcal{U}, w_i \in \mathbb{N}$. The keys are values observed from the stream. We refer to the multiset of values a representation stores as *words*, and the set

of distinct keys as *symbols*. A query routine $Q : \mathcal{R}_{\mathcal{A}} \times \mathcal{U} \to \mathbb{N}$ takes a representation and a key $Q(r, u)$ and returns an estimate of $R(u; \pi)$. For an error-bound function $f(\pi, \varepsilon, u)$, we refer to a representation $r \in \mathcal{R}_{\mathcal{A}}$ as $f(\pi, \varepsilon, u)$-valid if:

$$\exists Q_{\mathcal{A}}, \forall u \in \mathcal{U} : \Pr[|Q(r, u) - R(u; \pi)| > f(\varepsilon, \pi, u)] < \delta$$

where δ is the failure probability. We describe the set of all valid representations for a given stream and error bound as

$$\mathcal{V}_{\mathcal{A}}(\pi, \varepsilon) \coloneqq \{r : \exists Q_{\mathcal{A}}, \forall u \in \mathcal{U}, \mathsf{r} \text{ is } f(\pi, \varepsilon, u)\text{-valid}\}$$

We refer to two representations as *semantically equivalent* $r_1 \Leftrightarrow^* r_2$ when $r_1, r_2 \in \mathcal{V}_{\mathcal{A}}(\pi, \varepsilon)$. When representations are semantically equivalent, they may be used interchangeably to serve queries for a particular stream at the configured error bound. A merge routine $\mathcal{M}_{\mathcal{A}} : \mathcal{V}_{\mathcal{A}}(\pi, \varepsilon) \times \mathcal{V}_{\mathcal{A}}(\pi, \varepsilon) \to \mathcal{V}_{\mathcal{A}}(\pi_i \oplus \pi_j, \varepsilon)$ takes as input valid representations of streams π_i, π_j and outputs a valid representation for the combined stream without any input order assumption $\pi_i \oplus \pi_j$. The *construction algorithm* of a sketch takes as input a pair (π, ε) and outputs a representation $r \in \mathcal{A}(\pi, \varepsilon)$. The representation is readily queriable by $Q_{\mathcal{A}}$ and is $f-$valid, so $\mathcal{A}(\pi, \varepsilon) \subset \mathcal{V}_{\mathcal{A}}(\pi, \varepsilon) \subset \mathcal{R}_{\mathcal{A}}$.

Measuring and Encoding Representations

Let $W : \mathcal{R}_{\mathcal{A}} \to \mathbb{N}$ measure the size of a representation in words, subject to an asymptotic worst-case bound $W(\mathrm{Z}) \leq \mathcal{B}(\pi, \varepsilon)$. Let $C : \mathcal{R}_S \to \{0, 1\}^*$ be a universal, prefix-free lossless encoding that maps representations to a codeword. We measure the *codeword length* of a representation $\ell : \mathcal{R}_S \to \mathbb{N}$. The codeword of a representation r is optimal if its codeword length is minimal among all valid representations in $\mathcal{V}_{\mathcal{A}}(\pi, \varepsilon)$ with a *fixed number of words* $W(r)$, We associate with every pair (π, ε) a *lexicographically minimal (LM)* summary produced by an oracle $\mathcal{O}$. If multiple word-optimal summaries exist for an input stream, the oracle outputs are equiprobable and are therefore all codeword optimal. We refer to the difference between $\ell(\mathcal{A})$ and $\ell(\mathcal{O})$ *representational redundancy* and distinguish between *intra-* and *inter*-representation redundancy. The latter concerns instances sequentially created out of chunks of a single stream without assumptions on chunk size or distribution drift. For example, a system may provide real-time monitoring at high temporal resolution by creating one instance per minute.

6.2 Representational Redundancy

6.2.1 Intra-Representation Redundancy

Imagine you are purchasing ingredients at a market for a dish. Since keeping stock of diverse inventories is more expensive, a tax scales the base price of an item by the number of distinct items. While you may be unable to reduce the volume you must purchase, there may be a possibility to substitute some ingredients by increasing the frequency of others without ruining the dish.

For the remainder of this section, consider a running example where we create streams by incrementally drifting from a type that concentrates all probability mass on a single symbol to the type that never repeats a symbol within $t = 10$ steps. Figure 6.4 illustrates the concept for $\pi_0, \dots \pi_9$.

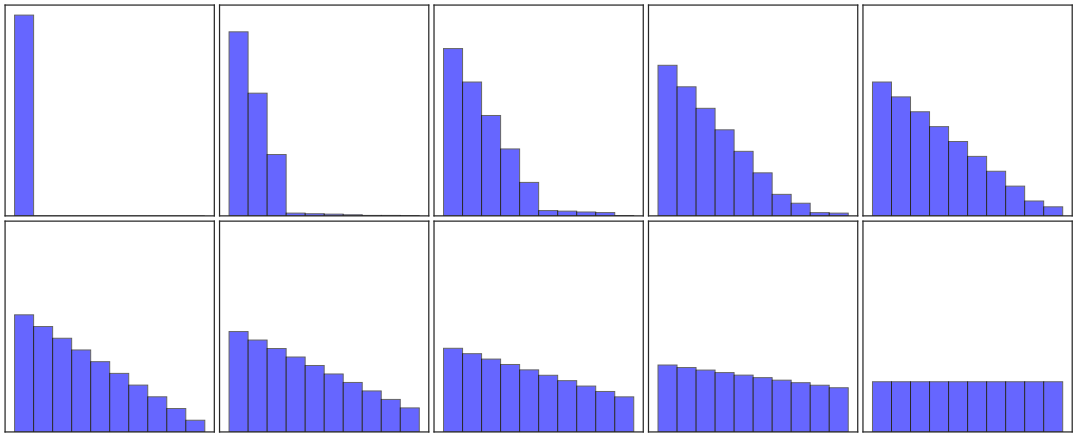
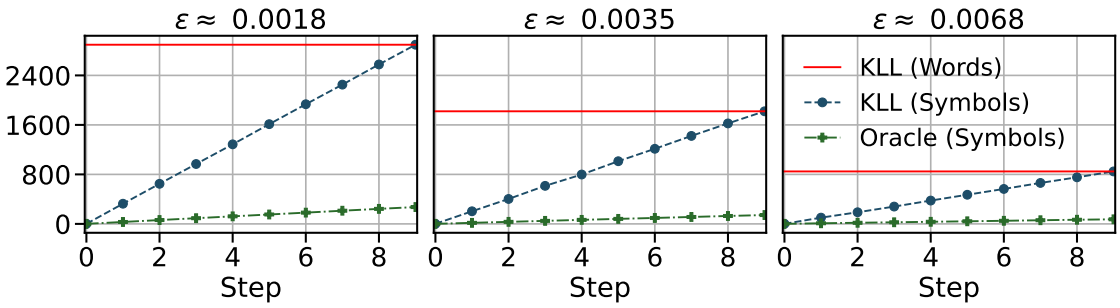


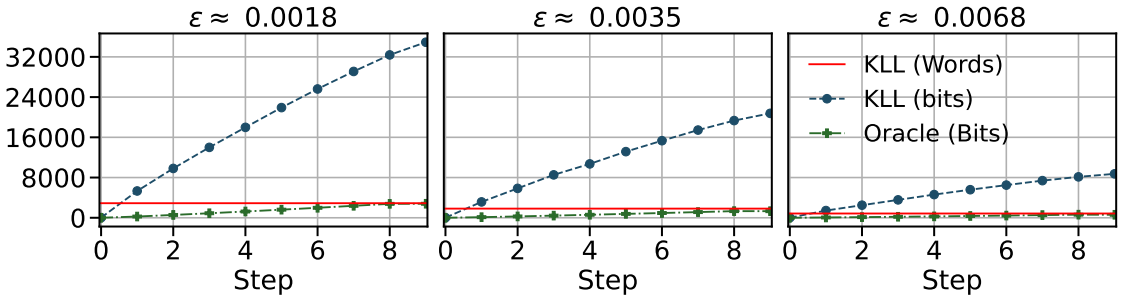
Figure 6.4: Stream types π_i with incremental drift and uniform size.

The entropy increases from 0 (single symbol) to $\log_2 |\pi|$ (all unique). Notice that for step indices, i, j with $i < j$ π_i always majorizes π_j. An online construction $\mathcal{A}(\pi_i, \varepsilon)$ may remove symbols in the summary as it ingests a stream. Still, any construction algorithm must always maintain at least as many symbols as the LM summary $\mathcal{O}(\pi_i, \varepsilon)$. The stream that emits a single symbol has zero entropy, and $\mathcal{O}(\pi_0, \varepsilon)$ consists of a single word. The codeword length for $\mathcal{A}(\pi_0, \varepsilon)$, the online algorithm is trivially optimal with uniform codeword length, as any valid summary will contain a single symbol, regardless of $W(\mathcal{A}(\pi_0, \varepsilon))$. For the stream with maximal entropy π_t (never repeating a symbol), the LM summary has *exactly* $\lceil (1/2\varepsilon) \rceil$ symbols. In contrast, the online summary must have *at least* $\lceil (1/2\varepsilon) \rceil$ symbols. Every other stream type will result in summaries where the minimal number of required symbols is between 1 and $\lceil (1/2\varepsilon) \rceil$. Hence, two sources drive the intra-representation redundancy: (I) increased codeword length from larger messages, and (II) the cost per word is directly proportional to $\frac{W(\mathcal{A})}{W(\mathcal{O})}$, regardless of whether they are apparent in an asymptotic analysis using Bachmann-Landau notation.

Figure 6.5 demonstrates the relationship between words, symbols, and codeword length, as we synthesize streams each with 2^{15} samples from maximal (lowest entropy) to minimal skew (highest entropy), comparing an LM summary with a KLL [KLL16] construction. We use arithmetic coding with a simple adaptive entropy model. Section 6.4 details the experiment setup. The construction algorithm assumes the worst-case uncertainty by design, so the number of words only scales with the target error bound. Notice that the number of symbols perfectly matches the number of words in the stream type with maximal entropy. The message length and the entropy rate (the expected cost per



(a) Measuring Symbols as the sum of all Keys in a Summary



(b) Measuring Codewords as the bitstring size output by Arithmetic Coding

Figure 6.5: Comparing KLL scaling (Top: Symbols, Bottom: Codeword length) of stream type against number of words.

word) determine the codeword length, so we can precisely quantify the representational redundancy. Notice that as a stream becomes less skewed, we are paying *twice* for every additional word, i.e., for increasing the message size, and by increasing the cost per element. Since we are sampling from a possibly unbounded universe, the number of redundant symbols scales virtually one-to-one to the factor of redundant words that an online summary has over an LM summary, unless the source distribution is highly skewed.

The following shows how to quantify the redundancy between an LM and an arbitrary summary according to the number of symbols.

Lemma 1 (Codeword Length Scaling)**.** *Let* $m := |\mathcal{O}| \in [1, \lceil 1/2\varepsilon \rceil]$ *as the number of words in the lexicographically minimal summary output by an oracle and* $k \geq 1$ *the word factor realized by an online algorithm, and define the intra-representation redundancy of a single representation as* $\ell\Delta(\cdot,\cdot,\cdot) = \ell(\mathcal{A}) - \ell(\mathcal{O})$ *Assume the* $km - m$ *additional words are all distinct. Then, the increase in expected code-word length is*

$$\Delta\ell(k, m, 1) = m[k \log k + (k-1) \log m] \tag{6.1}$$

Proof. $\mathcal{O}$ carries the uniform distribution on m symbols, so $\ell(\mathcal{O}) = m \log m$ and $\mathcal{A}$ carries the uniform distribution on km symbols, so $\ell(\mathcal{A}) = km \log km = km(\log k + \log m)$. Hence, $\ell(\mathcal{A}) - \ell(\mathcal{O}) = m[k \log k + (k-1) \log m]$. □

We may generalize the redundancy measure by accounting for the exact number of symbols in an oracle summary according to the stream type.

Corollary 1 (Parametric Redundancy Curve)**.** *Let υ denote the number of unique additional symbols and its ratio to the number of re-occurrences of existing m oracle symbols $(k-1)m-\upsilon$ as $\alpha \in [0,1] := \frac{\upsilon}{(k-1)m-\upsilon}$. Then,*

$$\begin{aligned}\Delta\ell(k,m,\alpha) = m\big[&(1+\alpha(k-1))\log(1+\alpha(k-1)) \\ &+\alpha(k-1)\log m\big]\end{aligned} \tag{6.2}$$

Proof. When the sketch keeps a factor $k \geq 1$ more words than the oracle and a fraction $\alpha \in [0,1]$ of the $(k-1)m$ extra words introduce new keys, the number of distinct keys in the sketch is $n = m(1+\alpha(k-1))$. The oracle's encoding cost is $\ell(\mathcal{O}) = m\log m$ because its m keys are equiprobable. For the sketch, whatever the true frequency vector on its n keys, entropy is maximized when that vector is uniform; hence $\ell(\mathcal{A}) \leq n\log n$, with equality exactly when every retained key appears once (Lemma 1). Substituting n yields $\ell(Z) - \ell(\mathcal{O}) = n\log n - m\log m = m[(1+\alpha(k-1))\log(1+\alpha(k-1)) + \alpha(k-1)\log m]$, establishing the claimed redundancy formula. □

The derivative of this expression with respect to α is $m(k-1)\log(1+\alpha(k-1)) + \frac{m(k-1)}{1+\alpha(k-1)}$, which is strictly positive for $k > 1$, so the redundancy increases monotonically from 0 at $\alpha = 0$ (all duplicates) to $m[k\log k + (k-1)\log m]$ at $\alpha = 1$ (all unique).

6.3 Augmented Construction Algorithms

The augmentation of an existing construction algorithm is framed as a deterministic post-transform on existing summaries to reflect their intended usage. However, we will show that such augmentations are completely online without relying on any assumption on the system, such as when and how instances are finalized, stored, or transmitted.

6.3.1 Syntax- and Order Preserving Transforms

The idea is to frame the problem as augmenting an algorithm with certain constraints. The constraints are sufficient for server-side transparency, but only mandatory to ensure that the augmentation matches the guarantees of the base algorithm. A property that immediately follows is that any algorithm implementing the constraints will only modify the state when there is a reduction in data volume.

We decouple the construction algorithm of a sketch $\mathcal{A}$ from the query and merge routines, as the receiver does not need to know how the representation is constructed. Since augmentations do not change the in-memory size of a transform, we formulate entropy-minimizing constructions as ad-hoc applicable *transforms*. For a $b \geq 1$ batch of representations $r^{(i)} \in [b]$, express the values and structural information of a representation $r_{\text{val}}, r_{\text{struct}}$ as

finite multiset of observation and weight pairs $r^{(i)} = \left\{\left(x_j^{(i)}, w_j^{(i)}\right)\right\}_{j=1}^{k_i}, x_j^{(i)} \in \mathcal{U}, w_j^{(i)} > 0$ listed in non-decreasing key order $x_1^{(i)} \leq \cdots \leq x_{k_i}^{(i)}$.

Definition 2 (Syntax and Order-Preserving Transform)**.** *A syntax and order-preserving transform*

$$\mathrm{T} : \prod_{i=1}^{b} (\mathcal{U} \times (0,\infty))^{k_i} \longrightarrow \prod_{i=1}^{b} (\mathcal{U} \times (0,\infty))^{m_i}$$

maps each instance $r^{(i)}$ *to* $r'^{(i)} = \left\{\left(y_l^{(i)}, w_l'^{(i)}\right)\right\}_{l=1}^{m_i}$ *preserving the order* $y_1^{(i)} \leq \cdots \leq y_{m_i}^{(i)}$ *subject to the existence of a non-decreasing surjection* $\sigma^{(i)} : \{1,\ldots,k_i\} \twoheadrightarrow \{1,\ldots,m_i\}$*, such that* $y_{\sigma^{(i)}(j)}^{(i)} \geq x_j^{(i)}$ *and* $w_l'^{(i)} = \sum_{j:\,\sigma^{(i)}(j)=l} w_j^{(i)}$*. The constraints state that a mapping may (i) move keys rightwards or (ii) merge any contiguous block, but it may never reorder or introduce new keys.* $\sum_l w_l'^{(i)} = \sum_j w_j^{(i)}$.

Assuming we directly pass a representation to an entropy coder, an essential property that immediately follows from the definition is that no transform that implements the constraints can ever result in a longer codeword. Hence, any augmentation will only manipulate keys when there is a strict increase in expected compression efficiency.

Lemma 2 (Entropy Minimizing Property)**.**

$$\forall r \in \mathcal{A}(\pi,\varepsilon),\ \ H(r) \geq H(\mathrm{T(r)})$$

Proof. Consider that $\sigma : \{1,\ldots,k\} \twoheadrightarrow \{1,\ldots,m\}$ is a non-decreasing surjection, so we can partition the index set into contiguous blocks $1 = a_1 \leq b_1 < a_2 \leq b_2 < \cdots < a_m \leq b_m = k$ such that $\sigma(j) = l \Leftrightarrow a_l \leq j \leq b_l$. Define the *aggregation matrix* as

$$A_\sigma = \begin{pmatrix} \mathbf{1} & \cdots & \mathbf{1} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & \mathbf{1} & \cdots & 0 \\ \vdots & & & & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & \mathbf{1} \end{pmatrix} \in \{0,1\}^{m \times k} \tag{6.3}$$

where $(A_\sigma)_{lj} = \mathbf{1}[a_l \leq j \leq b_\ell] = \mathbf{1}[\sigma(j) = l]$, i.e., the l-th row contains ones precisely in the contiguous block $a_l,\ldots,b_l$ and zeros elsewhere. Each column therefore contains exactly one $\mathbf{1}$,i.e., A_σ is *column-stochastic*. Applying it to the input weight vector $\mathbf{w} = (w_1,\ldots,w_k)^\top$ gives $\mathbf{w}^l = A_\sigma \mathbf{w}$, i.e., $w_l^l = \sum_{j=a_l}^{b_l} w_j$ Therefore, $\mathbf{w}^l$ results from summing contiguous indexes, operation sums consecutive coordinates, implying $\mathbf{w}^l \prec \mathbf{w}$ (weak majorization). Because Shannon entropy is Schur-concave, weak majorization guarantees $H(\mathbf{w}^l) \leq H(\mathbf{w})$. □

By definition, for every syntax and order-preserving transform $r \in \mathcal{A}(\cdot,\cdot)$ implies $\mathrm{T}(r) \in \mathcal{R}_{\mathsf{S}}$, so a receiver may readily deserialize and instanciate a queriable sketch using $\mathrm{T}(r)$. The following introduces a procedure for arbitrary syntax and order-preserving transforms to certify transforms where $r \in \mathcal{A}(\pi,\varepsilon)$ implies validity $\mathrm{T}(r) \in \mathcal{V}_{\mathcal{A}}(\pi,\varepsilon)$.

6.3.2 Verification Certificates

With certificates, the constraints are now sufficient to ensure that the guarantees of the base algorithm are met.

The mechanism of $f(\pi, \varepsilon, u)$-validity depends on the properties of the error bound function f. For sketch types with strong guarantees, the error bound function is either *unbiased* or *biased*.

Verification Certificate for the Additive Error Bounds

The *additive error bound* $f(\pi, \varepsilon, u) = \varepsilon n$ is unbiased toward any particular rank and treats all query positions uniformly for error calculation. The following theorem shows how a transform (Definition 2) can maintain the validity of a representation.

Theorem 1 (Additive Error Certificate)**.**

$$\begin{aligned} &\forall p \in \mathscr{C}(r, r') : |Q(r, p) - Q(r', p)| \leq \varepsilon n \\ &\Rightarrow \forall u \in \mathcal{U} : |Q(r, u) - Q(r', u)| \leq \varepsilon n \end{aligned}$$

Proof. $Q(\cdot, \cdot)$ is a monotonously increasing step function that only changes value at the points $\{z_1, z_2, \ldots, z_k\}$. Therefore, $Q(r, u)$ and $Q(r', q)$ are left-continuous step functions with discontinuities in $x_1, \ldots, x_k, y_1, \ldots, y_k$. Set $F(q) := Q(r, q) - Q(r', q)$ as the rank difference step function. The real line decomposes into open intervals $I_0 = (-\infty, y_1)$, $I_1 = (y_1, y_1]$, $I_2 = (y_1, y_2)$, $\ldots$, $I_{2k} = (y_k, \infty)$. Since there are no y inside (y_j, y_{j+1}), $Q(r', q)$ is constant there. Only x_i strictly between y_j and y_{j+1} affect $Q(r, u)$. Each such x_i increases $Q(r, u)$ by $w_i > 0$ exactly once Hence, F is a piecewise-constant, non-decreasing function on I_{2j} and attains its maximum magnitude at the right endpoint $y^-_{j+1} \in \mathscr{C}(r, r')$, such that F is constant on each even-indexed I_{2j} (i). At $q = y_j$, the function $Q(r', q)$ jumps by w_j, while $Q(r, u)$ may or may not jump. So F can change sign or shrink, but its value is directly probed at $y_j \in \mathscr{C}(r, r')$. Therefore, F is checked at every odd-indexed I_{2j-1} Now assume q lies in an even interval $I_{2j} = (y_j, y_{j+1})$. By (i) and the initial hypothesis, $|F(q)| \leq \max\left\{ |F(y_j)|, |F(y^-_{j+1})| \right\} \leq \varepsilon n$. Lastly, Let q lie in an even interval $I_{2j} = (y_j, y_{j+1})$. By (ii) and the initial hypothesis, if $q = y_i$, the bound again holds. Hence $|F(q)| \leq \varepsilon n$ for all $q \in \mathcal{U}$. □

Corollary 2 ($\mathscr{C}(r, r')$ is minimal)**.** *No strict subset of $\mathscr{C}(r, r')$ exists. Hence, no set with fewer than $2k{+}1$ points can certify the εn bound for every pair (r, r').*

Proof. Fix weights $w_1 = \cdots = w_{k-1} = \delta$, $w_k = (\varepsilon + \delta)n$ with $\delta > 0$ small. Let $x_i = i$ and relocate only x_k, setting $y_k = x_k + \eta$ for some tiny $\eta > 0$; keep $y_i = x_i$ for $i < k$. Then, $Q(r, u) - Q(r', q) = w_k > (\varepsilon + \delta)n$ while $Q(r, p) = Q(r', q)$ $\forall p \notin (y^-_k, y_k]$. So omitting either y_k or y^-_k from the probe set allows an error $> \varepsilon n$. □

Definition 3 (Unbiased Quantiles Critical Point Set)**.** *Let $f : \mathbb{R} \mapsto \mathbb{R}$ be any function and $f(y^-) := \lim_{g \to y^-} f(q)$ the one-sided (left) limit for arbitrary point $q \in \mathcal{U}$. For a representation r and a syntax and order-preserving transform T with $r' = \mathrm{T}(r)$ define the critical point set for the unbiased quantile problem as*

$$\mathscr{C}(r, r') \coloneqq \{y_i^- | 1 \leq i \leq k\} \cup \{y_i | i \leq i \leq k\} \cup \{\infty\}.$$

While this work focuses on the unbiased quantile problem, the following introduces the equivalent of $\mathscr{C}$ for the biased variant.

Verification Certificate for Relative Error Bounds

The *relative error bound*, $f(\pi, \varepsilon, u) = \varepsilon R(u; \pi)$, biases the accuracy guarantee by scaling it with the rank, resulting in tighter error near one tail of the distribution.

Definition 4 (Biased Quantiles Critical Point Set)**.** *For a representation r and a syntax and order-preserving transform T with $r' = \mathrm{T}(r)$ define the critical point set for the biased quantile problem as*

$$\mathscr{C}_{\mathrm{rel}}(r, r') \coloneqq \{x_i | i \leq i \leq k\} \cup \{y_i | i \leq i \leq k\} \cup \{\infty\}.$$

Theorem 2 (Relative Error Certificate)**.**

$$\forall p \in \mathscr{C}_{\mathrm{rel}}(r, r') : \ \big|Q(r, p) - Q(r', p)\big| \leq \varepsilon\, Q(r, p) \Rightarrow$$
$$\forall u \in \mathcal{U} : \ \big|Q(r, u) - Q(r', u)\big| \leq \varepsilon\, Q(r, u)$$

Proof. Let $P = \{x_1, \ldots, x_k, y_1, \ldots, y_k\}$ as a multiset and $p_1 < p_2 < \cdots < p_m$ ($m \leq 2k$) be the strictly increasing list of its *distinct* values. Because the rank functions $Q(r, \cdot),\ Q(r', \cdot) : \mathcal{U} \to \mathbb{N}$ are non-decreasing step functions that may change value only at points of P, the real line decomposes into the half-open intervals$I_0 = (-\infty, p_1),\ I_l = [p_l, p_{l+1})$ ($1 \leq l < m$), $I_m = [p_m, \infty)$. Inside any fixed I_l both the numerator $|Q(r, u) - Q(r', u)|$ and the denominator $Q(r, u)$ are constant. Hence, the ratio $g(u) = \frac{|Q(r,u) - Q(r',u)|}{Q(r,u)}$ is constant on I_l as well. Because the left endpoint p_l (or ∞ for I_m) belongs to $\mathscr{C}_{\mathrm{rel}}$, verifying the bound $g(p_l) \leq \varepsilon$ implies $g(u) \leq \varepsilon$ for every $u \in I_l$. The union of the I_l covers $\mathcal{U}$, such that the stated inequality holds universally. □

Corollary 3 ($\mathscr{C}_{\mathrm{rel}}$ is minimal)**.** *For general representations with identical weights, no proper subset of $\mathscr{C}_{\mathrm{rel}}(r, r')$ is sufficient to ensure the above inequality.*

Proof. Remove any boundary $b \in P$ from the set and suppose, w.l.o.g that $b = x_i$ for some i (the case $b = y_j$ is symmetric). Construct a new transform $\tilde{r}'$ by shifting the value x_i to $x_i + \Delta$ with $0 < \Delta \ll 1$ while keeping its weight unchanged. The modified sketch $\tilde{r}'$ agrees with r' at every remaining test point,. Yet, on the whole interval $[b, b + \Delta)$ we now have $Q(r, u) = Q(r, b)$ and $Q(\tilde{r}', u) = Q(r', b)$, so $\frac{|Q(r,u) - Q(\tilde{r}',u)|}{Q(r,u)} = \frac{w_i}{Q(r,b)} > \varepsilon$ for sufficiently small Δ that lies entirely inside I_l. □

6.3.3 Prefix-Additive Error

A certificate can only verify a representation after modifications. Here, we show that we may design algorithms that maintain the invariants set by the validity constraint with capped prefix sum that track accumulating error.

We introduce a prefix-additive error εn that can certify transforms without explicitly verifying against $\mathscr{C}(r, r')$ repeatedly. For a T, let $\Delta(x) = Q(r, x) - Q(\mathrm{T}(r), x)$ with $\|\Delta\|_\infty := \sup_{x \in \mathcal{U}} |\Delta(x)|$ be the *prefix-error function*. The following shows that tracking the prefix error reduces to verifying T against $\mathscr{C}(r, r')$ explicitly.

Theorem 3 (Reduction to $\mathscr{C}$ Verification)**.** *Assume* T *maintains an online prefix-error counter* $\mathrm{err}_{pref} = \max_{i \le m} |\sum_{j \le i} w_j|$ *over a totally ordered representation* $r = (v_j, w_j)_{j=1}^k$ *and enforces* $\mathrm{err}_{pref} \le \varepsilon$*. Then a constant number of passes ensures* $\|\Delta\|_\infty \le \varepsilon n$ *is equivalent to verification against* $\mathscr{C}(r, \mathrm{T}(r))$*.*

Proof. For any key x, the rank of the original representation is the total weight of items with value $v \le x$. While scanning the representation, the prefix carries the weight $Q(r, v_i^-) = \sum_{j: v_j < v_i} w_j$. By definition $\mathrm{T}(r)$ has the same weighted computation, except with modified values v'_j, such that the same scan at position the rank is $Q(\mathrm{T}(r), v_i^-) = \sum_{j: v'_j < v_i} w_j$. The difference of the two sums is the *net weight* shifted left of v_i, i.e., precisely the running counter $\mathrm{err}_{\mathrm{pref}}(i) = Q(r, v_i^-) - Q(\mathrm{T}(r), v_i^-) = \Delta(v_i^-)$. By definition, transforms are only accepted when $\|\Delta\|_\infty \le \varepsilon n$. Therefore, $\mathrm{err}_{\mathrm{pref}} = \max_i |\Delta(v_i -)| \ge \|\Delta\|_\infty^{\mathscr{C}}$, and $|\Delta(p)| \le \varepsilon n$ at every critical point $p \in \mathscr{C}(r, (\mathrm{T}(r))$ follows immediately. Therefore, invoking Theorem 1 extends to all $u \in \mathcal{U}$. Conversely, suppose $\mathscr{C}(r, \mathrm{T}(r))$, but $|\Delta(v_j^-)| > \varepsilon n$. However, because by definition $v_j^- \in \mathscr{C}(r, \mathrm{T}(r))$, such a prefix contradicts the certificate. □

Definition 5 (Augmented Constructions)**.** *Denote a* $\mathscr{C}$*-certified syntax- and order preserving transform* T *as* $\mathscr{T}$*. Assume* $\mathcal{A}$ *is an online construction of valid summaries. Then, we refer to* $\mathscr{T}$ *as an augmentation, and an augmented construction as the composition* $\mathcal{A}_{\mathscr{T}} := \mathscr{T} \circ \mathcal{A}$*.*

The remainder of the work assumes the construction algorithm as $\mathcal{A} = \mathrm{KLL}$ to show finer-grained properties of an augmented construction $\mathrm{KLL}_{\mathscr{T}}$. Starting with how from a $\mathscr{C}$-certificate, the global rank error is also bound by εn with a negligible constant increase in failure probability.

Lemma 3 (Global Rank Error Failure Probability)**.** *Fix a query value* u*, and representations* r*,* $r' = \mathscr{T}(r)$*. Denote the respective global rank errors as* $E(u) = Q(r, u) - R(u; \pi)$ *and* $E'(u) = R(u; \pi) - Q(r', u)$*. Let* $F(u) = Q(r, u) - Q(r', u)$ *be the local deterministic offset where by the properties of* $\mathscr{T}$ *and Theorem 3* $|F(u)| \le \varepsilon n$*. Define the prefix error ratio* $\rho_\Delta := \frac{\|F\|_\infty}{\varepsilon n} \in [0, 1]$*. Since* $\Pr[|E(u)] > \varepsilon n \le \delta$ *has symmetric parts*

$\Pr[E(u) < -\varepsilon n] = \delta/2$ *for an augmented KLL construction, the failure probability is*

$$\Pr[|E'(u)| > \varepsilon n] \leq \frac{\delta}{2} + \left(\frac{\delta}{2}\right)^{(1-\rho_\Delta)^2} \tag{6.4}$$

Proof. For the right tail over-estimates, $E'(u) > \varepsilon n \;\Rightarrow\; E(u) = E'(u) + F(u) > \varepsilon n$. Hence, $\Pr[E'(u) > \varepsilon n] \leq \Pr[E(u) > \varepsilon n] = \delta/2$. For the left tail under-estimates $E'(u) < -\varepsilon n$ iff $E(u) < -(\varepsilon n - F(u)) =: -\theta$ Since $\rho_\Delta \leq 1$ we have $\theta = (1 - \rho_\Delta)\varepsilon n$. The Hoeffding bound for a centered sub-Gaussian gives $\Pr[E(u) < -\theta] \leq \exp(-\theta^2/2\sigma^2)$. From $\sigma^2 = (\varepsilon n)^2/(2\log(2/\delta)$ and $\theta = (1-\rho)\varepsilon n$ we have $\Pr[E(u) < -\theta] \leq \exp\left(-(1-\rho_\Delta)^2 \log(2/\delta)\right) = \left(\frac{\delta}{2}\right)^{(1-\rho_\Delta)^2}$. The union bound yields (6.4). □

The following shows the possibility of further ingesting elements from a stream after applying $\mathscr{T}$.

Corollary 4 (Post-Transform Processing)**.** *Consider processing a stream in intervals $\pi = \pi_{t_0} \cup \pi_{t_1} \cup \ldots \pi_{t_s}$, where $n_{t_i} = |\pi_{t_0} \cup \cdots \cup \pi_{t_i}|$. Let r_{t_0} be the representation after the first interval. For $t > t_0$, feed the sketch instance with an unmodified representation of the remaining items. Then, for every query key x, $\Pr\Big[|Q(r'_t, x) - R(\pi_t, x)| > \varepsilon n_t\Big] \;\leq\; 2e^{-K\varepsilon^2 k}$, where k is a constant propotional to (ε, δ) and $K > 0$ the usual Hoeffding constant.*

Proof. For fixed values x, the random error $Q(r_t, x) - R(\pi_t; x)$ of an instance r at time t is $\sum_{h=1}^{H(t)} \sum_{i=1}^{m_h(t)} w_h X_{h,i}$, where every $X_{h,i} \in \{-1, 0, +1\}$, $\mathbb{E}[X_{h,i}] = 0$, and $\sum_{h,i} w_h^2 \leq K, n(t)/k$ for the usual Hoefding constant K. Invoking Hoeffding immediately gives $\Pr[\; |Q(r_t, x) - R(\pi_t; x)| \;>\; \varepsilon n_t] \;\leq\; 2e^{-C'\varepsilon^2 k}$. A *transformed* instance $|Q(r, u) - Q(\mathscr{T}(r), u)| \leq \varepsilon n$ for $u \in \mathcal{U}$, introduces a deterministic (not necessarily positive) error offset $e_{t=0}(x)$. For a $t > 0$ split the error $Q(\mathscr{T}(r_t), x) - Q(r_t, x)$ into $Q(\mathscr{T}(r_t), x) - Q(\mathscr{T}(r_0), x) + \Delta_{t=0}(x)$. Since the $\mathscr{C}$-certificate guarantees that $\Pr[|Q(r, x) + R(\pi, x) + \Delta(x)| \leq \varepsilon n] \geq 1 - \delta$ for arbitrary π , the total error is $|Q(\mathscr{T}(r_t), x) - R(x; \pi_t)| \leq \varepsilon n_0 + \varepsilon\,(n_t - n_0) = \varepsilon n_t$ with probability $\geq 1 - \delta$. □

Corollary 5 (Chained Transforms)**.** *Suppose we track the prefix error incurred by a $\mathscr{T}$ that is executed at the end of intervals $t_1 < t_2 < \cdots < t_s$ producing offsets $\Delta_t(x)$ with $|\Delta_{t_j}(x)| \leq \varepsilon(n_{t_j} - n_{t_{j-1}})$ with $(j \geq 1)$. Then $|Q(\mathscr{T}(r_t), x) - R(\pi_t, x)| \;\leq\; \varepsilon n_t$ for every query x with probability $\geq 1 - \delta$.*

Proof. The offsets are deterministic and add linearly, so $\left|\sum_{j=1}^{s} \Delta_{t_j}(x)\right| \;\leq\; \varepsilon \sum_{j=1}^{s} (n_{t_j} - n_{t_{j-1}}) = \varepsilon n_{t_s}$. Applying Hoeffding deviation $\varepsilon(n_t - n_{t_s})$ from post-t_s updates and the triangle inequality concludes the proof. □

Together Corollary 4 and Corollary 5 imply that an augmentation is itself an online construction algorithm.

6.3.4 Prefix Bound Operations

Imagine the rank function as a staircase, where a new stair is added in intervals whenever a value changes. A summary containing the minimal number of points to ensure at most εn error is a smooth staircase at every segment where a non-minimal summary would introduce bumps. A prefix sum lets you safely iron out some of the bumps, reducing the number of distinct keys by increasing the frequency of existing keys. Since segments are disjoint intervals, we can run separate prefix sums.

Algorithm 1 shows the basic *PrefixBoundCollapse* (PBC) procedure based on the prefix error function that a transform may use to find an equivalent representation.

Algorithm 1 PrefixBoundCollapse(C, ε, n)

$F \leftarrow \text{FLATTEN}(C)$ to list of (v, w, l, i)
$B \leftarrow \lceil \varepsilon n \rceil, \quad \Delta \leftarrow 0, \quad L \leftarrow F[0].v$
for $j = 0$ to $|F| - 2$ **do**
    $(v_1, w_1, l_1, i_1) \leftarrow F[j], \quad (v_2, w_2, l_2, i_2) \leftarrow F[j+1]$
    if $|\Delta + w_2| \leq B$ **then**
        $C[l_2][i_2] \leftarrow v_1, \quad F[j+1].v \leftarrow v_1, \quad \Delta \leftarrow \Delta + w_2$
return C

The flatten operation maps the compactor elements to a flat list of named tuples to explicitly access the compactor level l and index i. The routine assumes that it is only ever called after a compaction operation, so $v_i \leq v_{i+1}$. From Lemma 2 it follows that PBC may only reduce entropy, and serves as the primitive operation for transforms that run prefix errors in *disjoint segments*.

Lemma 4 (Locally Bound Error)**.** *Let $I = (v_s, \ldots, v_e)$ be a contiguous segment of a lexicographically minimal summary, and $w(I) = \sum_{i=s}^{e} w_i \leq \varepsilon n$. For any query x if $x < v_s$ or $x \geq v_e$ then $\Delta_I(x) = 0$ and if $v_s \leq x < v_e$ then $|I(x)| \leq w(I) \leq \varepsilon n$.*

Proof. Inside the segment the transform moves every value to an existing symbol e, so the only items that can change their contribution to the rank at x are those within the original value in $[v_s, x]$ if $x \geq e$ or $[x, e]$ if $x < e$. In both cases, the total weight affected is bound by $w(I)$, i.e., outside the interval, all values either remain $\leq x$ or $> x$. □

We can now show that any $\mathscr{T}$ manipulating keys using the PBC routine within bound intervals retains mergeability properties using the augmented procedure $\mathcal{M}_{\mathcal{A}_{\mathscr{T}}}$.

Lemma 5 (Post-Transform Merges)**.** *Assume $\mathcal{A}$ to be the fully mergeable KLL variant with merge procedure $\mathcal{M}_{KLL}$. Let $\mathscr{T}$ denote a $\mathscr{C}$-certifiable transform manipulating keys using the PBC routines. For the $\mathcal{A}_{\mathscr{T}}$ the augmented construction algorithm there exists a merge procedure $\mathcal{M}_{\mathcal{A}_{\mathscr{T}}} : \mathcal{V}_{\mathsf{S}} \times \mathcal{V}_{\mathsf{S}} \mapsto \mathcal{V}_{\mathsf{S}}$ with polynomial runtime, such that for every triple of valid representations $r_1 = \mathscr{T}(z_1)$, $r_2 = \mathscr{T}(z_2)$, $r_3 = \mathscr{T}(z_3)$ with*

$z_i \sim \mathcal{A}(\pi_i, \varepsilon)$ the following properties hold: (I) $\mathcal{M}_{\mathcal{A}_\mathscr{T}}(r_1, z_2) \Leftrightarrow^* T(\mathcal{M}(z_1, z_2))$ (One-Sided Mergeability). (II) $\mathcal{M}_{\mathcal{A}_\mathscr{T}}(r_1, r_2) = \mathcal{M}_{\mathcal{A}_\mathscr{T}}(r_2, r_1)$ (Commutativity). (III) For any r_1, r_2, r_3, $\mathcal{M}_{\mathcal{A}_\mathscr{T}}(r_1, \mathcal{M}_{\mathcal{A}_\mathscr{T}}(r_2, r_3)) \Leftrightarrow^* \mathcal{M}_{\mathcal{A}_\mathscr{T}}(\mathcal{M}_{\mathcal{A}_\mathscr{T}}(r_1, r_2), r_3)$ (Associativity). Hence, the augmented algorithm is fully mergeable.

Proof. Define the augmented merge procedure

$$\mathcal{M}_{\mathcal{A}_\mathscr{T}}(r_1, r_2) := \mathscr{T}\left(\mathcal{M}_{\mathrm{KLL}}(\mathrm{KLL}(r_1), \mathrm{KLL}(r_2))\right).$$

Describe instanciating KLL with the multiset as $z_i^{\uparrow} := \mathrm{KLL}(r_i)$. Consider that for every key-weight pair (v, w) of r_i, $\mathscr{T}$ has only moved keys to the right and summed weights of contiguous blocks. No information required by KLL has been discarded. Let $z_* = \mathcal{M}(z_1^{\uparrow}, z_2^{\uparrow})$ and run the same $\mathscr{T}$ again $r_* := \mathscr{T}(z_*)$. All three operations are deterministic and require only constant left-to-right scans of r_1 and r_2, needing $O(|r_1| + |r_2|) = O(k)$ runtime. $\mathcal{A}_{\mathrm{KLL}}$ is fully mergeable, so $\mathcal{M}_{\mathrm{KLL}}(z_1, z_2) \Leftrightarrow^* \mathcal{A}_{\mathrm{KLL}}(\pi_1 \oplus \pi_2)$. From Corollary 4 it follows that $r_* = \mathscr{T}(z_*)$ is a valid summary for π. Consider that instanciating KLL ${z_1}^{\uparrow}$ now uses $r_i = \mathscr{T}(z_1)$ instead of z_1. Commutativity directly follows from both $\mathcal{M}_{\mathrm{KLL}}$ and $\mathscr{T}$ being symmetric in their arguments. From the associativity of the underlying KLL merge follows:

$$\begin{aligned} M_\mathscr{T}(r_1, M_\mathscr{T}(r_2, r_3)) &= \mathscr{T}\left(M_{\mathrm{KLL}}\left(z_1^{\uparrow}, M_{\mathrm{KLL}}(z_2^{\uparrow}, z_3^{\uparrow})\right)\right) \\ &= \mathscr{T}\left(M_{\mathrm{KLL}}\left(M_{\mathrm{KLL}}(z_1^{\uparrow}, z_2^{\uparrow}), z_3^{\uparrow}\right)\right) \\ &= M_\mathscr{T}(M_\mathscr{T}(r_1, r_2), r_3). \end{aligned}$$

Therefore, arbitrary merge trees yield equivalent results. □

6.3.5 Optimizing Instances Separately

The problem of maximizing allocatable prefix errors reduces to the continuous bin packing problem. By exploiting the properties of the cost function (Entropy), we can find an optimal solution in linear time and space.

We refer to a block as a contiguous segment of values that an algorithm collapses to a single key. Hence, the optimization problem is *finding the least number of blocks.*

Codeword Optimal Representations

Let $B = \varepsilon n$ and $E(I, j) = \min_P H(P)$ where P partitions $(w_1, \ldots, w_j)$ in to blocks with boundary $E(0, 0) = 0$ and $E(j, 0) = \infty$ for $j > 0$. Given the sorted weight vector $\mathbf{w} = (w_1, \ldots, w_k)$ we aim to find a partition into exactly W contiguous blocks, each of weight $\leq B$ that minimizes $H(P) = \sum_{I \in P} \frac{||I||}{n} \log \frac{n}{||I||}$. Define the expected codeword length contribution (block cost) as $h(x) := \frac{x}{n} \log \frac{n}{x}$. Since the cost function h is strictly concave, the edge-cost matrix satisfies the quadrangle inequality (Monge

property [BKR96]). Consider that the recurrence for any contiguous segmentation with concave per-block cost

$$\mathrm{dp}[j] = \min_{\substack{i<j,\\ P[j]-P[i]\le B}} (\mathrm{dp}[i] + h(P[j] - P[i])) \tag{6.5}$$

is an instance of the concave least-weight subsequence problem [Wil88]. Since the edge-cost matrix $C[i, j] = h(P[j] - P[i])$ is a Monge array the quadrangle inequality follows. Let $E_\lambda(j) = \min_{P\in\mathcal{P}_j(B)} (\sum_{I\in P} h(\|I\|))$Compute $E_\lambda(j)$ for $j = 1, \ldots, k$ by rolling over candidate predecessors

$$E_\lambda(j) = \min_{\substack{i\le j\\ S_{ij}\le B}} \{E_\lambda(i-1) + h(S_{ij})\}. \tag{6.6}$$

Algorithm 2 implements the recurrence that finds the least number of blocks as a single linear-time scan that dynamically aggregates contiguous ranges whose cumulative weight remains within the admissible bound.

Algorithm 2 SMA-Pass(P, Γ)

$Q \leftarrow [0]$, $\mathsf{cost}[0] \leftarrow 0$, $\mathsf{blk}[0] \leftarrow 0$, $\mathsf{par}[0] \leftarrow 0$
for $j = 1$ to k **do**
  while not $\Gamma(Q[0], j)$ **do**
    pop $Q[0]$
  $i^\star \leftarrow Q[0]$, $w \leftarrow P[j] - P[i^\star]$
  $\mathsf{cost}[j] \leftarrow \mathsf{cost}[i^\star] + w \log \frac{P[k]}{w}$
  $\mathsf{blk}[j] \leftarrow \mathsf{blk}[i^\star] + 1$, $\mathsf{par}[j] \leftarrow i^\star$
  while $|Q| \ge 2$ and $\mathrm{slope}_{i_1,i_2,j} \ge 0$ **do**
    pop $Q[-1]$
  append j to Q
return par

The key idea is that each element represents a prefix sum of weights, and we maintain a queue of candidate predecessors corresponding to feasible partition boundaries. For every new prefix, we remove candidates that would violate the weight constraint or cannot yield a lower cost. Because the block-cost function (entropy) $h(x) = \frac{x}{n} \log \frac{n}{x}$ is concave, the predecessor that minimizes the cost always advances monotonically with the scan index. This monotonicity allows the algorithm to update the optimal partition state using a two-ended queue, so each element is inserted and removed at most once. Intuitively, the procedure merges adjacent prefixes as long as doing so does not exceed the permissible cumulative error εn. Whenever the bound would be violated, a new block is started. The resulting sequence of parent indices identifies the least number of contiguous blocks that jointly minimize the expected codeword length under the admissibility constraint.

Intuitively, the algorithm behaves like a "running aggregator" that greedily fuses nearby weights while ensuring that the total rank error of every collapsed segment remains within the prescribed limit.

After the SMA pass, we must simply backtrack to recover the block boundaries and then collapse the values to their corresponding keys. Algorithm 3 describes the complete procedure that runs in $O(k)$ and requires $O(k)$ memory.

Algorithm 3 S-CLO(C, ε)

$n \leftarrow \sum_{l,i} 2^l |C[l]|$
$F, P \leftarrow$FlattenAndPrefixSum(C)
$\Gamma(i, j) : \; P[j] - P[i] \leq \lceil \varepsilon n \rceil$
par $\leftarrow$SMA-Pass(P, Γ)
$C \leftarrow$ back-track $\mathcal{S} = \{(f, e)\}$ from par
for $(f, e) \in \mathcal{S}$ **do** ▷ collapse blocks
    $r \leftarrow \lfloor (f + e)/2 \rfloor$, $v^\star \leftarrow F[r].v$
    for $p = f$ to e **do**
        $(\_, \_, l, i) \leftarrow F[p]$; $C[l][i] \leftarrow v^\star$
return C

Theorem 4 (S-CLO is Codeword Optimal)**.** *Let $w_1, \ldots, w_k \in \mathbb{N}$ denote the flattened weights, $n = \sum_{i=1}^{k} w_i$, $B = \lceil \varepsilon n \rceil$, and $h(x) = x \log_2 \frac{n}{x}$ the Shannon block-cost. For any contiguous partition $P \in \mathcal{P}_B$, define the code-word cost as $\ell(P) = \sum_{I \in P} h(\sum_{i \in I} w_i)$ with $P^* = \arg\min_{P \in \mathcal{P}_B}(\ell(P))$. Algorithm 3 outputs a representation that realizes P^*. Hence, it minimizes the expected codeword length among valid syntax- and order-preserving representations that can be obtained from the original representation.*

Proof. Inside a block, all keys are collapsed to a single representative. For an optimal prefix code, the expected length contributed by that block equals $h(\sum_{i \in I} w_i) = \sum_{i \in I} w_i \log_2 \sum \frac{n}{\sum_{i \in I} w_i}$. The recurrence Equation (6.5) with prefix sums $P[j] = \sum_{t \leq j} w_t$ computes the global optimum since (I) removing the last block of an optimal partition leaves an optimal partition of the prefix and (II) $h(x)$ is strictly concave so the edge-cost matrix $C[i, j] = h(P[j] - P[i])$ is totally monotone. Algorithm 2 maintains a dequeue of candidate predecessors and updates dp[j] in $O(1)$, exploiting the Monge property. Algorithm 3 invokes the Algorithm 2 with the predicate $\Gamma(i, j) : P[j] - P[i] \leq B$ to obtain the parent array that realizes the minima of the recurrence. After, it back-tracks the parents to recover P^* to collapse each block using the corresponding keys. No other admissible transform can yield a shorter codeword. □

6.3.6 Joint Optimization

The idea is to make the representations more compressible by remapping the observed keys post ingestion. The summaries remain valid for their corresponding chunk.

Encoding with concatenation as a batch strictly generalizes the single instance case, i.e., we solve a *multivariate segmentation problem*. The optimization objective is to minimize the codeword length according to the intra-stream novelty factor α_i *and* the inter-stream overlap ratio β. Applying S-CLO or Greedy-PBI to representations separately optimizes only according to α_i. Algorithm 4 describes the complete algorithm.

Algorithm 4 J-CLO($\boldsymbol{C}, \varepsilon$)

$\boldsymbol{n} \leftarrow \left(\sum_{l,i} 2^l |C_s[l]|\right)_{s=1}^{|\boldsymbol{C}|}$
$\mathcal{E} \leftarrow \lceil \varepsilon \cdot \boldsymbol{n} \rceil$
$\Gamma(i,j) : \forall s\ \boldsymbol{P}[s][j] - \boldsymbol{P}[s][i] \leq \mathcal{E}[s]$
$\boldsymbol{F}, \boldsymbol{P} \leftarrow$FlattenAndPrefixSum($\boldsymbol{C}$)
par $\leftarrow$SMA-Segment(P, Γ)
backtrack boundaries $\mathcal{S} = \{(f,e)\}$ from par
for $(f,e) \in \mathcal{S}$ **do**
    $r \leftarrow \lfloor (f+e)/2 \rfloor$, $v^\star \leftarrow F[r].v$
    for $p = f$ to e **do**
        $(\_, \_, l, i, s) \leftarrow F[p]$; $\boldsymbol{C}[s].C[l][i] \leftarrow v^\star$
return $\boldsymbol{C}$

For each candidate segment $[s,e]$ define its weight vector $\mathbf{w}_{s,e} = (w_{s,e}^{(1)}, \ldots, w_{s,e}^{(b)})$ with $w_{s,e}^{(i)} = \sum_{j=s}^{e} w_j^{(i)}$. A segment is admissible iff $w_{s,e}^{(i)} \leq \varepsilon n_i$ for every i. Analogous to the single instance case, under the worst-case uniform distribution in symbol frequency the cost of as segment is $h(\mathbf{w}_{s,e}) = \frac{||\mathbf{w}_{s,e}||}{N} \log \frac{N}{||\mathbf{w}_{s,e}||}$, i.e., a concave function with l_1 weight. Our goal is to partition the flattened list with minimal total cost. Since h is concave and the admissibility test is additive *and* separable by the representations, the Monge property still holds. The only difference is that the queue Equation (6.6) in now stores *vectors of cumulative weights*, so there are still $O(1)$ amortized pops per j. Hence, we may generalize the single stream Algorithm 3 to the concatenated joint case by first collecting the keys and passing the predicate $\Gamma(i,j) : \forall s, P[s][j] - P[s][j] \leq \mathcal{E}$ where $\mathcal{E}$ is the array of error bounds $n_s\varepsilon$ with runtime $O(bk)$ and space $O(bk)$.

6.4 Empirical Analysis

6.4.1 Methodology

For all experiments, we repeat each trial 100 times unless stated otherwise, reporting aggregated means with standard deviations displayed as error bars. We use double-precision floating-point numbers to emulate an unbounded universe model. This is particularly relevant when evaluating whether J-CLO can leverage input stream similarity.

Reproducibility & Evaluation Framework

All reported results are seamlessly reproducible with minimal effort, as the accompanying repository organizes configuration files by experiment type[2]. The evaluation framework minimizes the number of dependencies to reduce interference on performance from external factors. One challenge in reporting intuitive results for real-world performance is selecting and configuring the compression algorithm (e.g., delta coding, precision, etc.). However, we find few *relative* performance differences between KLL and S-/J-CLO, i.e., absolute differences in results vary at comparable scales. Therefore, we use the LZMA implementation of the Python 3.10 standard library with default configurations. Additionally, we implement serialization that is not version or programming language-dependent, using a special symbol for compactor tags. This minimizes reporting absolute performance differences that may differ considerably due to implementation details.

Datasets

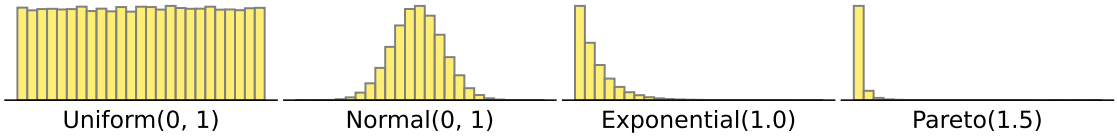


Figure 6.6: Common Sources drawn from to Synthesize Streams.

Experiments are categorized into two configuration archetypes according to dataset properties: (I) Streams sampled from the common Normal, Uniform, Pareto, and Exponential sources to evaluate single instance encoding (Figure 6.6). (II) Datasets that demonstrate specific edge cases and behaviors predicted in the theoretical analysis.

We favor synthesized datasets as we can engineer them to cover a wide range of monitoring patterns and demonstrate the whole range of performance expectations according to types of empirical distributions. Interested practitioners can find examples in the accompanying repository demonstrating how to integrate their datasets to determine whether the reported improvements will hold for their applications.

KLL and Augmentations Implementation

All augmentations are applied to the construction algorithm of the fully mergeable KLL variant based on the prototype[3] while considering the changes and configurations of the Apache Datasketches implementation[4]. The asymptotic number of words (i.e., retained observations) does not scale with stream size, and the precise size varies between $\approx$2-3k for tighter error bounds. Accordingly, we consider the spread in the number of words by sampling streams with sizes in a geometric sequence from 2^{14} to 2^{17} in 25 steps. Except

[2] https://github.com/rezafuru/Codeword-Optimal-Quantile-Approximation/config

[3] https://github.com/edoliberty/streaming-quantiles

[4] https://datasketches.apache.org/

Table 6.2: KLL Words by Base Capacity

Approx. Error	Capacity	Mean Words	SD	Max Words	Min Words
0.0018	1600	3111	776	4277	1251
0.0035	800	1640	393	2316	662
0.0068	400	841	199	1193	352
0.0133	200	430	104	608	192
0.0512	50	120	27	166	46
0.1004	25	69	17	99	27

for the experiment that focuses on scaling by stream size, we increase from 2^{14} to 2^{22} in 175 steps. Table 6.2 summarizes the number of words retained in KLL by target bound and base capacity using the larger sequence.

6.4.2 Performance on Separately Transformed Instances

Intra-Representation Symbol Scaling

The results in Figure 6.7 are from the same experiment as in Section 6.2.1, except that results are measured on the entire spread of retained elements of a target bound. Streams are synthesized by incrementally drifting from a maximally skewed (single symbol) to a perfectly uniform (all unique) type as illustrated in Figure 6.4.

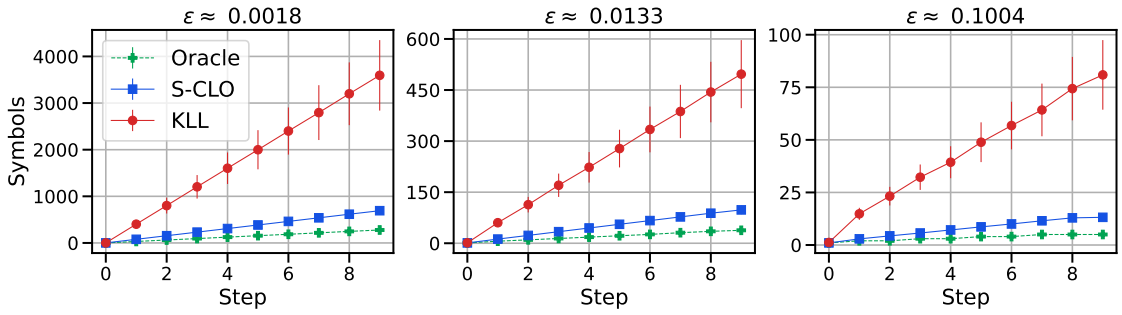


Figure 6.7: Contrasting redundant symbols scaling between S-CLO and KLL.

The mean symbols scale uniformly across error bounds relative to the oracle. Symbol SD in KLL increases with input stream uniformity, and at step $t = 9$, the Symbol and Words SD are identical. Symbol SD in S-CLO is minimal, even at the tightest error bound. Comparing how the mean and variance in symbols scale of KLL as we decrease the skew directly reflects the spread of retained words shown in Table 6.2. Not only does S-CLO significantly close the gap in symbols, but it is also nearly agnostic towards the number of words. We emphasize that S-CLO does not promise any new worst-case lower bounds. The type of input streams at the later steps may have high entropy at the last steps, but they are straightforward for S-CLO to optimize for.

Single-Instance Encoding on Common Sources

Figure 6.8 summarizes the results practitioners may expect if their distribution follows common sources or any interpolated mixture on the full range of error bounds and stream sizes.

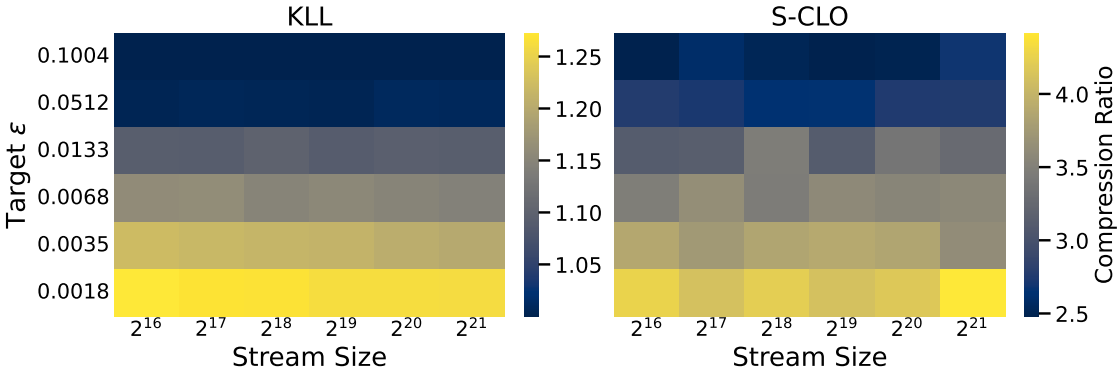


Figure 6.8: Mean compression ratio on common sources using the large-scale experiment configuration.

Note that the most dominant parameter is the error bound. The improvement over the sketch algorithm S-CLO augments is primarily determined by the factors that scale the number of words over a lexicographically minimal summary. Since the number of words of KLL does not scale by stream size, it only negligibly affects the performance of S-CLO. Figure 6.9 aggregates the stream sizes and datasets to show the spread in compression ratio.

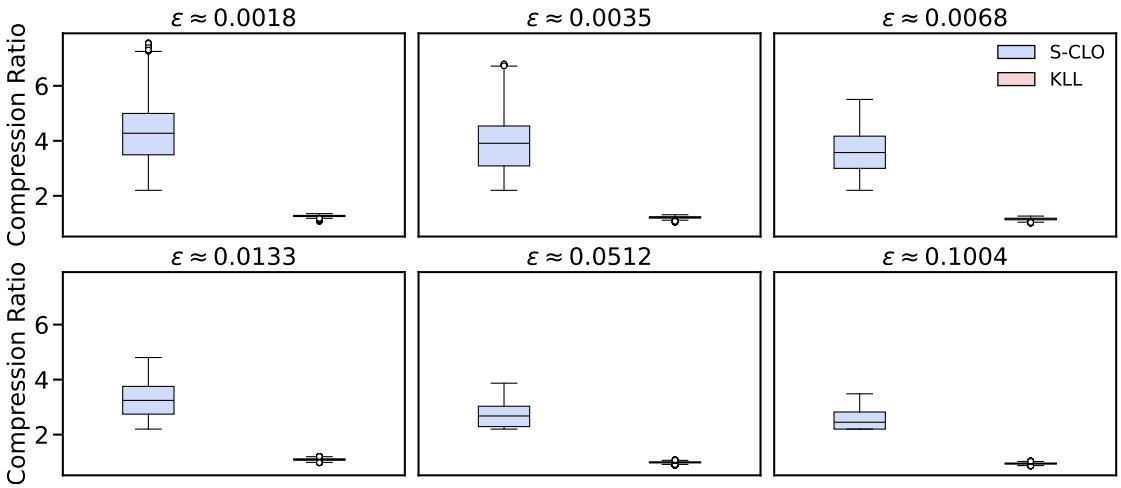


Figure 6.9: Contrasting relative performance between S-CLO and KLL.

Unsurprisingly, the KLL representations are not compressible (compression ratio ≈ 1), as the symbol distribution will be (near) uniform even when sampling from skewed

distributions. S-CLO has a noticeable IQR, as the performance depends on the initial state of the initial representation before applying the transform. Figure 6.10 explicitly shows the marginal influence of the input stream distribution and what practitioners may expect to save on bandwidth or storage in practice.

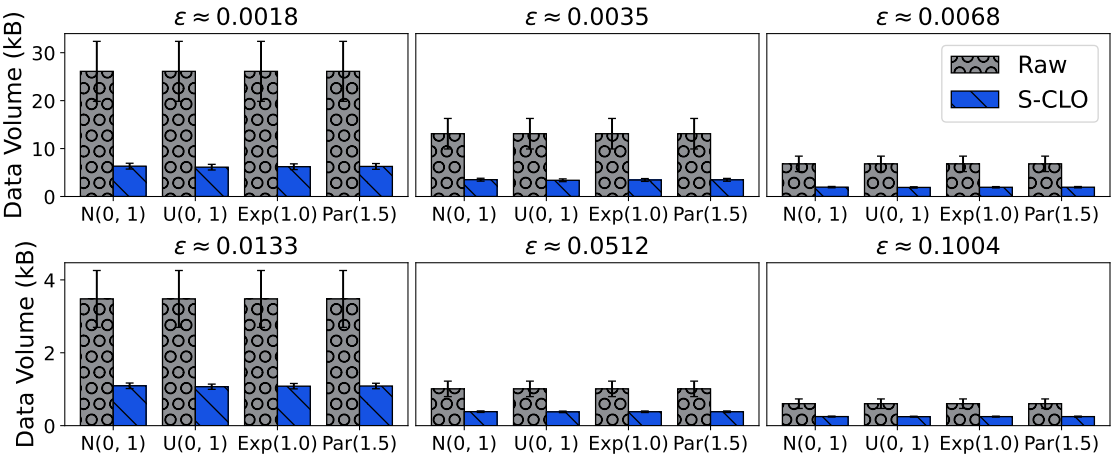


Figure 6.10: Absolute data volume reduction of S-CLO.

Raw volume is the serialized (uncompressed) in-memory size, identical for KLL and S-CLO. Notice that S-CLO alleviates the twofold cost discussed in Section 6.2, as the compression ratio of S-CLO scales with the target bound *and* the number of words. The larger the input data volume, the higher the multiplier, which decreases the output volume. Note that this is a *conservative* estimate, as it includes the overhead from LZMA, which is more suitable for larger file sizes. Practitioners will likely want to consider more suitable compression algorithms, unless they wish to encode multiple sequentially created sketches, which we handle in the following.

6.4.3 Performance on Jointly Transformed Instances

The instances are locally available and are jointly encoded. We assume that the input streams are non-overlapping chunks of a larger stream. The objective of J-CLO (Algorithm 4) is to find representations that remain separate, but *minimize the overall codeword of the concatenated input* by considering both intra-and inter-representation redundancy.

i.i.d samples from a common source

The experiments with i.i.d. streams are identical to those in the single instance evaluation, except we synthesize the stream with b times the samples, and pass b uniformly sized chunks as inputs to the construction algorithms. β-compressibility is defined as the ratio between the sum of separately compressed representations and the size of the compressed concatenation of those same representations. We will use β-compressibility to measure

how much of the scaling gains from increasing batch size result from improved coding efficiency through joint optimization. Figure 6.11 aggregates the stream sizes and dataset to compare the relationship between the batch size and compression ratio. S-CLO serves as a baseline that transforms each representation separately.

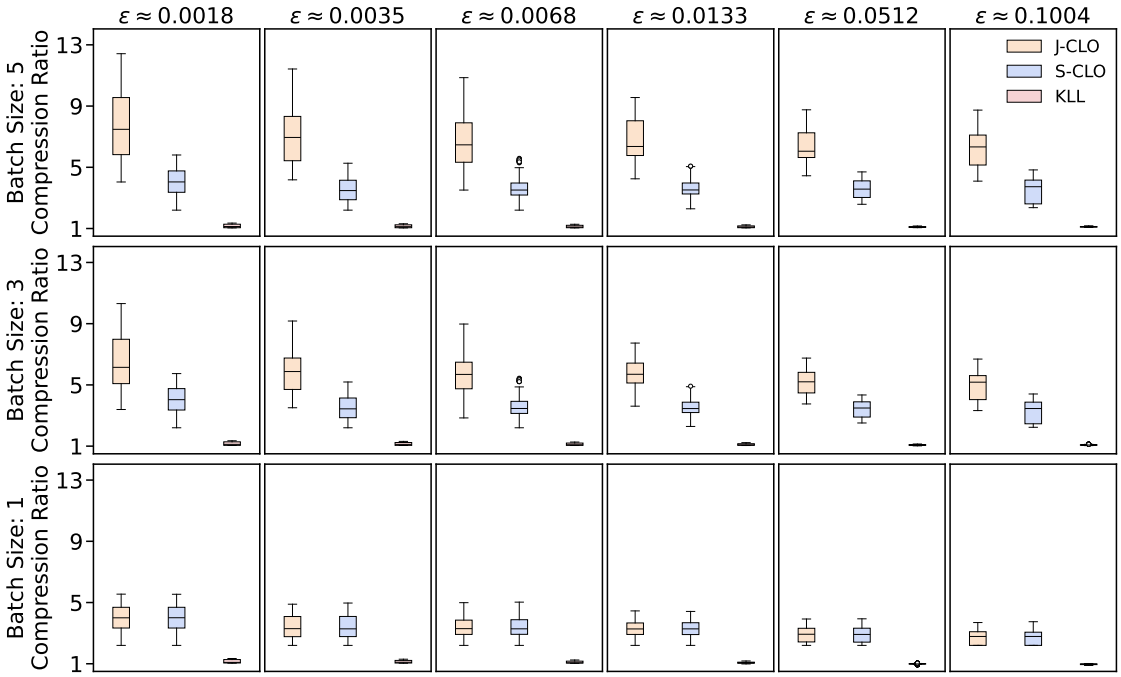


Figure 6.11: Contrasting relative performance for jointly optimized instances.

S-CLO and KLL have virtually no gain from compressing summaries as a batch. J-CLO jointly optimizes the summaries by exploiting intra- and inter-summary redundancy. At batch size 1, J-CLO and S-CLO behave identically as the J-CLO inputs are simply vectors of size 1, i.e., scalars. Notice that the less tight the error bound, the higher the gain from increasing the batch size. Since there is less intra-representation gain from less tight error bounds, J-CLO allocates more prefix error to exploiting inter-representation redundancy. The increase in interquartile range at larger batch sizes is expected, as β-compressibility is susceptible to sampling error intrinsic to empirical distributions. Increasing the batch size results in virtually no gain for S-CLO and KLL, as neither performs joint optimization.

Figure 6.12 again shows how marginal the influence is of the underlying distribution we sample from. Raw volume scales linearly in batch size, which may be prohibitively expensive for tighter error bounds. J-CLO can exploit input stream similarity by optimizing the representation of the corresponding summaries to decrease the volume significantly.

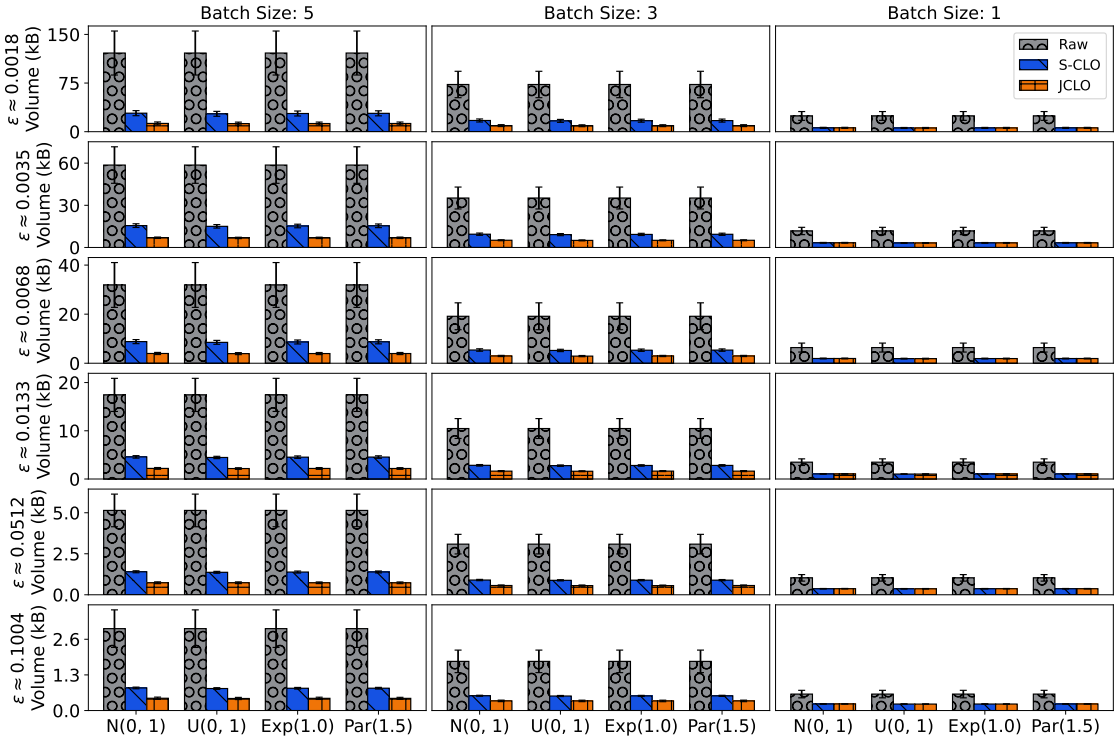


Figure 6.12: Absolute data volume reduction by batch size of J-CLO.

Efficiency gains from joint optimization

The following examines the effect of β-compressibility on overall compression gains for J-CLO. We create five datasets by sampling from a normal distribution and progressively increasing the range that overlaps between the minimum and maximum values.

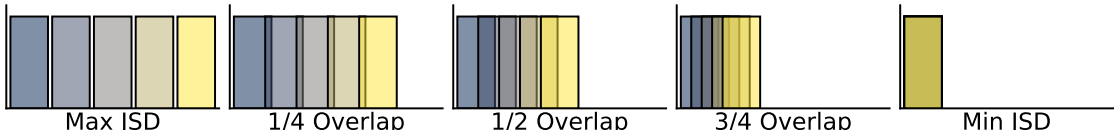


Figure 6.13: Synthesizing distributions with overlapping ranges.

Figure 6.13 illustrates the concept for batch size 5. Each bin represents a value range of a uniform distribution from which elements are sampled, but the number of values is unbounded. For example, 1/4 overlap means that there is up to 25% inter-representation redundancy. Analogous to previous experiments, we report results aggregated by target epsilon to show the full range of representation outcomes. Figure 6.14 shows the relationship between the target bound, compression ratio, and input stream similarity.

The larger the overlap, the larger the gain in compression ratio from exploiting inter-representation redundancy. Notice the widening gap between compression ratio and

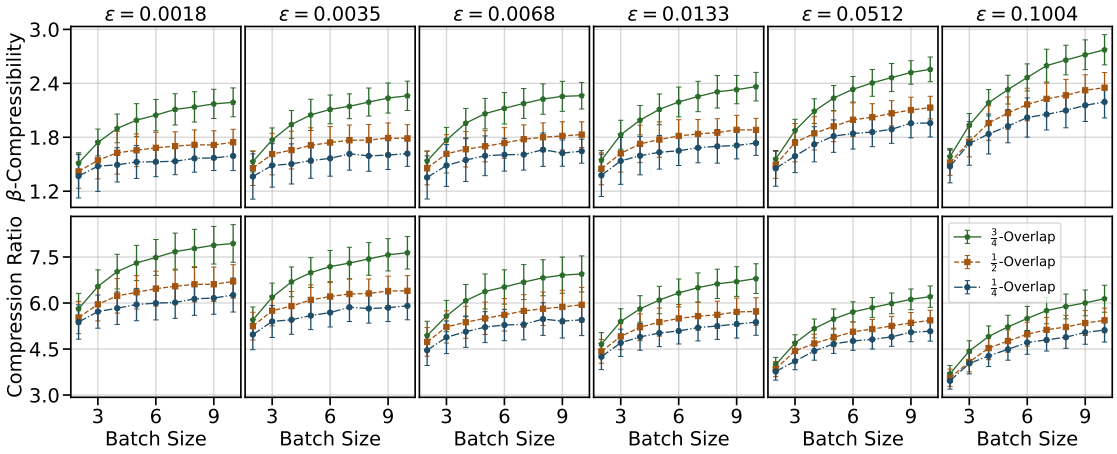


Figure 6.14: Measuring the efficiency gain from batch scaling at different target bounds.

β-compressibility as the error bound becomes less tight. The decrease in compression ratio when reducing the overlap is partially mitigated by J-CLO allocating prefix error to exploit intra-representation redundancy. β-compressibility increases by overlap and error bound since there is less to gain from intra-representation redundancy at less tight bounds. Conversely, the tighter the error bound, the less the adverse effect on the compression ratio from decreasing β-compressibility. Figure 6.15 shows the results for the extreme cases (Min/Max JSD) where there is maximal or no overlap. The input streams are identical or have no statistical similarity.

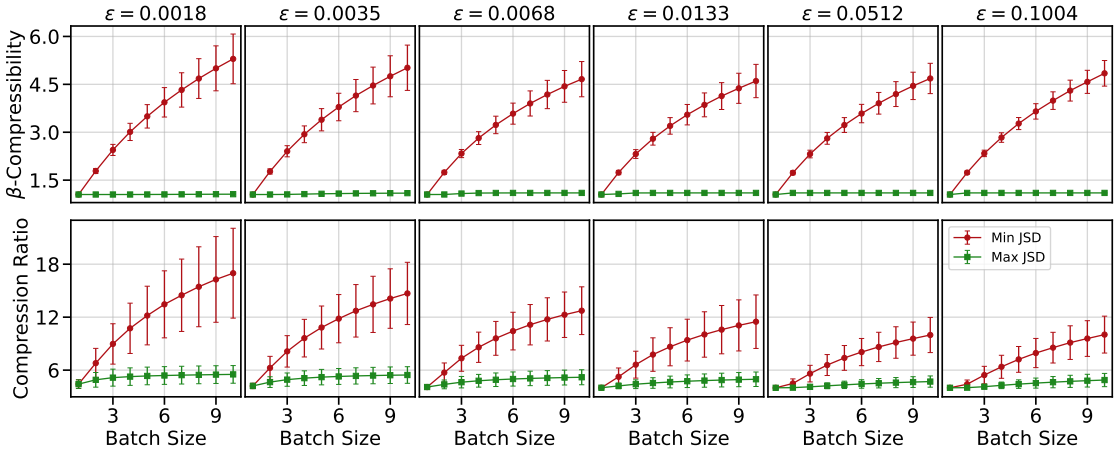


Figure 6.15: Measuring efficiency gain for extreme cases.

For Min JSD, the β-compressibility does not scale linearly, since J-CLO allocates prefix error to exploit intra-representation redundancy, resulting in an overall compression rate larger than the batch size. Conversely, the input streams have no statistical similarity for Max JSD, so J-CLO yields the same compression ratio as if all instances were encoded separately.

Scaling Rate Degredation

The final experiments determine whether J-CLO can handle drifts, with experiments that examine whether the rate of compression ratio scaling degrades according to the drift rate at large batch sizes. The dataset is engineered to control how much mass two neighboring chunks share using Gaussian mixture models. Each stream follows the mixture distribution $(1 - w) \cdot N(0, 1) + w \cdot N(\mu_i, \sigma_{\text{bump}})$, where $\sigma_{\text{bump}} = 1/b$ to account for the scaling batch size.

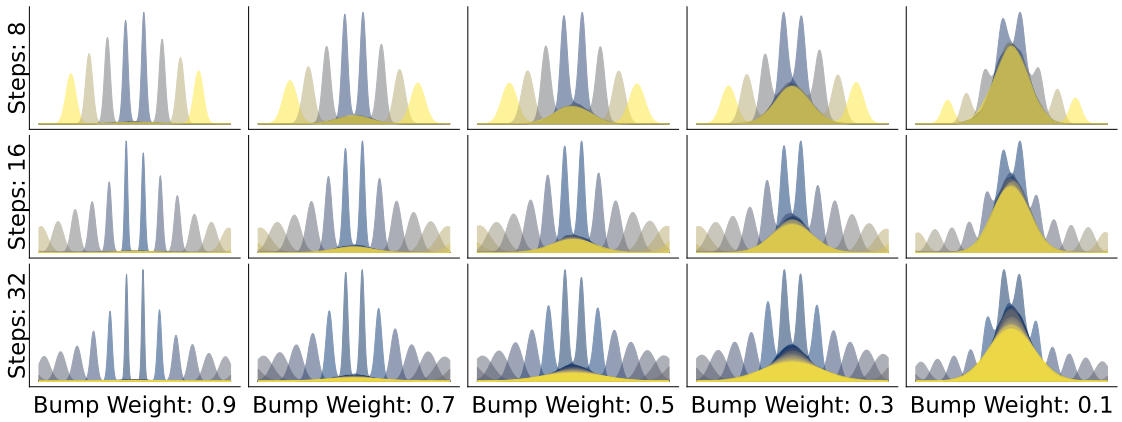


Figure 6.16: Gaussian Mixture Models that scale location shift by the number of steps to keep the shared mass between two input streams uniform.

Figure 6.16 illustrates the concept at different batch sizes. The bump component linearly shifts the mean μ_i from $-b/2$ to $+b/2$ across b streams. The weight w controls the ratio of moving and shared mass.

Figure 6.17 summarizes the results by showing the relationship between the bump weight, the target bound, and the compression ratio. Concept drift decreases the gain from joint optimization. However, the gain gracefully degrades relative to the drift's magnitude. While the scaling rate is small, since the empirical distributions with $w = 0.1$ are challenging to compress, the results show that the algorithm can exploit faint sources of redundancy, as even for $w = 0.9$, the compression ratio scales with the batch size.

6.4.4 Error Analysis

The experiments compare the normalized rank error of KLL with the augmentations. We sample from a normal distribution to synthesize streams, as it represents the type that the augmentations modify representations the most.

Table 6.3 shows the max and mean normalized rank error, averaged across 100 trials.

Max error measures the highest absolute difference between the true and the approximate at a rank. While the theoretical framework established in Section 6.3.1 that the error remains bounded, it is expected that the mean error marginally increases. However, we

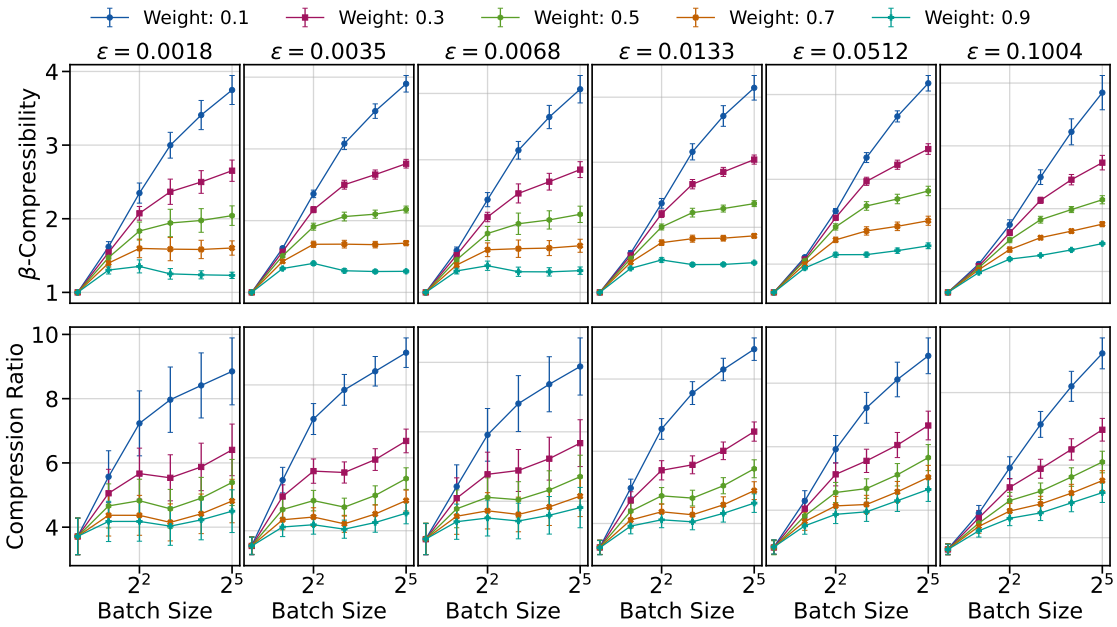


Figure 6.17: Measuring the efficiency gain from batch scaling under concept drift.

emphasize that from Lemma 2 it **strictly** follows that any slight increase in expected error implies a significant reduction in data volume. Figure 6.18 explicitly shows the confidence intervals for the tightest and least tight intervals. We filtered for the configuration that resulted in the number of words and measured the highest compression gain for the target bound. Notice that for KLL, there is roughly an equal margin for separate and merge queries. For S-CLO, the single instance queries are pushing against the boundaries, but the error "amortizes" back to a similar margin as KLL when merging. J-CLO additionally pushes against the error envelope on the union stream, since the instances were transformed jointly before the merger.

6.5 Summary

Besides introducing a generic framework for transparently augmenting sketches, we realized algorithms for the single and joint cases that readily apply to KLL and related algorithms. Extensive empirical analysis showed that, on typical sources, S-CLO improves coding efficiency considerably in linear time. Within the scope of a single study, the formal and empirical analysis has only explored using entropy as the cost function. The practical value of the proposed approach is not only to augment the sketch algorithm transparently, but also to accommodate arbitrary compression algorithms *without* engineering custom codecs. Hence, keys may be remapped using a cost function that matches a codec's assumptions to improve coding efficiency further. The crucial part is that such optimization does *not* require the sketch algorithm to make assumptions on the input. It is merely a flexible method to significantly reduce the data volume of the encoded

Table 6.3: Normalized Error Summary

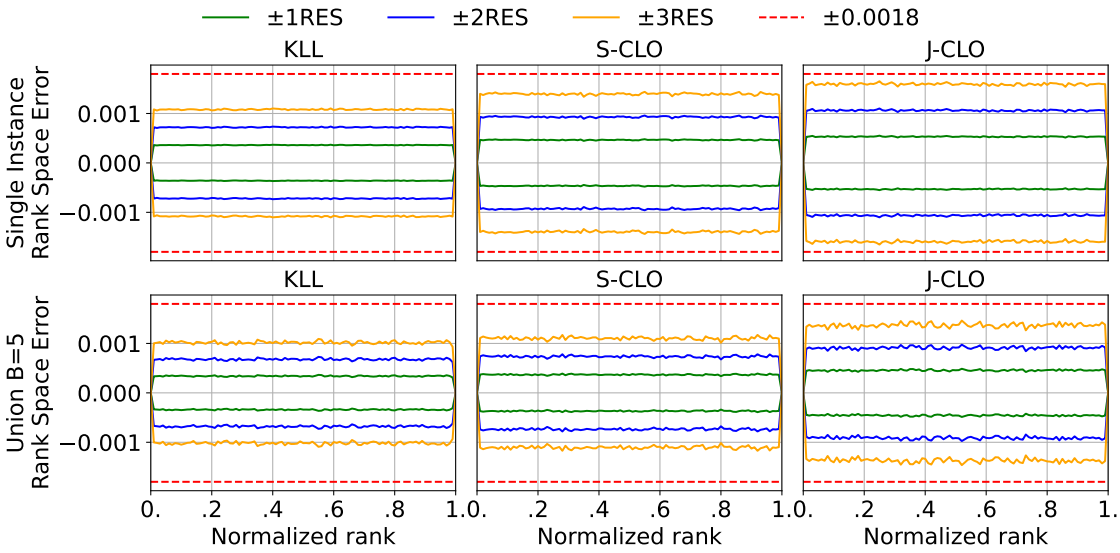
(a) Max Normalized Rank Error

	Separate Encoding			**Joint Encoding (Merge)**		
Target Bound	KLL	S-CLO	J-CLO	KLL	S-CLO	J-CLO
0.0018	0.001134	0.001248	0.001191	0.000974	0.001015	0.001249
0.0035	0.001729	0.001847	0.001836	0.001799	0.001802	0.002049
0.0068	0.004731	0.005044	0.005049	0.003773	0.003889	0.004847
0.0133	0.008126	0.008515	0.008253	0.007732	0.007799	0.008016
0.0261	0.020066	0.021031	0.021003	0.018892	0.018970	0.019323
0.0512	0.033123	0.037358	0.037557	0.029889	0.030014	0.037324
0.1004	0.079329	0.082539	0.081982	0.064143	0.068473	0.077228

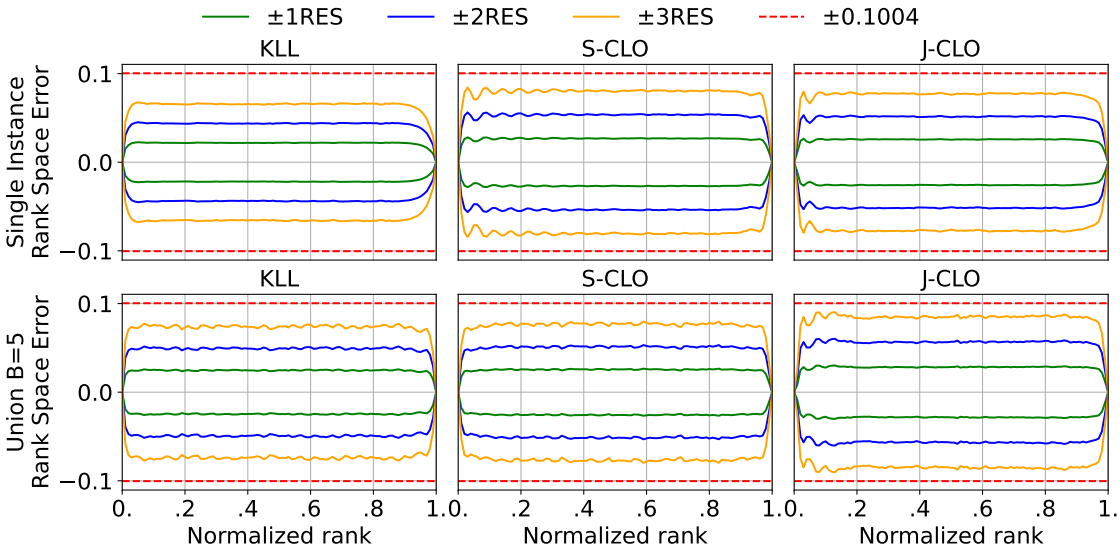
(b) Mean Normalized Rank Error

	Separate Encoding			**Joint Encoding (Merge)**		
Target Bound	KLL	S-CLO	J-CLO	KLL	S-CLO	J-CLO
0.0018	0.000378	0.000408	0.000449	0.000344	0.000361	0.000417
0.0035	0.000592	0.000641	0.000857	0.000588	0.000593	0.000802
0.0068	0.001676	0.001721	0.001785	0.001345	0.001406	0.001608
0.0133	0.002637	0.002928	0.002903	0.002645	0.002736	0.003401
0.0261	0.007005	0.007439	0.007524	0.005377	0.006073	0.006444
0.0512	0.011228	0.012139	0.012403	0.010638	0.011486	0.012087
0.1004	0.027958	0.029278	0.029153	0.022382	0.022848	0.025989

representation with minimal changes.



(a) Target Bound $\varepsilon \approx 0.0018$



(b) Target Bound $\varepsilon \approx 0.104$

Figure 6.18: Confidence intervals using the Relative Standard Error (RSE).

CHAPTER 7

Conclusion

The thesis has addressed fundamental problems of massively distributed systems with methods for efficient data transmission and reliable recovery of salient information. It first introduced a reference architecture to seamlessly deploy and scale applications in a hierarchical network with classical and quantum components. The key insight is the importance of seamless integration and that scaling such platforms is primarily a compression problem. Then, it discussed sound methods and experiment design for ML research in edge-cloud systems, where reproducibility and drawing conclusions about whether reported results generalize into real-world environments is challenging. It has demonstrated how advancements in orthogonal research areas can invalidate relative performance differences and introduced a research tool to streamline graph compiler benchmarking over heterogeneous compute infrastructure to facilitate iterative empirical analysis. The following two chapters concern handling data volume generated by request payloads for ANN inference. It investigated the problem at opposite extremes in network availability. In urban environments, latency is the priority, whereas in satellite computing with intermittently available connectivity, it is throughput measured by the expected transfer cost reduction. The thesis has also considered the modality expected for systems monitoring with advancements in stream algorithms. To maintain server-side transparency, it introduced augmentations to existing algorithms and a verification procedure that provably maintains the strong underlying mathematical properties.

We conclude the thesis by summarizing the contributions to answering the three research questions posed in Section 1.2 and a discussion on future work.

7.1 Research Questions

RQ1. How can we design (I) Methods and (II) Experiments for Machine Learning Research in Edge-Cloud Systems? Admittedly, this research question does not lend itself completely to methodologies that provide accurate answers. For the

problems the thesis addresses, formal methods can only help show correctness, and no empirical analysis in isolated environments can provide irrefutable evidence on claims regarding real-world efficacy. Still, the initial observation on how readily, risk-free, and free optimization methods may significantly change relative performance rankings has motivated the thesis to focus on fundamentals and report conservative estimates on expected performance gains. Refraining from convoluted systems, overengineered algorithm implementations, and complex scenarios has permitted us to evaluate our methods with readily reproducible experiments extensively.

RQ2. What approach exploits local resources best to (I) transparently meet the requirements of (II) latency-sensitive and (III) system-critical applications? When offloading to large, remotely deployed ANNs, three competing approaches exist for exploiting locally available hardware accelerators. Empirical results have shown that task partitioning by partially unloading ANN inference is an excessively inefficient use of local resources, due to resource asymmetry. Alternatively, local resources may be used to compress the data using NIC, which can significantly reduce transmission costs due to recent advancements. The advantage is that it requires the least effort for operators and naturally generalizes to any ANN architecture and task. However, even at the highest quality levels, there is a significant loss in prediction performance, which defeats the purpose of offloading, as applications may just execute a quantized and distilled version of foundational models locally. Feature compression with an information bottleneck is most promising in preserving the integrity for downstream tasks at significantly reduced bit rates. The downside is that existing methods assume knowledge of the downstream tasks and rely on large encoders that must be sufficiently deep to extract the high-level semantic information. The thesis has handled the limitations with *SVBI* that trades off higher bitrates for (near-)task agnosticity using an encoder with fewer parameters by orders of magnitude than deep methods. A follow-up extended SVBI to elaborate on the recovery of human interpretable data and to handle environments with intermittently available connectivity. Experiments compared SVBI extensively against competing approaches and have Our experiments with SVBI give conclusive evidence that omitting any requirements that reduce server-side processing time will not only yield disproportionally lower bitrate, but may also reduce overall request latency due to the reduced transmission costs. Lastly, FOOL has explicitly used metrics that measure the gain in reduced data volume by seconds and have convincingly outperformed baselines on efficiency.

RQ3. What are (I) suitable approaches to construct statistical summaries, and (II) can we improve their coding efficiency without breaking server-side transparency? The type of algorithm, namely *fully-mergeable quantile sketches with strong-guarantees in an unbounded universe model*, was chosen according to the requirements in Section 1.2. There are no assumptions on input distribution or data type properties. It is based on a well-developed and sound mathematical framework that provides strong worst-case guarantees on approximation error. The guarantees

establish trust for analytics tasks and automated decision mechanisms. The combination of mergeability and sublinear memory requirements, enabling hierarchical processing on mobile devices and servers, makes it a natural fit for a compute continuum. Mature implementations exist and are widely deployed in legacy systems.

The thesis has shown that improving coding efficiency with server-side transparency is feasible by introducing *Syntax- and Order Preserving Transforms*. It maintains the same guarantees as the underlying algorithm while finding representations that are more amenable to compression algorithms. Its adoption is as simple as overriding a serialization method with a few lines of code.

7.2 Preliminary Results of Future Work

This section discusses non-speculative future work that we find worth pursuing, based on preliminary results that were presented in smaller conferences.

7.2.1 Adversarial Robustness of Bottleneck-Injected Neural Networks

The pervasiveness of ANNs exposes significant vulnerabilities to adversarial attacks [AM18]. Since our work advocates increased pervasiveness of ANNs for communication, we found it necessary to conduct a study that discusses the implications. The optimistic view is that perturbations are intrinsically redundant information, and the IB objective naturally enhances the robustness of ANNs by learning to discard redundancy more aggressively along their information processing path [SZT17]. However, the established consensus on the value of IB-based objectives is based on the assumption that the networks are *deep*. Yet, task-oriented communication is feasible only when paired with lightweight compression such that the codec computational overhead is offset by the reduced transmission costs [MVH$^+$25, PXL$^+$24]. There are additional constraints on the encoder design. While a neural encoder may still be wide enough to leverage parallelization from onboard AI accelerators, meeting stringent latency requirements demands reducing the number of sequential operations. Hence, envisioned future communication networks that rely on neural encoding schemes will realistically converge towards *shallow* networks. To this end, the study is dedicated to investigating the robustness of methods that apply an IB-based objective intended for communication systems that use IB objectives to reduce transmission costs.

We apply several common adversarial attacks on recent approaches based on SVBI [FRD24, FZR$^+$25], and contrast their efficacy with the conventional deep variational information bottleneck [AFDM16]. The preliminary results show that deep networks trained with a traditional IB objective exhibit higher adversarial robustness than SVBI. Still, a shallow variational encoder is considerably more robust than purely discriminative models trained with non-IB objectives. We finalize our experiments by accentuating the increased attack surface of systems that rely on generative models for communicating salient information with a simple attack specifically targeting generative models. In other words, the overall

system is more vulnerable even if task-oriented communication is intrinsically more robust than passing messages through conventional channels for downstream tasks. While a comprehensive study is not within the thesis's scope, we hope our results and insights can facilitate research securing next-generation communication systems.

Attack Surface of IB-based Communication Systems

Figure 7.1 illustrates a simplified but sufficient view of the communication system.

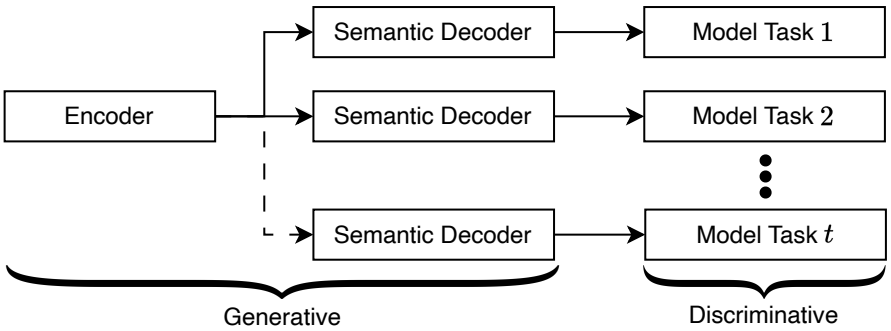


Figure 7.1: Attack Surface of IB-Based Communication Systems.

Before passing the input to a discriminative model for prediction, it is processed by a generative model. Arguably, even if training the codec with an IB-based objective improves adversarial robustness against attacks intended for discriminative tasks, it may still increase the attack surface of the overall communication system due to the generative components.

Considered Adversarial Attacks

The susceptibility of ANNs to adversarial examples was first investigated by Szegedy et al. [SZS+14], who demonstrated that small, imperceptible perturbations to input data can lead to significant misclassifications. Adversarial attacks are classified into white-box and black-box attacks. White-box attacks assume complete model knowledge, including architecture and gradient calculation, allowing for computing highly effective adversarial samples. In contrast, black-box attacks assume no access to model details and are generally more challenging but more realistic for real-world scenarios.

The following briefly describes the attacks we have chosen due to their influence and being subject to numerous follow-up studies. The focus is on white-box attacks due to the open nature of ML research and the popularity of readily available open-source weights for a wide range of tasks. *Fast Gradient Sign Method (FGSM)* by Goodfellow et al. [GSS15] efficiently generate adversarial examples by leveraging the gradient of the loss function. FGSM adjusts the input along the gradient's direction, with the perturbation defined as:

$$x_{adv} = x + \epsilon \cdot \text{sign}\left(\nabla_x J(x, y)\right) \tag{7.1}$$

where x is the input, ϵ controls perturbation magnitude, and $\nabla_x J(x, y)$ is the gradient of the loss concerning the input.

The attack by *Carlini and Wagner (C & W)* [CW17] minimizes the L_2, L_0, or L_∞ distance between the original input and the adversarial example, and a term that penalizes classifications other than the desired target class using the objective function:

$$J(x') = \alpha \cdot ||x - x'||_p + \beta \cdot \mathcal{L}_{\text{mcls}}(f(x'), y_t) \tag{7.2}$$

where x' is the perturbed input, α, β balance the terms, and $\mathcal{L}_{\text{mcls}}$ is the missclassification loss. Notably, this attack is shown to be highly effective against networks pre-trained on ImageNet, which are commonly used to finetune for practical recognition tasks.

The *Elastic-Net Attacks on ANNs (EAD)* [CSZ+18] is particularly useful for producing sparse perturbations, which can trick ANNs while maintaining minimal changes to the input. It generates adversarial samples by minimizing the objective

$$c \cdot f(\mathbf{x}, t) + \beta \left\|\mathbf{x} - \mathbf{x}_0\right\|_1 + \left\|\mathbf{x} - \mathbf{x}_0\right\|_2^2 \tag{7.3}$$

where $f(x, t)$ is a target loss function and $c, \beta \geq 0$ are the regularization parameters. EAD's dual-norm optimization is an interesting alternative benchmark for evaluating how variational bottleneck injection handles diverse attack strategies.

The Jacobian-based Saliency Map Attack (JSMA) attack by Papernot et al. [PMJ+15] constructs adversarial examples by identifying and perturbing input features most critical to the classifier's decision-making process. Unlike gradient-based methods, JSMA uses forward derivatives to create a saliency map, guiding perturbations to specific input features. Given that variational bottleneck techniques may alter feature representations, testing JSMA will allow us to explore how bottleneck injection influences feature saliency and adversarial resilience.

Lastly, we include the attack introduced by Tabacof et al. [TTV16] to demonstrate the increased attack surface of communication systems that deploy generative models. This attack disrupts reconstruction and induces the encoder to produce a completely different target image. This would undermine the potential defensive role of autoencoders in de-noising classifier inputs. Note that the efficacy of the attack towards the autoencoder is irrespective of whether we map the latent to an approximation of the original image (i.e., reconstruction) or use it for some image recognition downstream task [FZR+25].

Experiment Design

We generate adversarial samples using the torchattacks [Kim20] library. Except for the Tabacof attack, we create samples for *CIFAR-10*, textitSVHN, and *ImageNet64* (i.e., downsampled ImageNet but still using all original 1000 classes). Notably, we choose JSMA as it may provide a different perspective on model vulnerability by perturbing specific input features. However, JSMA has high memory requirements, which we cannot

accommodate with our limited resources for ImageNet64. Therefore, we implement a modified version of JSMA (JSMAOnePixel) that is inspired by [Son]. The OnePixel variant identifies only a single pixel with the highest impact on each iteration. As shown in Figure 7.2, the final perturbed image is comparable between JSMA and JSMAOnePixel.

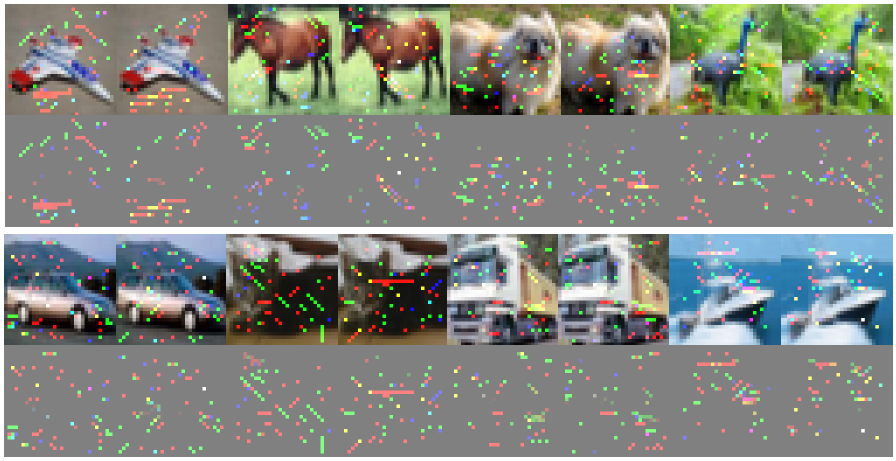


Figure 7.2: In pairs, comparing JSMA (left) with JSMAOnePixel variant (right).

Unlike with other attacks, we use MNIST for Tabacof as the purpose is to demonstrate the widened attack surface of IB-based communication. We train three sets of models. Baseline models with standard log-loss, models with a shallow bottleneck (SVBI), and models with a deep bottleneck (DVIB). We perform DVIB and SVBI as described in Chapter 4. For DVIB, we place a bottleneck at the penultimate layer and use a log-loss for the distortion term in the objective function. For SVBI, we follow the "blueprint" encoder design that replaces the layers until the first high-level block of the network (roughly 1% of the total model parameters) with a small variational autoencoder. We experimentally determine the lowest possible bitrate for both bottleneck approaches without sacrificing prediction performance.

Table 7.1 summarizes the model performances on the non-adversarial samples. The bits

Table 7.1: Models Prediction and Compression Performance

Dataset	Acc@1 [%]	Bpp (SVBI)	Bpp (DVIB)
MNIST	97.36 ± 1.77	0.0829	0.0161
CIFAR-10	85.25 ± 1.40	0.5677	0.0308
SVHN	94.04 ± 0.69	0.4321	0.0086
ImageNet64	49.36 ± 1.12	1.2673	0.0115

per pixel (bpp) is a lower bound we have empirically determined for a bottleneck injected model to perform (near-)lossless prediction as defined in [FRD24, FZR+25]. Naturally, DVIB has much lower bitrates due to task specificity, as described in Section 4.1.3.

Comparing Bottleneck Placements

Table 7.2 summarizes the effect on the adversarial samples represented as percentage points (lower is better). Unsurprisingly, base models trained using a standard log-loss have a significant drop in accuracy. Relative to the accuracy on the unperturbed dataset (Table 7.1), all attacks completely tank the model's performance. In particular, for the SVHN task, the performance is at times worse than random guessing. As conjectured, SVBI generally provides less adversarial robustness than DVIB across all datasets. Notably, task complexity apparently influences the gap in adversarial robustness between SVBI and DVIB. Still, SVBI exhibits considerably higher adversarial robustness than the baseline. Additionally, notice that the model depth on the base model does not considerably affect adversarial robustness. However, for the DVIB model, depth seems to correlate positively with adversarial robustness. Presumably, since deeper models have longer information paths, end-to-end training models with an IB objective have more opportunities to discard information that does not contribute to task performance gradually.

Table 7.2: Comparing Prediction Performance Decrease (% points; lower is better) between Objectives.

		CIFAR-10			SVHN			ImageNet64		
Model	Attack	Base	SVBI	DVIB	Base	SVBI	DVIB	Base	SVBI	DVIB
ResNet-18	FGSM	74.5621	48.7298	39.513	69.9521	55.8298	48.5728	37.7602	31.8935	28.9807
	EAD	85.5256	9.6447	8.8592	89.9427	19.8432	13.5824	35.4344	9.2381	8.0993
	C&W	87.0232	7.7732	6.7682	92.2149	22.9992	18.3259	37.9903	12.5742	10.2117
	JSMA	87.6210	20.7807	17.5784	91.5810	14.2348	11.1283	36.3821	11.1868	10.1935
ResNet-50	FGSM	68.5621	42.8942	34.0803	68.2679	53.2118	45.1977	39.6985	28.9273	23.1021
	EAD	85.1400	9.1258	7.8592	89.3852	18.0232	11.4375	32.6361	7.0377	4.8375
	C&W	88.9231	7.2009	6.5408	93.3284	18.0931	15.3259	34.1083	9.9654	5.4281
	JSMA	85.1010	19.6333	16.0549	90.2838	13.9125	10.1283	34.4847	10.1351	5.5213
ResNet-101	FGSM	69.3189	40.8912	32.1534	66.2082	40.4817	42.9004	38.4451	24.0620	20.9997
	EAD	87.6557	8.0322	7.8592	88.3294	16.4385	9.5729	32.4148	6.1124	3.0489
	C&W	87.4633	7.1819	6.0018	92.1923	17.3284	12.2482	38.0200	8.7985	2.9663
	JSMA	86.8781	19.439	15.2608	94.2933	11.2814	7.9833	36.6825	10.0382	1.4762

Analyzing Pixel Perturbations

We observe that IB-based objectives exhibit stronger robustness against attacks that focus on a small subset of salient pixels with strong intensity than attacks that perturb many pixels with smaller intensity. Moreover, similarly to the original work on deep variational IB [AFDM16], we observe that attacks targeting IB-based models perturb pixels considerably more than non-IB-based models. Nevertheless, since relative values align across all models (i.e., attacks behave comparably regardless of the model depth or objective), the following reports average values due to space constraints. Figure 7.3 visualizes the L_0 norm by attack averaged over test sets, i.e., it measures how many pixels an attack has perturbed.

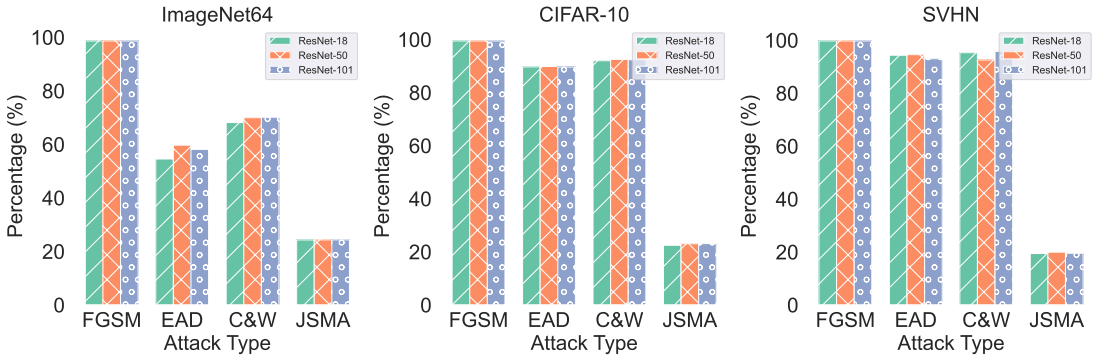


Figure 7.3: Average Percentage of pixels perturbed by an adversarial attack.

While FGSM perturbs nearly all pixels, JSMA only perturbs roughly 20% of the pixels. More interestingly, EAD and C&W perturb fewer pixels for ImageNet than for the simpler tasks. Generally, more complex tasks with many labels rely on more fine-grained information, where just a small subset of salient pixels can influence the decision boundaries. Figure 7.4 summarizes the magnitude of perturbations.

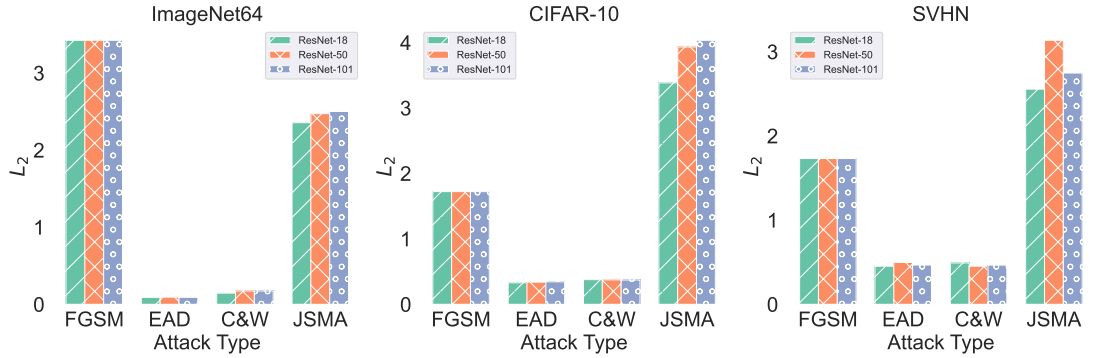


Figure 7.4: Measuring the magnitude of perturbations with the average L_2.

FSGSM and JSMA incur considerably higher perturbations than EAD and C&W. Further, notice that JSMA has a larger total magnitude in total perturbation than FGSM, despite JSMA focusing on a smaller subset of pixels. The reason becomes apparent when examining the L_∞ norm in Figure 7.5.

JSMA is more "pixel-efficient" by focusing on the most salient pixels but relies on high-magnitude perturbations. Figure 7.6 visually compares JSMA and FGSM.

While FGSM perturbs a large number of pixels, they are only faintly visible. Conversely, JSMA has clearly visible perturbations. This observation is consistent with the general objective of IB-based communication, which is to focus on the most salient information. Therefore, it may be reasonable to emphasize evaluating defense strategies for task-oriented communication against less perceptible attacks.

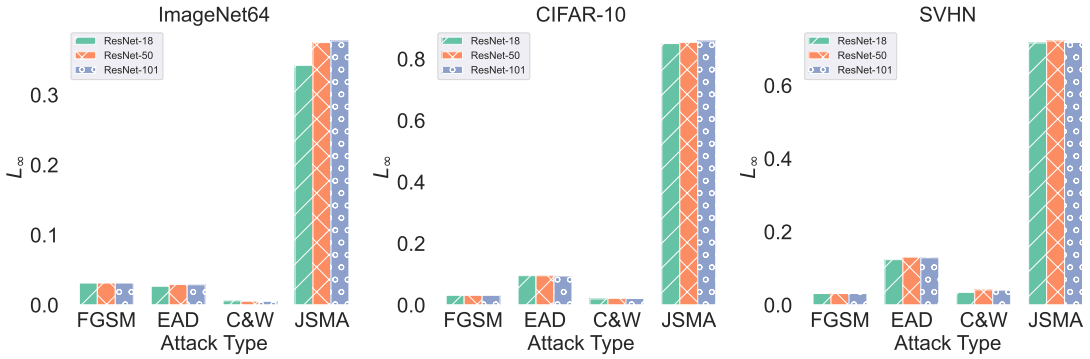


Figure 7.5: Average L_∞ measure to quantify the magnitude of perturbation.

Figure 7.6: Comparing magnitudes of pixels between JSMA and FGSM.

Targeted Autoencoder Attack

We briefly show the increased attack surface with the Tabacof [TTV16] attack. The results are summarized in Table 7.3.

Table 7.3: Tabacof attack

	Base		DVIB	
Model	Acc@1	# Hits	Acc@1	# Hits
Resnet-18	61.6	927	52.26	1802
Resnet-50	76.57	879	72.66	1126
Resnet-101	33.17	448	34.73	1641

We choose the label "1" as the target, and the *hits* column indicates how often the model has predicted "1" after the attack. Since the models have near-perfect accuracy on MNIST, and the test set has 10 000 samples that are uniformly distributed, we can infer the efficacy of the attack by the increase in predictions of "1". Figure 7.7 shows an example with a curated sample of perturbed images.

Examples depicting the numbers two and four contain the most clearly visible perturba-

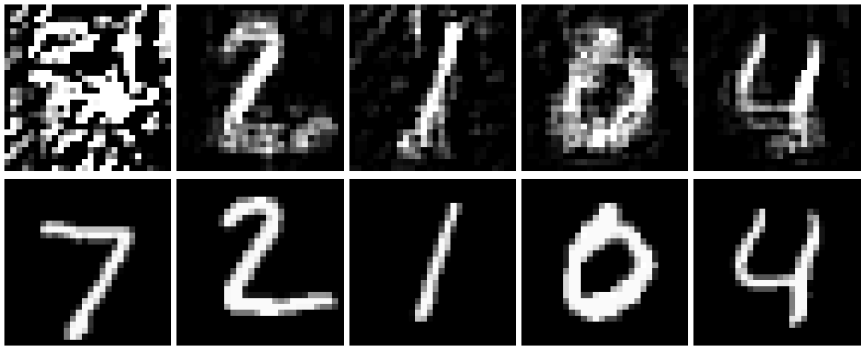


Figure 7.7: Base images (bottom) and corresponding perturbation using the Tabacof attack against DVIB (top).

tions to match this target. The leftmost example showcases a "failed" attack, where the network will likely missclassify the input, but not hit the intended target. ResNet-101 is noticeably less robust toward the attack. We explain the discrepancy by the simplicity of the task and the dataset size. Since ResNet-101 is significantly larger, the model may have been fitted to the samples, making it more susceptible to even slight perturbations. Still, comparing the performance of ResNet-18 and ResNet-50 shows that the DVIB model is considerably *less robust* than the baseline model. Notably, all DVIB models have a substantial increase in predicting the target label, indicative of the attack's efficacy.

Warm Starting Hybrid QNNs

Chapter 2 has extensively discussed the integration of both classical and quantum resources in a distributed inference engine. While Chapter 4 reflects the classical parts, a comprehensive method that exploits (simulated) quantum resources is not within the scope of the thesis. However, we have conducted a short preliminary study that empirically demonstrates how SVBI may be used as a warm-starting method as described in Section 2.2.7.

Circuit Design The compression model is taken directly from Section 4.2.2 without saliency guidance using Swin-Tiny as the teacher. However, here we replace the classical classifier for a task with a hybrid QNN. Figure 7.8 illustrates the Ansatz of the QNN with four qubits and layers.

The hybrid QNN takes as input the n-dimensional real-valued feature vector $\mathbb{Z}^n$ and classically projects it to a vector with dimensions equal to the number of qubits. Then, it passes the features as input to the Ansatz. Regardless of circuit depth, it first applies a Hadamard H Gate and a parameterized Z-rotation RZ to embed features in the quantum node. Next, it applies a repeating sequence of trainable variational layers. A layer consists of pairwise (shifted) C-NOT gates followed by alternating parameterized Y- or Z-rotations, i.e., a layer with Y-rotation is followed by a layer with Z-rotations. The

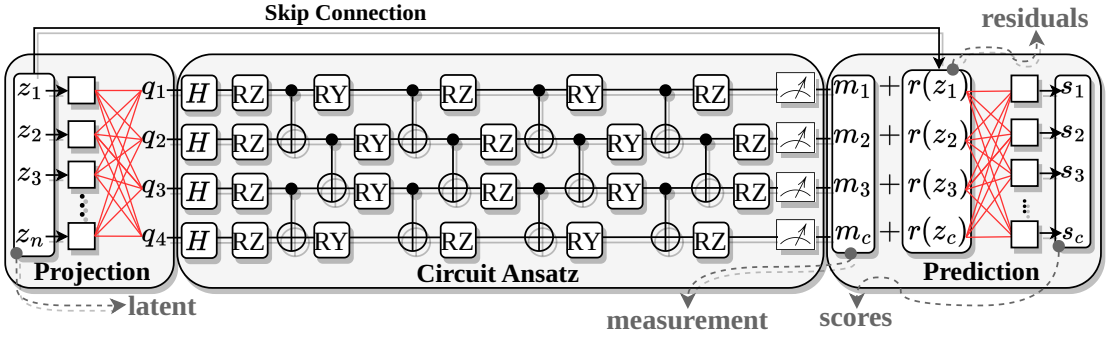


Figure 7.8: Hybrid QNN with Four Qubits and Layers

number of layers is a hyperparameter given by the depth of the circuit. Lastly, it passes the measurements to a fully connected layer to output the class scores. The circuit is optimized in a *noise-free* simulation using PennyLane [BIS+18].

Dataset Selection To emulate the scenario of Figure 2.8 with clients requesting inference from diverse environments, we create four thematically unrelated datasets from the 1000 labels from the ILSVRC [RDS+15] classification task. Compared to the main experiments, we evaluate task performance on smaller datasets derived from ILSVRC due to resource limitations. Table 7.4 summarizes the dataset properties.

Table 7.4: Training and Test Subsets of ILSVRC for Warm Starting

Task	Classes	Training Samples	Test Samples
Nutriment	10	13'000	500
Felidae	13	16'900	650
Buildings	14	18'200	700
Vessels	23	29'900	1150

Each dataset represents a different location requiring separate predictors. The accompanying repository contains a script and instructions to recreate the datasets.

Prediction Performance of Hybrid QNNs with compressed features We run experiments with 4, 6, 8, and 10 qubits with a classical predictor as the baseline. The depth of the circuit is fixed at 8. We performed additional experiments with varying depth sizes and found that increasing the depth yields diminishing accuracy improvement.

Table 7.5 summarizes the results. The Plots in Figure 7.9 and Figure 7.10 show how a hybrid model without and with the skip connection compares to their classical counterpart for each dataset. Relative to the classical baselines, Hybrid QNNs *without* skip connections gradually approach comparable, albeit still worse, Top-1 error as we increase the number of qubits. For 4 and 6 qubits, the Top-1 error is roughly 20-30% worse while the

Table 7.5: Top-1 (Err)or of (C)lassic, (H)ybrid, Hybrid with (Res)iduals

	Qubits	Top-1 Err. C. (%)	Top-1 Err. H. (%)	Top-1 Err. H. Res. (%)
Nutriment	4	13.00	37.13	12.11
	6	11.73	25.20	10.40
	8	11.30	16.90	10.07
	10	10.69	14.80	9.58
Felidae	4	19.82	31.57	19.05
	6	17.60	29.07	16.77
	8	16.77	18.77	15.85
	10	16.56	18.31	15.31
Buildings	4	9.29	31.57	8.57
	6	7.26	14.86	6.86
	8	6.14	10.57	5.71
	10	5.74	9.00	5.27
Vessels	4	32.43	62.00	30.69
	6	27.82	48.52	26.00
	8	25.48	31.56	24.96
	10	24.26	25.87	23.91

difference narrows to 2-5%. Interestingly, Hybrid QNNs *with* skip connections consistently outperform the classical baselines across all numbers of qubits. Although the motivation of skip connections in classical residual networks is to mitigate accuracy saturation for very deep neural networks, they essentially learn a residual function. Considering the autoencoder-backbone pair already heavily processes the input data, we hypothesized that a QNN could find more suitable representations for the classical features for some instances. In contrast, a QNN could decrease the performance for samples already sufficiently processed for classification. The QNN narrowing the performance gap with increasing qubits is consistent with our assumptions. The model without a skip connection cannot find a representation as good as the initial input for a low number of qubits. Conversely, the QNN with a skip connection can learn the residual function and sees a performance gain for the samples, where a quantum embedding is useful.

A skip connection was the first intuitive attempt to provide empirical evidence, and the initial results seem promising. Nevertheless, we remind the reader that this only serves as a PoC to determine whether our vision is viable. The evaluation strategy is not extensive enough, and thus, our results should not be considered conclusive. Moreover, even with the skip connection, the hybrid model only marginally outperforms the classical model across all task simulations with a *noise-free* device. We did not experiment with optimization algorithms more appropriate for QNNs. Further, the backbone is classical, i.e., the extracted features may be biased favorably towards classical predictors. Future work can significantly improve our results by experimenting with

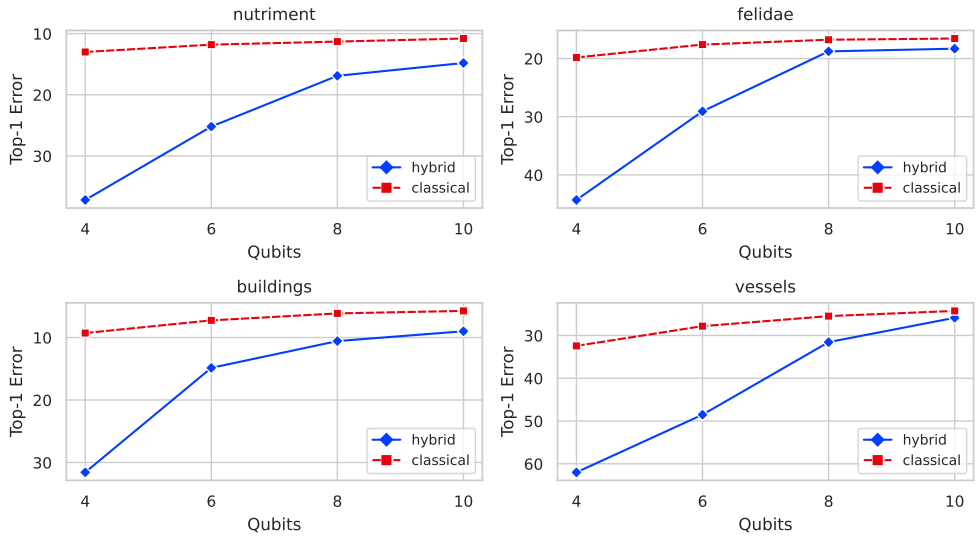


Figure 7.9: Hybrid QNN without Skip Connection

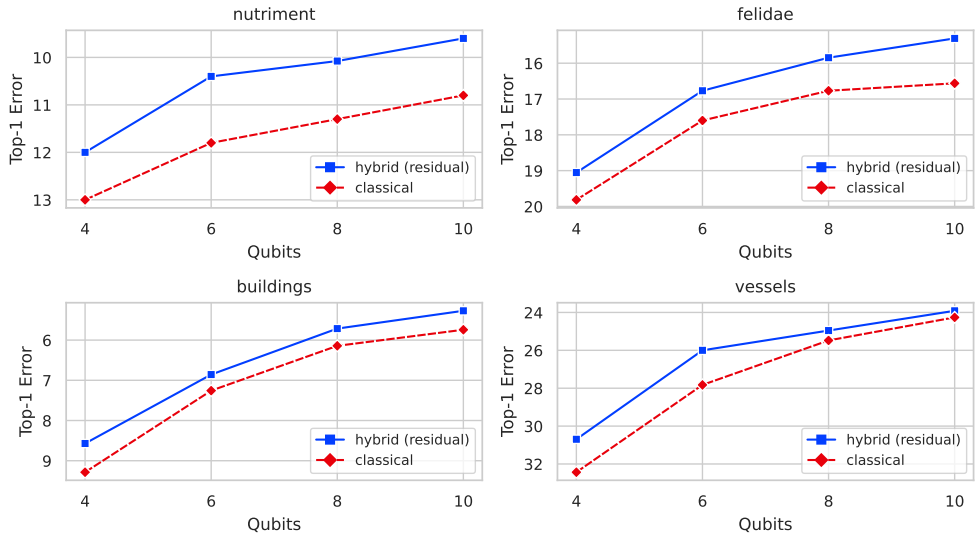


Figure 7.10: Hybrid QNN with Skip Connection

more sophisticated approaches for mapping low-dimensional qubits to a high-dimensional feature space [PCH+21]. Additionally, once training large QNN extractors is feasible, it would be interesting to determine whether we can train the classical compression model to find suitable representations for QNN embeddings.

Overview of Generative AI Tools Used

Grammarly[1] und LanguageTool[2] wurden als Grammatik- und Rechtschreibprüfprogramme verwendet. Perplexity[3] und Semantic Scholar[4] wurden zur Ergänzung der Literaturrecherche neben herkömmlichen Suchmaschinen eingesetzt.

[1] [https://app.grammarly.com/
[2] https://languagetool.org/
[3] https://www.perplexity.ai/
[4] https://www.semanticscholar.org

Übersicht verwendeter Hilfsmittel

Grammarly[5] and LanguageTool[6] were used as grammar and spellcheckers. Perplexity[7] and Semantic Scholar[8] were used to supplement the literature review with traditional search engines.

[5] https://app.grammarly.com/
[6] https://languagetool.org/
[7] https://www.perplexity.ai/
[8] https://www.semanticscholar.org

List of Figures

List of Tables

List of Algorithms

Bibliography

[AA20] Jordan Ash and Ryan P Adams. On warm-starting neural network training. *Advances in neural information processing systems*, 33:3884–3894, 2020.

[ACH+13] Pankaj K. Agarwal, Graham Cormode, Zengfeng Huang, Jeff M. Phillips, Zhewei Wei, and Ke Yi. Mergeable summaries. *ACM Trans. Database Syst.*, 38(4), December 2013.

[AFDM16] Alexander A. Alemi, Ian Fischer, Joshua V. Dillon, and Kevin Murphy. Deep variational information bottleneck. *CoRR*, abs/1612.00410, 2016.

[ALV+22] Mario Almeida, Stefanos Laskaridis, Stylianos I Venieris, Ilias Leontiadis, and Nicholas D Lane. Dyno: Dynamic onloading of deep neural networks from cloud to device. *ACM Transactions on Embedded Computing Systems*, 21(6):1–24, 2022.

[AM18] Naveed Akhtar and Ajmal S. Mian. Threat of adversarial attacks on deep learning in computer vision: A survey. *IEEE Access*, 6:14410–14430, 2018.

[AMT+17] Eirikur Agustsson, Fabian Mentzer, Michael Tschannen, Lukas Cavigelli, Radu Timofte, Luca Benini, and Luc Van Gool. Soft-to-hard vector quantization for end-to-end learning compressible representations. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 1141–1151, Red Hook, NY, USA, 2017. Curran Associates Inc.

[ANK+25] Muhammad Awais, Muzammal Naseer, Salman Khan, Rao Muhammad Anwer, Hisham Cholakkal, Mubarak Shah, Ming-Hsuan Yang, and Fahad Shahbaz Khan. Foundation models defining a new era in vision: A survey and outlook. *IEEE Trans. Pattern Anal. Mach. Intell.*, 47(4):2245–2264, 2025.

[ATC+21] Mohammad S Aslanpour, Adel N Toosi, Claudio Cicconetti, Bahman Javadi, Peter Sbarski, Davide Taibi, Marcos Assuncao, Sukhpal Singh Gill, Raj Gaire, and Schahram Dustdar. Serverless edge computing: vision and challenges. In *2021 Australasian Computer Science Week Multiconference*, pages 1–10, 2021.

[AYH+24] Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, CK Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Michael Suo, Phil Tillet, Eikan Wang, Xiaodong Wang, William Wen, Shunting Zhang, Xu Zhao, Keren Zhou, Richard Zou, Ajit Mathews, Gregory Chanan, Peng Wu, and Soumith Chintala. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, April 2024.

[BBG+23] Martin Beisel, Johanna Barzen, Simon Garhofer, Frank Leymann, Felix Truger, Benjamin Weder, and Vladimir Yussupov. Quokka: A service ecosystem for workflow-based execution of variational quantum algorithms. In *Service-Oriented Computing–ICSOC 2022 Workshops: ASOCA, AI-PA, FMCIoT, WESOACS 2022, Sevilla, Spain, November 29–December 2, 2022 Proceedings*, pages 369–373. Springer, 2023.

[BBL+23] Marvin Bechtold, Johanna Barzen, Frank Leymann, Alexander Mandl, Julian Obst, Felix Truger, and Benjamin Weder. Investigating the effect of circuit cutting in QAOA for the MaxCut problem on NISQ devices. *Quantum Science and Technology*, 8(4):045022, September 2023.

[BCM+20] Johannes Ballé, Philip A Chou, David Minnen, Saurabh Singh, Nick Johnston, Eirikur Agustsson, Sung Jin Hwang, and George Toderici. Nonlinear transform coding. *IEEE Journal of Selected Topics in Signal Processing*, 15(2):339–353, 2020.

[BF20] Dmytro Bondarenko and Polina Feldmann. Quantum autoencoders to denoise quantum data. *Physical review letters*, 124(13):130502, 2020.

[BGVG14] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101–mining discriminative components with random forests. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VI 13*, pages 446–461. Springer, 2014.

[BIS+18] Ville Bergholm, Josh Izaac, Maria Schuld, Christian Gogolin, Shahnawaz Ahmed, Vishnu Ajith, M Sohaib Alam, Guillermo Alonso-Linaje, B Akash-Narayanan, Ali Asadi, et al. Pennylane: Automatic differentiation of hybrid quantum-classical computations. *arXiv preprint arXiv:1811.04968*, 2018.

[BKR96] Rainer E. Burkard, Bettina Klinz, and Rüdiger Rudolf. Perspectives of monge properties in optimization. *Discrete Applied Mathematics*, 70(2):95–161, 1996.

[BLRPM19] Axel Barroso-Laguna, Edgar Riba, Daniel Ponsa, and Krystian Mikolajczyk. Key.Net: Keypoint Detection by Handcrafted and Learned CNN Filters. In *Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision*, 2019.

[BLS17] Johannes Ballé, Valero Laparra, and Eero P. Simoncelli. End-to-end optimized image compression. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[BM18] Yochai Blau and Tomer Michaeli. The perception-distortion tradeoff. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6228–6237, 2018.

[BM19] Yochai Blau and Tomer Michaeli. Rethinking lossy compression: The rate-distortion-perception tradeoff. In *International Conference on Machine Learning*, pages 675–685. PMLR, 2019.

[BMS+18] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. In *International Conference on Learning Representations*, 2018.

[BMZ+23] Arian Bakhtiarnia, Nemanja Milošević, Qi Zhang, Dragana Bajović, and Alexandros Iosifidis. Dynamic split computing for efficient deep edge intelligence. In *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5, 2023.

[BPS23] Lukas Brenner, Christophe Piveteau, and David Sutter. Optimal wire cutting with classical communication. *arXiv preprint arXiv:2302.03366*, February 2023.

[BRFP20] Jean Bégaint, Fabien Racapé, Simon Feltman, and Akshay Pushparaja. Compressai: a pytorch library and evaluation platform for end-to-end compression research. *arXiv preprint arXiv:2011.03029*, 2020.

[BS23] Tolga Bakirman and Elif Sertel. A benchmark dataset for deep learning-based airplane detection: Hrplanes. *International Journal of Engineering and Geosciences*, 8(3):212–223, 2023.

[BVR19] Reza Bahmanyar, Elenora Vig, and Peter Reinartz. Mrcnet: Crowd counting and density map estimation in aerial and ground imagery. *arXiv preprint arXiv:1909.12743*, 2019.

[CAB+21] Marco Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, et al. Variational quantum algorithms. *Nature Reviews Physics*, 3(9):625–644, 2021.

[CBFAB94] P. Charbonnier, L. Blanc-Feraud, G. Aubert, and M. Barlaud. Two deterministic half-quadratic regularization algorithms for computed imaging. In *Proceedings of 1st International Conference on Image Processing*, volume 2, pages 168–172 vol.2, 1994.

[CCA+21] Simon Cantrell, Jon Christopherson, Cody Anderson, Gregory L Stensaas, Shankar N Ramaseri Chandra, Minsu Kim, and Seonkyung Park. System characterization report on the worldview-3 imager. Technical report, US Geological Survey, 2021.

[CHJ+22] Justin Carnahan, Amy Hutputanasin, Alicia Johnstone, Wenschel Lan, Simon Lee, Arash Mehrpavar, Riki Munakata, David Pignatelli, and Armen Toorian. Cubesat design specification rev. 14.1. Technical Report 141, California Polytechnic State University, San Luis Obispo, CA, USA, February 2022.

[Cis20] Cisco. Cisco annual internet report (2018–2023). White Paper, March 2020. Accessed 29 July 2025.

[CK18] Yann Collet and Murray Kucherawy. Zstandard Compression and the application/zstd Media Type. RFC 8478, October 2018.

[CK19] Jaeho Choi and Joongheon Kim. A tutorial on quantum approximate optimization algorithm (qaoa): Fundamentals and applications. In *2019 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 138–142. IEEE, 2019.

[CKL+21] Graham Cormode, Zohar Karnin, Edo Liberty, Justin Thaler, and Pavel Veselý. Relative error streaming quantiles. In *Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS'21, page 96–108, New York, NY, USA, 2021. Association for Computing Machinery.

[Cla23] Class. Airbus aircraft detection dataset. `https://universe.roboflow.com/class-dvpyb/airbus-aircraft-detection`, jan 2023. visited on 2024-01-31.

[CLM+21] Tong Chen, Haojie Liu, Zhan Ma, Qiu Shen, Xun Cao, and Yao Wang. End-to-end learnt image compression via non-local attention optimization and improved context modeling. *IEEE Transactions on Image Processing*, 30:3179–3191, 2021.

[CMJ+18] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. Tvm: an automated end-to-end optimizing compiler for deep learning. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, OSDI'18, page 579–594, USA, 2018. USENIX Association.

[Com25] European Commission. Rolling plan for ICT standardization. `https://interoperable-europe.ec.europa.eu/collection/rolling-plan-ict-standardisation`, April 2025.

[CS25] Ziling Chen and Shaoxu Song. Randomized sketches for quantile in lsm-tree based store. *Proc. ACM Manag. Data*, 3(1), February 2025.

[CSM+20] Daniel Crankshaw, Gur-Eyal Sela, Xiangxi Mo, Corey Zumar, Ion Stoica, Joseph Gonzalez, and Alexey Tumanov. Inferline: latency-aware provisioning and scaling for prediction serving pipelines. In *Proceedings of the 11th ACM Symposium on Cloud Computing*, SoCC '20, page 477–491, New York, NY, USA, 2020. Association for Computing Machinery.

[CSTK20] Zhengxue Cheng, Heming Sun, Masaru Takeuchi, and Jiro Katto. Learned image compression with discretized gaussian mixture likelihoods and attention modules. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7939–7948, 2020.

[CSZ+18] Pin-Yu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. EAD: Elastic-net attacks to deep neural networks via adversarial examples. In *Proc. AAAI*, 2018.

[CV20] Graham Cormode and Pavel Veselý. A tight lower bound for comparison-based quantile summaries. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS'20, page 81–93, New York, NY, USA, 2020. Association for Computing Machinery.

[CVH+22] M Cerezo, Guillaume Verdon, Hsin-Yuan Huang, Lukasz Cincio, and Patrick J Coles. Challenges and opportunities in quantum machine learning. *Nature Computational Science*, 2(9):567–576, 2022.

[CW17] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *Proc. IEEE Symposium on Security and Privacy*, 2017.

[CY20] Graham Cormode and Ke Yi. *Small summaries for big data*. Cambridge University Press, 2020.

[CZS24] Hongrong Cheng, Miao Zhang, and Javen Qinfeng Shi. A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[DBB+12] C. Donlon, B. Berruti, A. Buongiorno, M.-H. Ferreira, P. Féménias, J. Frerick, P. Goryl, U. Klein, H. Laur, C. Mavrocordatos, J. Nieke, H. Rebhan, B. Seitz, J. Stroede, and R. Sciarra. The global monitoring for environment and security (gmes) sentinel-3 mission. *Remote Sensing of Environment*, 120:37–57, 2012. The Sentinel Missions - New Opportunities for Science.

[DBRUM21] Yann Dubois, Benjamin Bloem-Reddy, Karen Ullrich, and Chris J Maddison. Lossy compression for lossless prediction. *Advances in Neural Information Processing Systems*, 34:14014–14028, 2021.

[DCC+23] Bradley Denby, Krishna Chintalapudi, Ranveer Chandra, Brandon Lucia, and Shadi Noghabi. Kodan: Addressing the computational bottleneck in space. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 392–403, 2023.

[DDC+12] M. Drusch, U. Del Bello, S. Carlier, O. Colin, V. Fernandez, F. Gascon, B. Hoersch, C. Isola, P. Laberinti, P. Martimort, A. Meygret, F. Spoto, O. Sy, F. Marchese, and P. Bargellini. Sentinel-2: Esa's optical high-resolution mission for gmes operational services. *Remote Sensing of Environment*, 120:25–36, 2012. The Sentinel Missions - New Opportunities for Science.

[DKL+17] Kiruthika Devaraj, Ryan Kingsbury, Matt Ligon, Joseph Breu, Vivek Vittaldev, Bryan Klofas, Patrick Yeon, and Kyle Colton. Dove high speed downlink system. In *Small Satellite Conference*, 2017.

[DL19] Bradley Denby and Brandon Lucia. Orbital edge computing: Machine inference in space. *IEEE Computer Architecture Letters*, 18(1):59–62, March 2019.

[DL20] Bradley Denby and Brandon Lucia. Orbital edge computing: Nanosatellite constellations as a new class of computer system. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 939–954, New York, NY, USA, 2020. Association for Computing Machinery.

[DLH+20] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108(4):485–532, 2020.

[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.

[DTNQ19] Poulami Das, Swamit S Tannu, Prashant J Nair, and Moinuddin Qureshi. A case for multi-programming quantum computers. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 291–303, 2019.

[Dud14] Jarek Duda. Asymmetric numeral systems: entropy coding combining speed of huffman coding with compression rate of arithmetic coding. *arXiv preprint arXiv:1311.2540*, 2014.

[Dun21] Ted Dunning. The t-digest: Efficient estimates of distributions. *Software Impacts*, 7:100049, 2021.

[DXX+21] Jian Ding, Nan Xue, Gui-Song Xia, Xiang Bai, Wen Yang, Michael Yang, Serge Belongie, Jiebo Luo, Mihai Datcu, Marcello Pelillo, and Liangpei Zhang. Object detection in aerial images: A large-scale benchmark and challenges. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2021.

[DZF+20] Shuiguang Deng, Hailiang Zhao, Weijia Fang, Jianwei Yin, Schahram Dustdar, and Albert Y Zomaya. Edge intelligence: The confluence of edge computing and artificial intelligence. *IEEE Internet of Things Journal*, 7(8):7457–7469, 2020.

[EEP19] Amir Erfan Eshratifar, Amirhossein Esmaili, and Massoud Pedram. Bottlenet: A deep learning architecture for intelligent mobile cloud computing services. In *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6, 2019.

[EMW21] Daniel J Egger, Jakub Mareček, and Stefan Woerner. Warm-starting quantum optimization. *Quantum*, 5:479, 2021.

[Eri25] Ericsson. Ericsson mobility report. `https://ericsson.com/mobility-report`, June 2025.

[ERO21] Patrick Esser, Robin Rombach, and Björn Ommer. Taming transformers for high-resolution image synthesis. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pages 12873–12883. Computer Vision Foundation / IEEE, 2021.

[FBB+23] Alireza Furutanpey, Johanna Barzen, Marvin Bechtold, Schahram Dustdar, Frank Leymann, Philipp Raith, and Felix Truger. Architectural vision for quantum computing in the edge-cloud continuum. In *2023 IEEE International Conference on Quantum Software (QSW)*, pages 88–103, 2023.

[FD22] Alireza Furutanpey and Schahram Dustdar. Adaptive and collaborative inference: Towards a no-compromise framework for distributed intelligent systems. In *Proceedings of the 18th International Conference on Web Information Systems and Technologies (WEBIST)*, pages 144–151, 2022.

[FFSD25] Alireza Furutanpey, Pantelis A Frangoudis, Patrik Szabo, and Schahram Dustdar. Adversarial robustness of bottleneck injected deep neural networks for task-oriented communication. In *Proc. IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN)*, 2025.

[FHZ+22] Chuan Feng, Pengchao Han, Xu Zhang, Bowen Yang, Yejun Liu, and Lei Guo. Computation offloading in mobile edge computing networks: A survey. *Journal of Network and Computer Applications*, page 103366, 2022.

[FLW+25] Zhengru Fang, Zhenghao Liu, Jingjing Wang, Senkang Hu, Yu Guo, Yiqin Deng, and Yuguang Fang. Task-oriented communications for visual navigation with edge-aerial collaboration in low altitude economy. *arXiv preprint arXiv:2504.18317*, 2025.

[FMD+20] Gianluca Furano, Gabriele Meoni, Aubrey Dunne, David Moloney, Veronique Ferlet-Cavrois, Antonis Tavoularis, Jonathan Byrne, Léonie Buckley, Mihalis Psarakis, Kay-Obbe Voss, et al. Towards the use of artificial intelligence on the edge in space systems: Challenges and opportunities. *IEEE Aerospace and Electronic Systems Magazine*, 35(12):44–56, 2020.

[Fra99] Rich Franzen. Kodak lossless true color image suite. `https://r0k.us/graphics/kodak/`, 1999. Accessed 27 July 2025.

[FRD24] Alireza Furutanpey, Philipp Raith, and Schahram Dustdar. Frankensplit: Efficient neural feature compression with shallow variational bottleneck injection for mobile edge computing. *IEEE Transactions on Mobile Computing*, 23(12):10770–10786, 2024.

[FWR+25] Alireza Furutanpey, Carmen Walser, Philipp Raith, Pantelis A Frangoudis, and Schahram Dustdar. Leveraging neural graph compilers in machine learning research for edge-cloud systems. *arXiv preprint arXiv:2504.20198*, 2025.

[FZR+25] Alireza Furutanpey, Qiyang Zhang, Philipp Raith, Tobias Pfandzelter, Shangguang Wang, and Schahram Dustdar. Fool: Addressing the downlink bottleneck in satellite computing with neural feature compression. *IEEE Transactions on Mobile Computing*, 24(8):6747–6764, 2025.

[GARV+21] Jose Garcia-Alonso, Javier Rojo, David Valencia, Enrique Moguel, Javier Berrocal, and Juan Manuel Murillo. Quantum software as a service through a quantum api gateway. *IEEE Internet Computing*, 26(1):34–41, 2021.

[Gc21] Jacob Gildenblat and contributors. Pytorch library for cam methods. https://github.com/jacobgil/pytorch-grad-cam, 2021.

[GCA+21] Michele Grossi, Luca Crippa, Antonello Aita, Giacomo Bartoli, Vito Sammarco, Eleonora Picca, N Said, Filippo Tramonto, and Federico Mattei. A serverless cloud integration for quantum computing. *arXiv preprint arXiv:2107.02007*, 2021.

[GDdG+20] Gianluca Giuffrida, Lorenzo Diana, Francesco de Gioia, Gionata Benelli, Gabriele Meoni, Massimiliano Donati, and Luca Fanucci. Cloudscout: A deep neural network for on-board cloud detection on hyperspectral images. *Remote Sensing*, 12(14):2205, 2020.

[GDT+18] Edward Gan, Jialin Ding, Kai Sheng Tai, Vatsal Sharan, and Peter Bailis. Moment-based quantile sketches for efficient high cardinality aggregation queries. *Proc. VLDB Endow.*, 11(11):1647–1660, July 2018.

[GEBM19] Harper R Grimsley, Sophia E Economou, Edwin Barnes, and Nicholas J Mayhall. An adaptive variational algorithm for exact molecular simulations on a quantum computer. *Nature communications*, 10(1):3007, 2019.

[GFM+22] Gianluca Giuffrida, Luca Fanucci, Gabriele Meoni, Matej Batič, Léonie Buckley, Aubrey Dunne, Chris van Dijk, Marco Esposito, John Hefele, Nathan Vercruyssen, Gianluca Furano, Massimiliano Pastena, and Josef Aschbacher. The Φ-sat-1 mission: The first on-board deep neural network demonstrator for satellite earth observation. *IEEE Transactions on Geoscience and Remote Sensing*, 60:1–14, 2022.

[GK01] Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. *SIGMOD Rec.*, 30(2):58–66, May 2001.

[GK16] Michael B. Greenwald and Sanjeev Khanna. *Quantiles and Equi-depth Histograms over Streams*, pages 45–86. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[GKM22] Akshay Gadre, Swarun Kumar, and Zachary Manchester. Low-latency imaging and inference from lora-enabled cubesats. *arXiv preprint arXiv:2206.10703*, 2022.

[GLL+21] Alexey Galda, Xiaoyuan Liu, Danylo Lykov, Yuri Alexeev, and Ilya Safro. Transferability of optimal qaoa parameters between random graphs. In *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 171–180. IEEE, 2021.

[Gmb] DSaxonQ GmbH. SaxonQ Technology qubits in diamond. https://saxonq.com/technology. (accessed: 05.05.2023).

[Goy01] V.K. Goyal. Theoretical foundations of transform coding. *IEEE Signal Processing Magazine*, 18(5):9–21, 2001.

[GSS15] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *Proc. ICLR*, 2015.

[GST+18] Zacharias Georgiou, Moysis Symeonides, Demetris Trihinas, George Pallis, and Marios D Dikaiakos. Streamsight: A query-driven framework for streaming analytics in edge computing. In *2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)*, pages 143–152. IEEE, 2018.

[GSW24] Meghal Gupta, Mihir Singhal, and Hongxun Wu. Optimal quantile estimation: beyond the comparison model. In *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1137–1158. IEEE, 2024.

[GSWY25] Elena Gribelyuk, Pachara Sawettamalya, Hongxun Wu, and Huacheng Yu. Near-optimal relative error streaming quantile estimation via elastic compactors. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3486–3529. SIAM, 2025.

[GXL+22] Meng-Hao Guo, Tian-Xing Xu, Jiang-Jiang Liu, Zheng-Ning Liu, Peng-Tao Jiang, Tai-Jiang Mu, Song-Hai Zhang, Ralph R Martin, Ming-Ming Cheng, and Shi-Min Hu. Attention mechanisms in computer vision: A survey. *Computational visual media*, 8(3):331–368, 2022.

[GYMT21] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129:1789–1819, 2021.

[GZR23] Zhen Gao, Jinhua Zhu, and Pedro Reviriego. Reliability evaluation and fault tolerant design for kll sketches. *IEEE Transactions on Emerging Topics in Computing*, 12(4):1002–1013, 2023.

[HAL+22] Ronny Hänsch, Jacob Arndt, Dalton Lunga, Matthew Gibb, Tyler Pedelose, Arnold Boedihardjo, Desiree Petrie, and Todd M Bacastow. Spacenet 8-the detection of flooded roads and buildings. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1472–1480, 2022.

[HCH+14] Kiryong Ha, Zhuo Chen, Wenlu Hu, Wolfgang Richter, Padmanabhan Pillai, and Mahadev Satyanarayanan. Towards wearable cognitive assistance. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '14, page 68–81, New York, NY, USA, 2014. Association for Computing Machinery.

[HP24] Chih-Kai Huang and Guillaume Pierre. Aggregate monitoring for geo-distributed kubernetes cluster federations. *IEEE Trans. Cloud Comput.*, 12(4):1449–1462, 2024.

[HVD15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[HWC+22] Kai Han, Yunhe Wang, Hanting Chen, Xinghao Chen, Jianyuan Guo, Zhenhua Liu, Yehui Tang, An Xiao, Chunjing Xu, Yixing Xu, et al. A survey on vision transformer. *IEEE transactions on pattern analysis and machine intelligence*, 45(1):87–110, 2022.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[ILL+22] Nikita Ivkin, Edo Liberty, Kevin Lang, Zohar Karnin, and Vladimir Braverman. Streaming quantiles algorithms with small space and update time. *Sensors*, 22(24):9612, 2022.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[JCQ23] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. YOLO by Ultralytics. `https://github.com/ultralytics/ultralytics`, January 2023.

[JJT+24] Purvish Jajal, Wenxin Jiang, Arav Tewari, Erik Kocinare, Joseph Woo, Anusha Sarraf, Yung-Hsiang Lu, George K. Thiruvathukal, and James C. Davis. Interoperability in deep learning: A user survey and failure analysis of onnx model converters. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2024, page 1466–1478, New York, NY, USA, 2024. Association for Computing Machinery.

[JKC+18] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

[JT21] Longlong Jing and Yingli Tian. Self-supervised visual feature learning with deep neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11):4037–4058, 2021.

[JYB+25] Johannes Jakubik, Felix Yang, Benedikt Blumenstiel, Erik Scheurer, Rocco Sedona, Stefano Maurogiovanni, Jente Bosmans, Nikolaos Dionelis, Valerio Marsocci, Niklas Kopp, et al. Terramind: Large-scale generative multimodality for earth observation. *arXiv preprint arXiv:2504.11171*, 2025.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KH+09] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[KHG+17] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News*, 45(1):615–629, 2017.

[Kim20] Hoki Kim. Torchattacks: A pytorch repository for adversarial attacks. *arXiv preprint arXiv:2010.01950*, 2020.

[KLL16] Zohar Karnin, Kevin Lang, and Edo Liberty. Optimal quantile approximation in streams. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 71–78, 2016.

[Lan17] Kevin J Lang. Back to the future: an even more nearly optimal cardinality estimation algorithm. *arXiv preprint arXiv:1708.06839*, 2017.

[LB20] Frank Leymann and Johanna Barzen. The bitter truth about gate-based quantum algorithms in the nisq era. *Quantum Science and Technology*, 5(4):044007, 2020.

[LBC+18] Lawrence Leung, Vincent Beukelaers, Simone Chesi, Hyosang Yoon, Daniel Walker, and Joshua Egbert. Adcs at scale: Calibrating and monitoring the dove constellation. In *Proceedings of the AIAA/USU Conference on Small Satellites*, 2018.

[LBF+20] Frank Leymann, Johanna Barzen, Michael Falkenthal, Daniel Vietz, Benjamin Weder, and Karoline Wild. Quantum in the cloud: Application potentials and research opportunities. In *Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER)*, pages 9–24, 2020.

[LCH+23] Anqi Lu, Yun Cheng, Youbing Hu, Zhiqiang Cao, Yongrui Chen, and Zhijun Li. Satellite-terrestrial collaborative object detection via task-inspired framework. *IEEE Internet of Things Journal*, 10(23):20528–20544, 2023.

[LCS+21] Jingyun Liang, Jiezhang Cao, Guolei Sun, Kai Zhang, Luc Van Gool, and Radu Timofte. Swinir: Image restoration using swin transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1833–1844, 2021.

[LGS+22] Ming Lu, Peiyao Guo, Huiqing Shi, Chuntong Cao, and Zhan Ma. Transformer-based image compression. In *2022 Data Compression Conference (DCC)*, pages 469–469, 2022.

[LHC+24] Min Li, Zihao Huang, Lin Chen, Junxing Ren, Miao Jiang, Fengfa Li, Jitao Fu, and Chenghua Gao. Contemporary advances in neural network quantization: A survey. In *2024 International Joint Conference on Neural Networks (IJCNN)*, pages 1–10, 2024.

[LHJ+18] Hongshan Li, Chenghao Hu, Jingyan Jiang, Zhi Wang, Yonggang Wen, and Wenwu Zhu. Jalad: Joint accuracy-and latency-aware deep structure decoupling for edge-cloud execution. In *2018 IEEE 24th international conference on parallel and distributed systems (ICPADS)*, pages 671–678. IEEE, 2018.

[LHT+09] Simon Lee, A Hutputanasin, A Toorian, W Lan, R Munakata, J Carnahan, D Pignatelli, et al. Cubesat design specification rev. 13. *California Polytechnic State University, San Luis Obispo, USA*, 2009.

[LKM+18] Darius Lam, Richard Kuzma, Kevin McGee, Samuel Dooley, Michael Laielli, Matthew Klaric, Yaroslav Bulatov, and Brendan McCord. xview: Objects in context in overhead imagery. *arXiv preprint arXiv:1802.07856*, 2018.

[LL20] Wenbin Li and Matthieu Liewig. A survey of ai accelerators for edge environment. In *Trends and Innovations in Information Systems and Technologies: Volume 2 8*, pages 35–44. Springer, 2020.

[LLC+21] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.

[LLJL19] Li Lin, Xiaofei Liao, Hai Jin, and Peng Li. Computation offloading toward edge computing. *Proceedings of the IEEE*, 107(8):1584–1607, 2019.

[LLL+21] Mingzhen Li, Yi Liu, Xiaoyan Liu, Qingxiao Sun, Xin You, Hailong Yang, Zhongzhi Luan, Lin Gan, Guangwen Yang, and Depei Qian. The deep learning compiler: A comprehensive survey. *IEEE Transactions on Parallel and Distributed Systems*, 32(3):708–727, 2021.

[LLY+22] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE*

transactions on neural networks and learning systems, 33(12):6999–7019, 2022.

[LMW+22] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11976–11986, 2022.

[Ltd23] Quantum Brilliance Pty Ltd. Quantum Brilliance Hardware technology applications. https://quantumbrilliance.com/quantum-technology-application, 2023. accessed: 05.05.2023.

[ŁTV25] Aleksander Łukasiewicz, Jakub Tětek, and Pavel Veselỳ. Splinesketch: Even more accurate quantiles with error guarantees. *arXiv preprint arXiv:2504.01206*, 2025.

[LTY+21] Xiyang Luo, Hossein Talebi, Feng Yang, Michael Elad, and Peyman Milanfar. The rate-distortion-accuracy tradeoff: JPEG case study. In *Proc. 31st Data Compression Conference (DCC 2021)*, page 354, 2021.

[LVA+20] Stefanos Laskaridis, Stylianos I Venieris, Mario Almeida, Ilias Leontiadis, and Nicholas D Lane. Spinn: synergistic progressive inference of neural networks over device and cloud. In *Proceedings of the 26th annual international conference on mobile computing and networking*, pages 1–15, 2020.

[LZLG22] Hongzhou Liu, Wenli Zheng, Li Li, and Minyi Guo. Loadpart: Load-aware dynamic partition of deep neural networks for edge offloading. In *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*, pages 481–491, 2022.

[Mac05] D.J.C. MacKay. Fountain codes. *IEE Proceedings - Communications*, 152:1062–1068, December 2005.

[Mat21] Yoshitomo Matsubara. torchdistill: A Modular, Configuration-Driven Framework for Knowledge Distillation. In *International Workshop on Reproducible Research in Pattern Recognition*, pages 24–44. Springer, 2021.

[MBC+19] Yoshitomo Matsubara, Sabur Baidya, Davide Callegaro, Marco Levorato, and Sameer Singh. Distilled split deep neural networks for edge-assisted real-time systems. In *Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges*, HotEdgeVideo'19, page 21–26, New York, NY, USA, 2019. Association for Computing Machinery.

[MBI+20] Andrea Mari, Thomas R Bromley, Josh Izaac, Maria Schuld, and Nathan Killoran. Transfer learning in hybrid classical-quantum neural networks. *Quantum*, 4:340, 2020.

[MBT18] David Minnen, Johannes Ballé, and George D Toderici. Joint autoregressive and hierarchical priors for learned image compression. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[MCJ+24] Gaocheng Ma, Yinfeng Chai, Tianhao Jiang, Ming Lu, and Tong Chen. Tinylic-high efficiency lossy image compression method. *arXiv preprint arXiv:2402.11164*, 2024.

[MEHW18] Johannes Manner, Martin Endreß, Tobias Heckel, and Guido Wirtz. Cold start influencing factors in function as a service. In *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, pages 181–188. IEEE, 2018.

[MF21] Kosuke Mitarai and Keisuke Fujii. Constructing a virtual two-qubit gate by sampling single-qubit operations. *New Journal of Physics*, 23(2):023021, 2 2021.

[Mit20] Mina Mitry. Routers in space: Kepler communications' cubesats will create an internet for other satellites. *IEEE Spectrum*, 57(2):38–43, 2020.

[MLR22] Yoshitomo Matsubara, Marco Levorato, and Francesco Restuccia. Split computing and early exiting for deep learning applications: Survey and research challenges. *ACM Comput. Surv.*, 55(5), dec 2022.

[MRL19] Charles Masson, Jee E. Rim, and Homin K. Lee. Ddsketch: a fast and fully-mergeable quantile sketch with relative-error guarantees. *Proc. VLDB Endow.*, 12(12):2195–2205, August 2019.

[MVH+25] Arsham Mostaani, Thang X. Vu, Hamed Habibi, Symeon Chatzinotas, and Björn Ottersten. Task-oriented communication design at scale. *IEEE Transactions on Communications*, 73(1):378–393, 2025.

[MYLM22] Yoshitomo Matsubara, Ruihan Yang, Marco Levorato, and Stephan Mandt. Supervised compression for resource-constrained edge computing systems. In *Proc. IEEE/CVF Winter Conference on Applications of Computer Vision*, 2022.

[MYLM23] Yoshitomo Matsubara, Ruihan Yang, Marco Levorato, and Stephan Mandt. SC2 benchmark: Supervised compression for split computing. *Transactions on Machine Learning Research*, 2023.

[Nat24] National Aeronautics and Space Administration. Landsat-8 / ldcm (landsat data continuity mission). `https://www.eoportal.org/satellite-missions/landsat-8-ldcm#eop-quick-facts-section`, 2024. Accessed: 28 July 2025.

[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):39–50, 2020.

[NRF+22] Stefan Nastic, Philipp Raith, Alireza Furutanpey, Thomas Pusztai, and Schahram Dustdar. A serverless computing fabric for edge & cloud. In *2022 IEEE 4th International Conference on Cognitive Machine Intelligence (CogMI)*, pages 1–12. IEEE, 2022.

[NUB24] Hoa T Nguyen, Muhammad Usman, and Rajkumar Buyya. Qfaas: A serverless function-as-a-service framework for quantum computing. *Future Gener. Comput. Syst.*, 154:281–300, 2024.

[NZ08] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pages 722–729, 2008.

[OMK20] Daniel W Otter, Julian R Medina, and Jugal K Kalita. A survey of the usages of deep learning for natural language processing. *IEEE transactions on neural networks and learning systems*, 32(2):604–624, 2020.

[OSVM22] Yasuhiro Ohkura, Takahiko Satoh, and Rodney Van Meter. Simultaneous execution of quantum circuits on current and near-future nisq systems. *IEEE Transactions on Quantum Engineering*, 3:1–10, 2022.

[PCH+21] Evan Peters, João Caldeira, Alan Ho, Stefan Leichenauer, Masoud Mohseni, Hartmut Neven, Panagiotis Spentzouris, Doug Strain, and Gabriel N Perdue. Machine learning of high dimensional data on a noisy quantum processor. *npj Quantum Information*, 7(1):161, 2021.

[PHOW19] Tianyi Peng, Aram Harrow, Maris Ozols, and Xiaodi Wu. Simulating large quantum circuits on a small quantum computer. *Physical Review Letters*, 125:150504, 3 2019.

[PMJ+15] Nicolas Papernot, Patrick Mcdaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Proc. IEEE EuroS&P*, 2015.

[PW21] Seth Pettie and Dingyu Wang. Information theoretic limits of cardinality estimation: Fisher meets shannon. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 556–569, 2021.

[PXL+24] Jincheng Peng, Huanlai Xing, Yang Li, Li Feng, Lexi Xu, and Xianfu Lei. Task-oriented multi-user semantic communication with lightweight semantic encoder and fast training for resource-constrained terminal devices. *IEEE Wireless Communications Letters*, 13(9):2427–2431, 2024.

[QBW23] Nils Quetschlich, Lukas Burgholzer, and Robert Wille. Predicting good quantum circuit compilation options. In *Proc. IEEE International Conference on Quantum Software (QSW)*, pages 43–53, 2023.

[Rau21] Thomas Rausch. *A distributed compute fabric for edge intelligence*. PhD thesis, Technische Universität Wien, 2021.

[RBK+15] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. In *Proc. 3rd International Conference on Learning Representations (ICLR)*, 2015.

[RCS+21] Maryam Rahnemoonfar, Tashnim Chowdhury, Argho Sarkar, Debvrat Varshney, Masoud Yari, and Robin Roberson Murphy. Floodnet: A high resolution aerial imagery dataset for post flood scene understanding. *IEEE Access*, 9:89644–89654, 2021.

[RD21] Philipp Raith and Schahram Dustdar. Edge intelligence as a service. In *2021 IEEE International Conference on Services Computing (SCC)*, pages 252–262, 2021.

[RDS+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[RHS+21a] Thomas Rausch, Waldemar Hummer, Christian Stippel, Silvio Vasiljevic, Carmine Elvezio, Schahram Dustdar, and Katharina Krösl. Towards a platform for smart city-scale cognitive assistance applications. In *2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pages 330–335. IEEE, 2021.

[RHS+21b] Thomas Rausch, Waldemar Hummer, Christian Stippel, Silvio Vasiljevic, Carmine Elvezio, Schahram Dustdar, and Katharina Krösl. Towards a platform for smart city-scale cognitive assistance applications. In *2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pages 330–335, 2021.

[RLF+20] Thomas Rausch, Clemens Lachner, Pantelis A. Frangoudis, Philipp Raith, and Schahram Dustdar. Synthesizing plausible infrastructure configurations for evaluating edge computing systems. In *3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20)*. USENIX Association, June 2020.

[RLYK21] Francisco Romero, Qian Li, Neeraja J Yadwadkar, and Christos Kozyrakis. Infaas: Automated model-less inference serving. In *USENIX Annual Technical Conference*, pages 397–411, 2021.

[RMJ+22] Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. Ai and ml accelerator survey and trends. In *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–10. IEEE, 2022.

[RRD+22] Philipp Raith, Thomas Rausch, Schahram Dustdar, Fabiana Rossi, Valeria Cardellini, and Rajiv Ranjan. Mobility-aware serverless function adaptations across the edge-cloud continuum. In *2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)*, pages 123–132, 2022.

[RRFD23] Philipp Raith, Thomas Rausch, Alireza Furutanpey, and Schahram Dustdar. faas-sim: A trace-driven simulation framework for serverless edge computing platforms. *Software: Practice and Experience*, 53(12):2327–2361, 2023.

[RRP+22] Philipp Raith, Thomas Rausch, Paul Prüller, Alireza Furutanpey, and Schahram Dustdar. An end-to-end framework for benchmarking edge-cloud cluster management techniques. In *2022 IEEE International Conference on Cloud Engineering (IC2E)*, pages 22–28, 2022.

[SA19] Ruslan Shaydulin and Yuri Alexeev. Evaluating quantum approximate optimization algorithm: A case study. In *2019 tenth international green and sustainable computing conference (IGSC)*, pages 1–6. IEEE, 2019.

[SAEHJ+20] Saurabh Singh, Sami Abu-El-Haija, Nick Johnston, Johannes Ballé, Abhinav Shrivastava, and George Toderici. End-to-end learning of compressible features. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 3349–3353, 2020.

[Sat17] Mahadev Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, 2017.

[SBAS04] Nisheeth Shrivastava, Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri. Medians and beyond: new aggregation techniques for sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 239–249, 2004.

[SBB+20] Marie Salm, Johanna Barzen, Uwe Breitenbücher, Frank Leymann, Benjamin Weder, and Karoline Wild. The NISQ analyzer: automating the selection of quantum computers for quantum algorithms. In *Service-Oriented Computing: 14th Symposium and Summer School on Service-Oriented Computing, SummerSOC 2020, Crete, Greece, September 13-19, 2020 14*, pages 66–85. Springer, 2020.

[SCD+19] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-CAM: Visual explanations

from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128(2):336–359, oct 2019.

[SDBR15] J Springenberg, Alexey Dosovitskiy, Thomas Brox, and M Riedmiller. Striving for simplicity: The all convolutional net. In *ICLR (workshop track)*, 2015.

[Sha59] Claude E Shannon. Coding theorems for a discrete source with a fidelity criterion. In *IRE National Convention Record, 1959*, volume 4, pages 142–163, 1959.

[SHB+20] Jacob Shermeyer, Daniel Hogan, Jason Brown, Adam Van Etten, Nicholas Weir, Fabio Pacifici, Ronny Hansch, Alexei Bastidas, Scott Soenen, Todd Bacastow, et al. Spacenet 6: Multi-sensor all weather mapping dataset. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 196–197, 2020.

[SHVE+21] Jacob Shermeyer, Thomas Hossler, Adam Van Etten, Daniel Hogan, Ryan Lewis, and Daeil Kim. Rareplanes: Synthetic data takes flight. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 207–217, 2021.

[SICM23] Md. Maruf Hossain Shuvo, Syed Kamrul Islam, Jianlin Cheng, and Bashir I. Morshed. Efficient acceleration of deep learning inference on resource-constrained edge devices: A review. *Proceedings of the IEEE*, 111(1):42–91, 2023.

[Son] Kenny Song. Adversarial.js. https://kennysong.github.io/adversarial.js/. Accessed: 2024-07-29.

[SRS20] Rajendra P. Sishodia, Ram L. Ray, and Sudhir K. Singh. Applications of remote sensing in precision agriculture: A review. *Remote Sensing*, 12(19), 2020.

[SSTM21] Marion Sbai, Muhamad Risqi U. Saputra, Niki Trigoni, and Andrew Markham. Cut, distil and encode (cde): Split cloud-edge deep inference. In *2021 18th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pages 1–9, 2021.

[SZ20] Jiawei Shao and Jun Zhang. Bottlenet++: An end-to-end approach for feature compression in device-edge co-inference systems. In *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6, 2020.

[SZS+14] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *Proc. ICLR*, 2014.

[SZT17] Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*, 2017.

[TBB+24] Felix Truger, Johanna Barzen, Marvin Bechtold, Martin Beisel, Frank Leymann, Alexander Mandl, and Vladimir Yussupov. Warm-starting and quantum computing: A systematic mapping study. *ACM Comput. Surv.*, 56(9), April 2024.

[TCC+22] Jules Tilly, Hongxiang Chen, Shuxiang Cao, Dario Picozzi, Kanav Setia, Ying Li, Edward Grant, Leonard Wossnig, Ivan Rungger, George H Booth, et al. The variational quantum eigensolver: a review of methods and best practices. *Physics Reports*, 986:1–128, 2022.

[TCD+21] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herve Jegou. Training data-efficient image transformers & distillation through attention. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10347–10357. PMLR, 18–24 Jul 2021.

[TD22] A.C. Teodoro and L. Duarte. Chapter 10 - the role of satellite remote sensing in natural disaster management. In Adil Denizli, Marcelo S. Alencar, Tuan Anh Nguyen, and David E. Motaung, editors, *Nanotechnology-Based Smart Remote Sensing Networks for Disaster Prevention*, Micro and Nano Technologies, pages 189–216. Elsevier, 2022.

[The24] Lucas Theis. Position: What makes an image realistic? In *Forty-first International Conference on Machine Learning*, 2024.

[TL19] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[TMGV23] Bill Tao, Maleeha Masood, Indranil Gupta, and Deepak Vasisht. Transmitting, fast and slow: Scheduling satellite traffic through space and time. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, MobiCom '23, page October, New York, NY, USA, 2023. Association for Computing Machinery.

[TMN20] Damian A. Tamburri, Marco Miglierina, and Elisabetta Di Nitto. Cloud applications monitoring: An industrial study. *Information and Software Technology*, 127:106376, 2020.

[TMR20] Daniel Ting, Jonathan Malkin, and Lee Rhodes. Data sketching for real time analytics: Theory and practice. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3567–3568, 2020.

[Tow20] James Townsend. A tutorial on the range variant of asymmetric numeral systems. *arXiv preprint arXiv:2001.09186*, 2020.

[TPB00] Naftali Tishby, Fernando C. Pereira, and William Bialek. The information bottleneck method, 2000.

[TPK21] Ricardo Tubío-Pardavila and Naomi Kurahara. 18 - ground station networks. In Chantal Cappelletti, Simone Battistini, and Benjamin K. Malphrus, editors, *Cubesat Handbook*, pages 353–364. Academic Press, 2021.

[TST+20] George Toderici, Wenzhe Shi, Radu Timofte, Lucas Theis, Johannes Balle, Eirikur Agustsson, Nick Johnston, and Fabian Mentzer. Workshop and challenge on learned image compression (clic2020), 2020.

[TTV16] Pedro Tabacof, Julia Tavares, and Eduardo Valle. Adversarial images for variational autoencoders. *CoRR*, abs/1612.00155, 2016.

[TZ15] Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. In *2015 IEEE Information Theory Workshop (ITW)*, pages 1–5, 2015.

[VDDP18] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios D. Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. *Comput. Intell. Neurosci.*, 2018:7068349:1–7068349:13, 2018.

[VEH21] Adam Van Etten and Daniel Hogan. The spacenet multi-temporal urban development challenge. *arXiv preprint arXiv:2102.11958*, 2021.

[VELB18] Adam Van Etten, Dave Lindenbaum, and Todd M Bacastow. Spacenet: A remote sensing dataset and challenge series. *arXiv preprint arXiv:1807.01232*, 2018.

[VSC21] Deepak Vasisht, Jayanth Shenoy, and Ranveer Chandra. L2d2: Low latency distributed downlink for leo satellites. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 151–164, 2021.

[WBH+20] Karoline Wild, Uwe Breitenbücher, Lukas Harzenetter, Frank Leymann, Daniel Vietz, and Michael Zimmermann. Tosca4qc: two modeling styles for tosca to automate the deployment and orchestration of quantum applications. In *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)*, pages 125–134. IEEE, 2020.

[WBK+18] Michael Wurster, Uwe Breitenbücher, Kálmán Képes, Frank Leymann, and Vladimir Yussupov. Modeling and automated deployment of serverless applications using tosca. In *2018 IEEE 11th conference on service-oriented computing and applications (SOCA)*, pages 73–80. IEEE, 2018.

[WBL+21] Benjamin Weder, Johanna Barzen, Frank Leymann, Marie Salm, and Karoline Wild. Qprov: A provenance system for quantum computing. *IET Quantum Communication*, 2(4):171–181, 2021.

[WBLV21a] Manuela Weigold, Johanna Barzen, Frank Leymann, and Daniel Vietz. Patterns for hybrid quantum algorithms. In *Service-Oriented Computing: 15th Symposium and Summer School, SummerSOC 2021, Virtual Event, September 13–17, 2021, Proceedings 15*, pages 34–51. Springer, 2021.

[WBLV21b] Manuela Weigold, Johanna Barzen, Frank Leymann, and Daniel Vietz. Patterns for hybrid quantum algorithms. In *Service-Oriented Computing: 15th Symposium and Summer School, SummerSOC 2021, Virtual Event, September 13–17, 2021, Proceedings 15*, pages 34–51. Springer, 2021.

[WBLZ21] Benjamin Weder, Johanna Barzen, Frank Leymann, and Michael Zimmermann. Hybrid Quantum Applications Need Two Orchestrations in Superposition: A Software Architecture Perspective. In *Proceedings of the 18IEEE International Conference on Web Services (ICWS 2021)*, pages 1–13. IEEE, 2021.

[Wig19] Ross Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019.

[Wil88] Robert Wilber. The concave least-weight subsequence problem revisited. *J. Algorithms*, 9(3):418–425, September 1988.

[WL23] Shangguang Wang and Qing Li. Satellite computing: Vision and challenges. *IEEE Internet of Things Journal*, 10(24):22514–22529, 2023.

[WLX+23] Changhao Wu, Yuanchun Li, Mengwei Xu, Chongbin Guo, Zengshan Yin, Weiwei Gao, and Chuanxiu Xi. A comprehensive survey on orbital edge computing: Systems, applications, and algorithms. *arXiv preprint arXiv:2306.00275*, 2023.

[WLYC13] Lu Wang, Ge Luo, Ke Yi, and Graham Cormode. Quantiles over data streams: An experimental study. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 737–748, 2013.

[WM15] James Warren and Nathan Marz. *Big Data: Principles and best practices of scalable realtime data systems*. Simon and Schuster, 2015.

[WNC87] Ian H. Witten, Radford M. Neal, and John G. Cleary. Arithmetic coding for data compression. *Commun. ACM*, 30(6):520–540, June 1987.

[WTJ21] Ross Wightman, Hugo Touvron, and Hervé Jégou. Resnet strikes back: An improved training procedure in timm. *arXiv preprint arXiv:2110.00476*, 2021.

[WY22] Lin Wang and Kuk-Jin Yoon. Knowledge distillation and student-teacher learning for visual intelligence: A review and new outlooks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(6):3048–3068, 2022.

[XAD+22] Qin Xin, Mamoun Alazab, Vicente García Díaz, Carlos Enrique Montenegro-Marin, and Rubén González Crespo. A deep learning architecture for power management in smart cities. *Energy Reports*, 8:1568–1577, 2022.

[XFM+21] Mengwei Xu, Zhe Fu, Xiao Ma, Li Zhang, Yanan Li, Feng Qian, Shangguang Wang, Ke Li, Jingyu Yang, and Xuanzhe Liu. From cloud to edge: a first look at public edge platforms. In Dave Levin, Alan Mislove, Johanna Amann, and Matthew Luckie, editors, *IMC '21: ACM Internet Measurement Conference, Virtual Event, USA, November 2-4, 2021*, pages 37–53. ACM, 2021.

[XGD+17] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[XWW+19] Yu Xing, Jian Weng, Yushun Wang, Lingzhi Sui, Yi Shan, and Yu Wang. An in-depth comparison of compilers for deep neural networks on hardware. In *2019 IEEE International Conference on Embedded Software and Systems (ICESS)*, pages 1–8, 2019.

[YMT23] Yibo Yang, Stephan Mandt, and Lucas Theis. An introduction to neural data compression. *Found. Trends. Comput. Graph. Vis.*, 15(2):113–200, April 2023.

[ZBG18] Zhuoran Zhao, Kamyar Mirzazad Barijough, and Andreas Gerstlauer. Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2348–2359, 2018.

[ZC25] Hailin Zhong and Donglong Chen. High-efficiency split computing for cooperative edge systems: A novel compressed sensing bottleneck. *arXiv preprint arXiv:2504.15295*, 2025.

[ZCC+24] 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*, 23(5):5069–5082, 2024.

[ZIE+18] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*

(CVPR), pages 586–595, Los Alamitos, CA, USA, jun 2018. IEEE Computer Society.

[ZJS+25] Xiaoyu Zhang, Weipeng Jiang, Chao Shen, Qi Li, Qian Wang, Chenhao Lin, and Xiaohong Guan. Deep learning library testing: Definition, methods and challenges. *ACM Comput. Surv.*, 57(7), March 2025.

[ZK16] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *Proceedings of the British Machine Vision Conference 2016 (BMVC)*. BMVA Press, 2016.

[ZKL+16] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2921–2929, 2016.

[ZLC+22] Qiyang Zhang, Xiang Li, Xiangying Che, Xiao Ma, Ao Zhou, Mengwei Xu, Shangguang Wang, Yun Ma, and Xuanzhe Liu. A comprehensive benchmark of deep learning libraries on mobile devices. In *Proceedings of the ACM Web Conference 2022*, WWW '22, page 3298–3307, New York, NY, USA, 2022. Association for Computing Machinery.

[ZMW+21] Fuheng Zhao, Sujaya Maiyya, Ryan Wiener, Divyakant Agrawal, and Amr El Abbadi. Kll±approximate quantile sketches over dynamic datasets. *Proceedings of the VLDB Endowment*, 14(7):1215–1227, 2021.

[ZXWZ23] Hongbin Zhang, Mingjie Xing, Yanjun Wu, and Chen Zhao. Compiler technologies in deep learning co-design: A survey. *Intelligent Computing*, 2:0040, 2023.

[ZY22] Yuxiao Zhou and Kecheng Yang. Exploring tensorrt to improve real-time inference for deep learning. In *2022 IEEE 24th Int Conf on High Performance Computing & Communications; 8th Int Conf on Data Science & Systems; 20th Int Conf on Smart City; 8th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*, pages 2011–2018. IEEE, 2022.

[ZYX+24] Qiyang Zhang, Xin Yuan, Ruolin Xing, Yiran Zhang, Zimu Zheng, Xiao Ma, Mengwei Xu, Schahram Dustdar, and Shangguang Wang. Resource-efficient in-orbit detection of earth objects. In *Proc. IEEE INFOCOM*, pages 551–560, 2024.

[ZZC+22] Kongyange Zhao, Zhi Zhou, Xu Chen, Ruiting Zhou, Xiaoxi Zhang, Shuai Yu, and Di Wu. Edgeadaptor: Online configuration adaption, model selection and resource provisioning for edge dnn inference serving at scale. *IEEE Transactions on Mobile Computing*, pages 1–16, 2022.