A Privacy Enforcing Framework for Data Streams on the Edge

Boris Sedlak, Ilir Murturi, *Member, IEEE,* Praveen Kumar Donta, *Senior Member, IEEE,* Schahram Dustdar, *Fellow, IEEE*

Abstract—Recent developments in machine learning (ML) allow for efficient data stream processing and also help in meeting various privacy requirements. Traditionally, predefined privacy policies are enforced in resource-rich and homogeneous environments such as in the cloud to protect sensitive information from being exposed. However, large amounts of data streams generated from heterogeneous IoT devices often result in high computational costs, cause network latency, and increase the chance of data interruption as data travels away from the source. Therefore, this paper proposes a novel privacy-enforcing framework for transforming data streams by executing various privacy policies close to the data source. To achieve our proposed framework, we enable domain experts to specify high-level privacy policies in a human-readable form. Then, the edge-based runtime system analyzes data streams (i.e., generated from nearby IoT devices), interprets privacy policies (i.e., deployed on edge devices), and transforms data streams if privacy violations occur. Our proposed runtime mechanism uses a Deep Neural Networks (DNN) technique to detect privacy violations within the streamed data. Furthermore, we discuss the framework, processes of the approach, and the experiments carried out on a real-world testbed to validate its feasibility and applicability.

Index Terms—Edge Computing, Privacy Models, Data Stream Transformations, Data Anonymization

1 Introduction

In recent years, the number of Internet of Things (IoT) devices has significantly expanded in multiple applications such as smart cities, autonomous vehicles, smart factories, etc. These applications produce enormous volumes of data, including continuously streaming high-quality videos or images and sending them to a central cloud service for further analytics. Since these applications are time-critical, analytical results are required rapidly [1]. Nevertheless, the massive amount of data streams, heterogeneous devices, and networks involved causes high traffic, which in turn affects the overall latency [2].

One well-known strategy that has gained attention recently suggests using distributed computing devices (also known as edge devices) close to end-users at the edge of networks. Dedicated edge servers, network routers, telecommunications stations, or edge gateways make up edge networks [3]. Edge networks or edge infrastructures are characterized as heterogeneous, volatile, and dynamic environments. The computing devices within such infrastructures can be leveraged to process multiple data or workloads instead, and as such, the edge emerges as a central architectural entity. More specifically, utilizing edge devices provides several benefits and overcomes many challenges [4]. First, it decreases user-perceived latency and reduces the need to transfer data to the cloud. Second, it enhances privacy by analyzing released information by users without their consciousness or information that can violate privacy policy requirements defined by a stakeholder (e.g., company, school, etc.). In this context, edge devices are essential to support network affairs and improve privacy protection. To prevent the release of sensitive data (such as user information, sensory data streams, etc.), edge devices can operate as an intermediary entity to implement different privacy policies and protect the information that has already been exposed. For instance, a security camera may capture sensitive information from laptop screens in the smart factory and violate privacy requirements. Therefore, an edge-based mechanism aims to prevent sensitive information from being released and ensure that third parties vendors cannot identify individuals or discover sensitive content without their consent.

To overcome the above-mentioned challenges, we advocate that data streams must be transformed based on rule-based procedures to ensure the desired privacy level. Additionally, transformations should occur in a uniform runtime environment to eliminate the need to implement a policy multiple times for different edge devices. We refer to transformations as modifications of data or metadata - an operation that arbitrarily combines or discards information based on predefined privacy policies. More precisely, transformations derive from privacy policies which comprise a set of rules to transform data and ensure privacy. We assume that the closest edge device should first analyze data streams, transform data if privacy policies are violated, and release the data for further usage. For instance, a roaming mobile IoT device may capture personal or confidential data in a smart factory. Before leaving the local network, data streams can be first forwarded and transformed by an arbitrary device on the edge network. Any edge device in proximity may analyze and transform the data stream according to the predefined set of privacy policies. However, we still lack a human-readable representation for specifying

I. Murturi is the Corresponding author
Authors are with Distributed Systems Group, TU Wien, 1040, Austria
Email: boris.sedlak@tuwien.ac.at, {imurturi, pdonta, dustdar}@dsg.tuwien.ac.at

such a model, as well as a compiler and execution environment that enforces privacy policies.

In this paper, we extend and validate our novel privacy-enforcing framework [5] for transforming data streams according to predefined privacy policies close to the data source - at the edge. A central trusted entity manages edge devices and privacy policies for the entire data pipeline. Such a central entity is a trustable cloud platform that ensures exactly how and when privacy is enforced. As a result, anytime an IoT device records sensitive or private information, an edge gateway (also known as an edge device) nearby analyzes and adjusts the data stream in accordance with a predetermined set of rules. Such rules build up privacy models, representing an enterprise's privacy policies. Models consisting of trigger and transformation functions describe how data has to be modified before it can be released to stream subscribers. Our concrete contributions are as follows:

- We enable domain experts to specify high-level privacy policies in a human-readable format, expressed through chains of privacy-violating constraints and countermeasures to cope with data that does not comply with these policies. Furthermore, we developed an edge-based runtime system for controlling and interpreting privacy policies deployed on edge devices.
- We use Deep Neural Network (DNN) to detect privacy violations within the streamed data. Our approach is extendable with publicly available and self-trained DNN models to adapt to changing policies and support custom use cases.
- We perform an extensive evaluation via several experiments regarding the latency that our framework introduces to the network. The results underline the feasibility and applicability of the approach to run on edge and ensure privacy.

The remaining paper is structured as follows. Related work is considered in Section 2. Section 3 shows the motivating example used throughout the paper. In Section 4, we describe in detail the framework. The processes and details of the proposed architecture are discussed in Section 5. Evaluation and results are discussed in Section 6. Finally, Section 7 concludes the paper and outlines future work directions.

2 Related Work

One way to preserve the contributors' privacy is to anonymize the streaming data. In [6], the authors propose data collection schemes for IoT sensors that do not provide proof of the data source. By doing this, the relationship between an IoT sensor and the measured data is obscured, yet, it is still difficult to remove sensitive information from the data. In [7], the authors demonstrated an edge-based solution that removes private attributes from sensor information. Their research provides a machine learning model that determines which attributes to remove throughout the transformation process. However, their strategy does not allow more complex privacy settings beyond just removing single attributes. This is similar to the work of [8], which focuses on enforcing privacy policies on the data based on the edge device's context (e.g., proximity, role, network). The authors limited their work to role-based access schemes, where policies represented precise rules on which role can consume the data type. This approach seamlessly does not require any moderating entity in the architecture but is very restricted to the type of privacy policies it can represent.

A fundamental problem when enforcing differential privacy on streams is that data is non-stationary, meaning that structural attributes might evolve until privacy violations cannot be detected anymore. To that extent, the authors in [9] discuss monitoring stationary shifts in the data and adjusting the data release process accordingly. Their answer ensures k-anonymity on the resulting data, which is also the content of [10] where the authors proposed a novel approach to ensure privacy on data streams without any delay. A notable improvement for latency-aware systems that need to provide an applicable latency; however, they do not consider stationary shifts in their work. *Baniya et al.* [11] explore the use of privacy models to express role-based access control strategies. The authors implemented an edge-based system where data is consumed over a message broker after transformation. More generally, in [12], *Tsigkanos et al.* investigates privacy issues on edge, including stream processing and anonymization techniques that can be applied, e.g., like the aforementioned z-anonymity. The authors discuss how federated learning preserves privacy by keeping training data at the edge level, an idea that is pursued by [13] for deep learning image recognition. *Yi et al.* [14] present solution for transforming video streams on edge. The proposed solution enables scaling up to many transformation workers when the incoming stream latency decreases due to the increased computation on the edge device. Such methods are undoubtedly a beneficial extension for stream transformation scenarios for maintaining low and stable latency.

Several works address open challenges in modeling privacy requirements. The authors in [15] contribute in many regards toward the specification of privacy or security requirements. They investigated possibilities to enable the specification of privacy requirements through modeling languages. However, they remain to provide an underlying privacy-enforcing environment that supports the transformations imposed by the privacy models. Furthermore, the authors in [16] have discussed the efficiency of using edge computing environments for data collection in IoT systems. Protecting sensor data from undesired access is crucial to any modern system. Cao et al. [17] advertise edge computing facilities for latency-aware tasks. The authors give a broad outlook to fields that will benefit from processing IoT data at the edge without involving any central cloud server. However, they do not consider securing data according to privacy requirements. In [18], the authors show the advantages of edge computing for collecting geospatial data, which is related to numerous papers in the context of data collection [19].

The aforementioned research demonstrates that the edge has drawn significant interest for several purposes, including enforcing privacy policies. According to research literature, transformations on edge have only ever been used for specific and selected operations (e.g., image removal, etc.).

As previously indicated, numerous research studies define enterprise privacy requirements. These efforts, however, are more concerned with business operations than low-level issues like deleting private information from a continuous data stream. Therefore, we introduce a novel framework for transforming data streams on the edge.

3 Motivational Example

As a simple, motivating scenario for a privacy-enforcing system, consider a modern smart manufacturing environment containing various appliances and field technicians monitoring and operating them. Several privacy policies may be applied to protect the organization's legitimate interests in these environments. We consider a situation where field technicians aided by Augmented Reality (AR) applications running on mobile devices perform maintenance or repair tasks of appliances. Such an AR-based application uses a mobile device camera to discover and visualize which appliance is running correctly and which requires calibration. Once a defective machine is found, the data, including visual content is automatically shared with the corresponding appliance producer responsible for the maintenance or calibration of the machine. However, the visual content shared may contain various information which may violate the organization's privacy policies.

Confidential information might be leaked by somebody who gets hold of the live stream or a recording. Privacy breaches are prevented by either securing data appropriately or removing confident information entirely; thus, it is evident that the data should be transformed before it is released to third parties. In this sense, edge devices in proximity to end-users (e.g., field technicians) can transform such data streams by enforcing predefined privacy policies at design time.

4 The Framework

The proposed framework describes three aspects: (i) the structure of the privacy model, (ii) the execution of these models on data streams, and (iii) managing the structure of the edge networks. The last aspect concerns data transmission between IoT nodes and edge gateways, including how to consume the privacy-preserved output data.

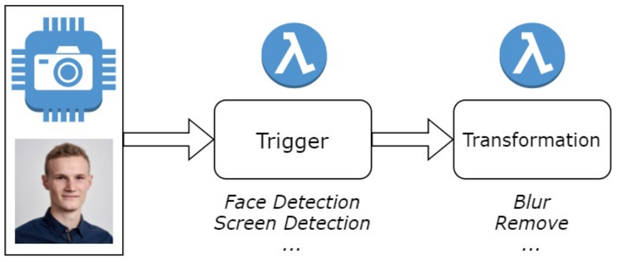


Fig. 1: General privacy model with triggers and transformations functions (λ).

The privacy model is represented using an acyclic graph in which each node and its relations are embedded with rules. Figure 1 illustrates how trigger and transformation functions can be used to secure IoT streams; these (cause-and-effect) rules can also be chained to apply only once the previous rule has been met. The grammar is validated using a model compiler on the edge before running the privacy model.

Static analysis of images containing a particular pattern is one example of a trigger and transformation that does not require a state. Stateless λ-functions can be chained together in the aforementioned acyclic graph, passing the results from one step to the next before returning to the stream subscribers. Some cases would require maintaining a state, like z-anonymity. The λ-functions can address such matters when temporary storage is available. In almost all cases, functions are combined through a UI in similar environments, such as AWS lambdas [20] and NVIDIA Deep Stream[1].

From a central cloud application, a model graph is defined and deployed to all edge gateways. Input data streams are analyzed and transformed based on the specified criteria once the model has been compiled. It is separated from the stream processing to keep lambda functions and the privacy model separate. When a new privacy model or λ-function is received, the gateway can continuously transform a data stream without resetting active connections. The λ-functions and privacy models are updated as lightheartedly as possible to maintain stable stream latency and not impact performance.

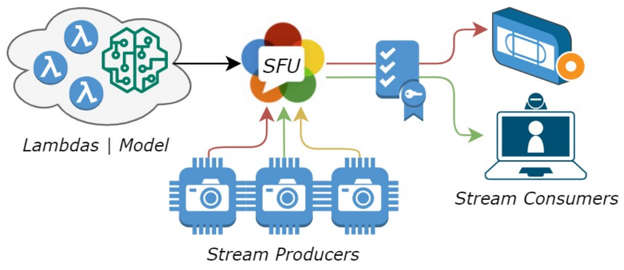


Fig. 2: The privacy-enforcing framework for transforming data streams on the edge.

In addition to receiving and transmitting data streams, edge gateways must decode stream data after performing transformations required by the privacy model and reencode it at the end of the transformation process. Streaming of videos and other data over the Internet using WebRTC[2] is a real-time communication protocol for the web. Nevertheless, an edge gateway can be extended to be a Selective Forwarding Unit (SFU) that receives and forwards data streams between multiple clients. Figure 2 shows all the major components of the privacy-enforcing framework. Those parts are summarized as follows:

1) A cloud application, which is used by a policy manager to define new λ-functions and privacy models. The application communicates with the edge gateway through an exposed REST endpoint for configuration.
2) There are multiple stream producers (such as IoT devices) that send data using the WebRTC protocol to the edge gateway.

1. DeepStream, https://developer.nvidia.com/deepstream-sdk
2. Real-time communication for the web, https://webrtc.org/

3) A variety of stream consumers are available, including consumer devices or recorders. WebRTC must be supported by all devices, regardless of their purpose.
4) The SFU is deployed at the edge gateway as the core of the topology. A REST endpoint receives new models and functions without interfering with the ongoing stream connection. Before streams are streamed to peers, incoming stream packages are decoded and transformed in compliance with a privacy model.

The SFU uses a dedicated Session Description Protocol (SDP) to establish connections between producer and a consumer based on session identifier, data type, etc. The connection between these peers is established once both the peers have agreed. WebRTC can provide multiple channels to communicate among peers: *media stream* for audio or video, and *data channel* for data encoded in text or byte arrays. In both media stream and data channel, the data transformation and analysis are always in terms of frames. Because of this, it can not detect the privacy violations that emerge in consecutive frames. Upon receiving frames from each channel, the SFU analyzes them and transforms them based on the privacy model.

The SFU environment has to provide the analysis according to the data arrived through input frames. For instance, when streaming video frames over the SFU, a suitable environment that supports video operation, such as OpenCV, will be used to analyze incoming frames. We can then use our privacy model to execute an algorithm that detects privacy-violating patterns in video frames (triggers). Once a face is detected in this frame, the frame is blurred (transformation) before being sent to consumer devices. In addition, the SFU can be extended with other environments in order to perform various analyses and transformations on data frames.

5 The Process

This section describes the process for transforming data streams on edge. We first explain the data flow from producer to consumer devices. Later we show how we detect privacy-violating patterns within frames and explain how to transform them based on policies. Lastly, we discuss privacy requirements specifications and how to enforce the model.

5.1 Data Provision and Consumption

The SFU running on an edge device manages peer connections to the producer- and consumer devices. Each new client must establish a peer connection through SDP. The SFU interacts with producers and consumers to create peer connections, as shown in Figure 3.

A peer connection is established after the producer or consumer receives the SDP response. At this point, producer and consumer can still choose what type of data to transfer over the stream - in our case, they can provide and consume video data from the SFU. After the track is added to the peer connection, video frames flow from the producer to the SFU. Afterward, they are routed to their destination, respectively, to the consumers. For consuming the data stream, we make use of a Media Relay[3], which ensures that each frame is transferred with the same frequency to a list of consumers.

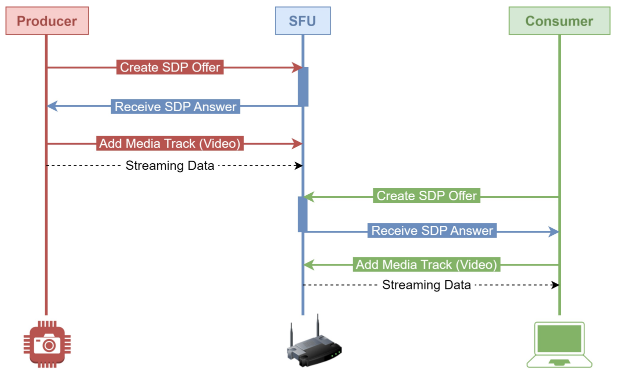


Fig. 3: Establishing peer connections between producer, SFU, and consumer.

Frames are thus relayed to consumers, i.e., whatever content the SFU provides, is copied and sent to each consumer. Contrary to the presented framework, our prototype is limited to process a single stream. This limitation should be addressed in future work.

5.2 Pattern Detection and Transformations

We leverage Convolutional Neural Networks (CNN) to find patterns in streamed data; however, we did not train any networks ourselves since this was not the main goal of our research. Instead, we apply existing models created with ML frameworks like PyTorch[4] or Tensorflow[5]. As a consequence, existing types of models by design lack interoperability between different environments. Each model requires its runtime environment to support its execution. Nevertheless, executing multiple runtime environments on edge devices is resource intensive and not feasible in terms of performance. Therefore, we suggest converting all models to the Open Neural Network eXchange (ONNX) format[6]. More specifically, we deploy the ONNX runtime on edge devices which will serve as the core runtime platform for pattern detection.

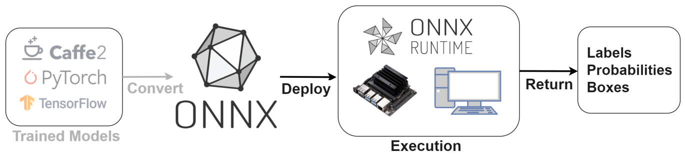


Fig. 4: ONNX model lifecycle from conversion to execution.

Figure 4 depicts the conversion of a PyTorch-trained model to the ONNX standard. The resulting model can then be executed with the ONNX runtime on edge devices like the NVIDIA Jetson or any conventional device that supports Python3. An essential feature of the ONNX runtime is the ability to facilitate GPU-accelerated image processing on edge devices through NVIDIA Cuda®[7]. After the execution process, the model yields a set of labels, probabilities, and

3. MediaRelay, https://aiortc.readthedocs.io/en/latest/helpers.html
4. PyTorch, https://www.pytorch.org
5. Tensorflow, https://www.tensorflow.org/?hl=es-419
6. ONNX, https://onnx.ai/
7. Cuda, https://developer.nvidia.com/cuda-toolkit

boxes, i.e., where boxes represent a detected pattern, a label (i.e., detected object), and the probability value.

5.2.1 Video Stream Function

Video frames that the SFU receives contain metadata such as image characteristics (i.e., width, height, and video codec) and a time base (i.e., which relates to the video frame rate). A typical video frame could have a resolution of 640x480 pixels and is encoded in yuv420p (i.e., an image codes, more commonly known as H.264). Depending on these specs, specific image recognition models might not even be applicable because they require a higher resolution to process the image. We can always resize video frames to a lower resolution, but we can not scale them up arbitrarily; in either case, a video frame has to conform to a precise format that the ONNX model can process.

TABLE 1: Applied ONNX models for pattern detection

Name	Description
Face Detection 320 [21]	Lightweight face detection for edge devices
Face Detection 640 [21]	As above, higher accuracy through resolution
Age Classification [22]	Returns age range (e.g., 25-32) and probability
Gender Clsf. [22], [23]	Returns gender (male/female) and probability

We wrap ONNX models in a uniform method structure and refer to them as *trigger functions*. In Table 1, we show the four trigger functions used in our implementation. Furthermore, we implemented three privacy-enforcing transformations based on the OpenCV Python package[8]. Table 2 contains all *transformation functions* together with a short description.

TABLE 2: Overview of OpenCV transformation functions

Name	Description
Blur_Area_Pixelate [24]	Blurs an area with a pixel grid of x*x rectangles
Fill_Area_Box	Replaces a frame area with a colored box
Max_Spec_Resize	Resizes a frame if it exceeds given boundaries

As mentioned previously, transformations are countermeasures to ensure privacy once a privacy violation is detected. To that extent, every frame routed through the SFU must be processed before relaying it to consumers. Our implementation can easily be extended with other ONNX models and OpenCV transformations; for instance, we might as well train a custom model and supply it to the framework. We provided an abstract interface that all trigger and transformation functions must implement, thus maintaining a uniform function structure. The presented transformation and trigger functions do not keep any state; they represent static operations on the input data that return labels, boxes, and probabilities, as shown in Figure 4. A limitation that emerges from this static nature is that we can only detect privacy violations that appear from a single data frame, i.e., our approach can not detect any movements or transitions between frames.

Although the SFU receives video frames from the producer with an overall stable frame rate, individual frames might arrive faster or slower due to minor network instabilities. To re-stabilize the stream's frame rate and relay a frame exactly every $1/fps$ seconds to connected consumers, we implemented a frame queue between the SFU and the video transformation track. Whenever the SFU receives video frames from a producer, the frames are appended to a thread-safe queue, where a second thread retrieves them for running trigger and transformation functions. After a consumer connects through the peer-to-peer protocol, the second thread consumes the enqueued frames and executes the configured trigger and transformation functions. The second thread operates through the queue with exactly the rate $1/fps$, and thus, reestablishes the original frame rate of the stream before frames are relayed to consumers. This aids in masking minor timing issues in the transmission of video frames between SFU and the producer.

8. Opencv-python: Wrapper package for OpenCV python bindings, https://github.com/skvark/opencv-python

5.3 Privacy Model Specification

To model privacy requirements, we advocate the principle of procedural abstraction: specifying several parameters of function internals may be impractical for developers/end-users, but considering the effects of function processes is feasible. For instance, one does not need to know how our Face_Trigger function detects faces in a video stream on a technical scale, but only by invoking our service face patterns and other information are obtained. Similarly, a specific transformation must be executed once face patterns are detected to hide the sensitive information. To this end, we advocate that trigger functions and transformation functions can chain together to form a privacy-enforcing model.

A privacy enforcing model is a set of chains of functions given in a textual representation, where individual chain links represent the trigger and transformation functions defined earlier. Chain links are connected with one-directional arrows ("→") and could as well be described as linked lists. We argue that this follows the logical order of execution and is well applicable to developers/end-users who are unaware of technical implementations hidden behind the function names. For instance, a privacy enforcing model for blurring human faces on a surveillance camera can be represented as in the given model (see Listing 1):

$$\begin{aligned} video: \{'tag':'webcam'\} \rightarrow Face\_Trigger: \{\} \\ \rightarrow Blur\_Area\_Pixelate: \{'blocks': 5\} \end{aligned}$$

Listing 1: An example of a privacy enforcement model.

The first link in the chain defines the media source to which a privacy chain should be applied. The above example uses the provided chain for 'video' streams. Even though the current framework only supports video stream processing, the detection and transformation of streaming data follow the same pattern for all data types. Furthermore, we developed a whitelist mechanism: in case the SFU does not have active privacy chains for an incoming data type, clients cannot consume it. An empty chain that does not contain any triggers or transformations (e.g., video:{'tag':'webcam'}) is a configured rule that allows the data to be consumed without applying further operations.

The two remaining links in Listing 1 describe the operations executed on every video frame that the SFU processes. First, we detect whether or not there are human faces in the video frame (Face_Trigger:{}), and for every detected face, a blur transformation (Blur_Area_Pixelate:{'blocks':5}) is applied to the respective region. The curly braces contain parameters passed to the functions; the only given parameter represents the number of pixelated squared in the blurred area. Such a chain can be quickly converted into a flow diagram and vice-versa: Figure 5 contains a graphical representation of the face blurring function presented above.

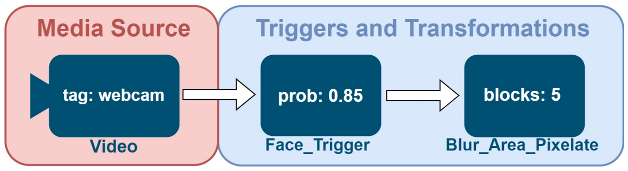


Fig. 5: Privacy chain as a figure for blurring faces.

In the below Figure 6, we show the effect that the 'blocks' parameter has on the blurred images. A value like five blocks produces a prettier result, but with an increasing number of blocks, the processing time rises equally; this will be evaluated later. A high number of blocks in fact results in a decreased anonymization effect, as we see in the last image with a number of $30 * 30 = 900$ blocks calculated.

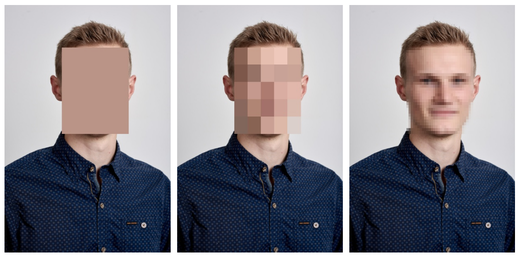


Fig. 6: Blurred face with a number of $1 * 1$, $5 * 5$, or $30 * 30$ blocks calculated.

5.3.1 Model Syntax and Compilation

Privacy chains are compiled internally by splitting a model string along the arrows and processing the links of the chain in sequence: the first link must define a valid media source; every further link must be correctly resolved through a function map. The conversion map translates a function name as a string (e.g., 'Face_Trigger') to a Python function (e.g., Face_Trigger()) from Section 5.2.1. Parameters attached to functions must be specified in curly braces and describe a valid Python dictionary; otherwise, the compilation will fail and indicate where the syntax is invalid. Depending on whether a function string is resolved to a trigger or transformation function, the internal method signature is different; we will discuss below what methods can be chained together without compilation errors. To save execution time, a chain is processed up to the nth element that yields a result. If a link n resolves to a trigger function that does not detect any pattern in the frame, the remaining chain links $n+1, n+2, ..$ are skipped.

The last aspect of specifying privacy chains is related to the "tag" parameter included in the above chain and Figure 5. Through tags, it becomes possible to define many chains in only one privacy model deployed on the SFU. A privacy model could thus consist of two or more chains, e.g., one for anonymizing car plates ('road scene'), and another for blurring faces ('video vigilance'), which use entirely different trigger and transformation functions. Now whenever a producer connects to the SFU, it supplies a tag (e.g., 'road scene'), which is stored alongside the peer connection on the SFU. Before the SFU starts to process any provided data, it browses the list of available privacy chains and selects the one most fitting, i.e., the one with the same tag or at least with a matching data type. The SFU now processes the data according to the 'road scene' chain, applying only the triggers and transformations specified for this chain.

5.3.2 Trigger and Transformation Functions

The method signature of the trigger function describes the expected parameters and the function's return values. For some triggers, we also provide a list of labels that the function returns (e.g., 'male' when using the Gender_Trigger). A developer/end-user must specify a label parameter in the privacy chain string for these functions.

We introduce the term 'boxes' as a parameter and return type: a box is precisely a tuple of two points in the video frame that spans a rectangular area between them. So one box would be, for example, [0, 0, 640, 480], where [0,0] and [640,480] represent points on the plain, and we define the area between them as the content of the box. A single box b_1 has the structure of $[x_1, y_1, x_2, y_2]$, while a set of boxes is encoded as $[b_1, b_2, b_3, \ldots, b_n]$ and can be used to pass sections of image frames between trigger and transformation functions. We can chain the above trigger functions arbitrarily to detect patterns in the video frame (see Listing 2). All of them accept boxes as parameters to restrict the area of the trigger processes; they can either be specified explicitly or passed between functions anonymously. This means that we can specify a sub-chain of two trigger functions linked together where the results (i.e., video frame & list of boxes) of chain c_1 are passed internally to chain c_2.

$$X \to Face\_Trigger : \{\}$$
$$\to Age\_Trigger : \{'label' : (25-32)\} \to Y$$

Listing 2: Face and Age trigger functions in a privacy enforcement model.

Notice that a trigger function c_n is not executed if a previous trigger c_{n-1} does not return any boxes. This indicates that c_n did not detect any privacy-violating pattern and released the frame without transformation. Furthermore, boxes also play a central role in transformations. Each transformation requires at least a box or a set of boxes to specify explicitly which areas in the video frame should be processed.

6 Evaluation

To evaluate the performance of the proposed solution, we focus on two central metrics that concern the streaming capabilities of the implementation: the latency to/from the SFU, and the SFU's performance. In this context, we define the overall streaming latency as the time it takes to transfer a frame from the producer, over the SFU, to the consumer.

Since our frames only flow in one direction, from the producer to the consumer, we use the term Round Trip Time (RTT) interchangeably.

The latency between the stream producer and SFU and from the SFU to the stream consumer is supposedly related to the network architecture in which we deploy the framework. An intuitive assumption is that the latency is lower if the SFU is close to the producer/consumer client instead of a cloud center. We do not expect a cloud setup to yield better results than an edge architecture, but we will evaluate how big the difference between both architectures is for our WebRTC streaming setup. We estimate that the first share of the RTT, producer → SFU, does not diverge significantly from the share SFU → consumer, at least not based on the edge or cloud architecture. In that context, we provide results on the latency from producer → SFU for the cloud setup and from SFU → consumer for the edge setup. In Figure 7, we visualized how the overall RTT is related to the latency between SFU, producer, and consumer (Label 1 & 3). Their share of the RTT is evaluated through metrics together with the SFU's performance (Label 2).

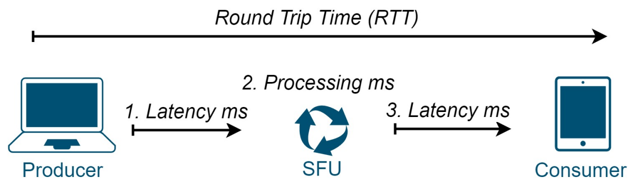


Fig. 7: Evaluating three shares of the prototype's RTT.

6.1 Privacy Models for Evaluation

In Figure 8, we give an overview of the four privacy models that will be evaluated. Each model contains a reasonable combination of trigger and transformation functions, where a sequence of colored arrows represents one model. For example, the orange chain, which describes *Model #1*, reads as follows: receiving data from a video source, applying a face trigger, and finally blurring all detected faces. We evaluate the overall RTT (i.e., total streaming latency) and the function's execution time in the SFU. The remaining chains (purple, green, gray) describe equally a privacy model and can be read the same way.

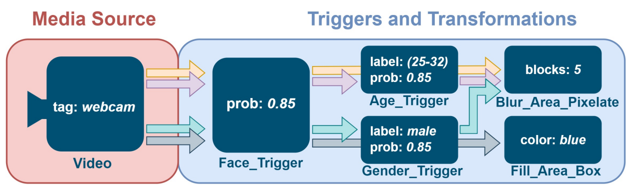


Fig. 8: Evaluated models (orange, purple, green, and grey).

Our evaluation goes further than comparable research on privacy policy enforcement [8], [11], where streams were merely secured through access-control restrictions. The privacy models submitted to the SFU facilitate finer-grained configurability for privacy policies, greatly increasing the variety of tools a developer/end-user has available.

We provide two prerecorded videos streamed from the producer to the SFU to evaluate the SFU's performance processing the stream. Using prerecorded videos instead of live streams prevents outer influences from distorting the measurement sessions. The second video originates from the first one; the difference is in its resolution and frame rate decreased. Therefore, there are no substantial differences in the content of the videos, only that some details of the video stream might be lost when decreasing the frame rate. Table 3 provides a summary of the specifications mentioned for the recorded videos.

TABLE 3: List of two recorded videos that are used to evaluate the prototype

ID	Width	Height	Duration	Frame Rate
Video #1	1280px	720px	00:00:10	30 FPS
Video #2	640px	320px	00:00:10	16 FPS

A central aspect of choosing a frame rate when providing a certain media type is that it directly affects the time frame for processing a data frame. With a growing frame rate, i.e., we transfer a higher number of frames per second (FPS), the available time frame for processing is decreased.

6.2 Hardware Setup

The RTT in our framework consists of three major shares: the latency of the peer connection between the producer and SFU, the performance of the SFU, and the latency between SFU and consumer clients. We evaluated the SFU's performance on a single edge device with the given privacy models. For the remaining share of the RTT, we conducted two further measurements: the influence on the producer's latency depending on whether the SFU is deployed on the edge or the cloud and the influence of the consumer's latency depending on the consumer device type.

The hardware specifications of the device we used to evaluate the SFU is depicted in Table 4. Our edge device has an NVIDIA graphic card installed which supports NVIDIA Cuda for GPU-accelerated graphic processing of video streams.

TABLE 4: Hardware specifications of the device used to evaluate the prototype

OS Version	CPU	RAM	GPU	Cuda
Windows 10	AMD FX-6300	16 GB	NVIDIA GTX 960	11.4.3

All the entities used for the evaluation scenario are reflected in Figure 9, which contains a Python producer client application that connects to the SFU through SDP. Once a stable connection exists, the producer client starts streaming the prerecorded videos from Table 3 with the respective frame rate and resolution. The SFU processes the received stream according to the supplied privacy model from Figure 8, and the transformed stream is then streamed to consumer devices. The consumer devices connect themselves to the SFU through a web application hosted on the edge device. The application simplifies consumer connection progress and takes charge of the SDP handling in the background. For clarification, the consumer app is merely supplied through the SFU but is executed on consumer devices. Therefore it supposedly does not affect the performance of the SFU. The edge device that hosts the SFU and the consumer app has the device specifications we presented in Table 4. This is the environment we used to evaluate the performance of the privacy models and individual trigger and transformation functions.

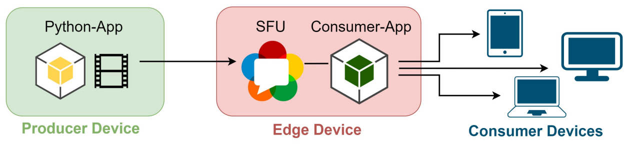


Fig. 9: Streaming architecture for prototype evaluation.

To compare the latency from the producer to the SFU instances in an edge/cloud environment, we deployed the SFU once on the edge device from Table 4, and once on AWS. To simplify the deployment on AWS, we created a Docker image of the SFU, including all respective dependencies. The image was shared through Docker Hub and can be pulled and executed on an arbitrary Docker instance. The AWS server we used to evaluate the latency in cloud setups was an EC2 instance, namely the t2.micro, with a single vCPU and 1 GB of RAM. The cloud setup is evidently not as powerful as the edge device we use for evaluating the SFU's performance, but this is negligible since we do not process video frames within AWS but solely measure the latency between the producer client and AWS.

The remaining part, evaluating the latency between the SFU and different consumer device types, was conducted using a set of three devices. All of those devices connected themselves to the SFU as described through the web application hosted on the edge device, and they were connected one each during a measurement session. However, our SFU can relay video frames to multiple destinations in parallel. The different consumer devices were as follows:

1) A smartphone is a useful choice if we want to consume streams on the go. We used the Samsung Galaxy S9 (SM-G960F/DS) as a mobile device.
2) A laptop, which can be seen as a hybrid solution for consuming streams on the go or as well in an office scenario. We consumed the stream on the Lenovo Yoga 530-14IKB. Due to the absence of a 4G/5G module, we connect to the SFU through the available Wi-Fi connection.
3) Consuming the stream locally on the edge device is a special case because we do not transfer frames between different devices.

6.3 Streaming Latency

This section focuses on the latency between the producer and SFU and SFU towards the consumer.

In both cases, the producer streamed *Video #1* over the SFU to the consumer, which has a frame rate of 30; this is relevant because the frame rate defines how often the latency can be calculated. We did not calculate the latency ourselves but accessed it through the peer-to-peer connection statistics maintained by the SFU. These statistics are updated and accessed for evaluation; however, we cannot measure the latency more often than 30 times a second.

We measured the latency over a total time of 60 seconds, retrieving the latency once a second (i.e., 60 values for each session that we can compare to each other). Within the SFU, we always had *Model #1* executed, though this supposedly does not make any difference for the producer's or consumers' latency. The producer client was always executed on the same IoT device, the laptop from Section 6.2. Three consumer devices were used to compare their results to retrieve the transformed stream. Consumer and producer streams were started simultaneously, and the results were captured from the beginning of the connection. We wanted to capture a possible unstable streaming latency at the beginning of the established connection. Therefore, within 60 seconds, we would supposedly already arrive in a stable state where we have less fluctuation in the latency.

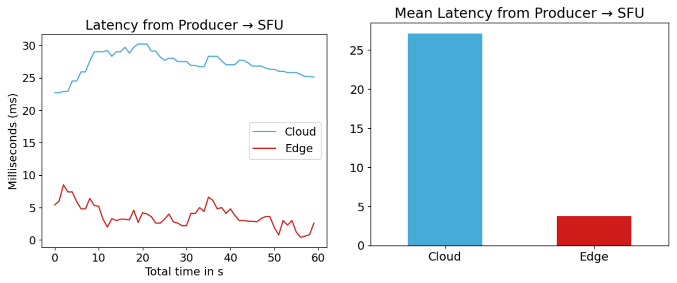


Fig. 10: Producer latency over 60 seconds for cloud and edge setups with mean values.

In Figure 10, we visualized the edge and cloud setup latency in comparison over 60 seconds. The average values provided in the right sub-graphic are for the same respective time frame. It is clearly visible that the evaluated deployment environments greatly influence the latency between the producer and the SFU. Transferring streaming data to a distant cloud server increases the RTT considerably, whereas processing it directly on edge helps maintain the low latency and thus supports time-critical tasks.

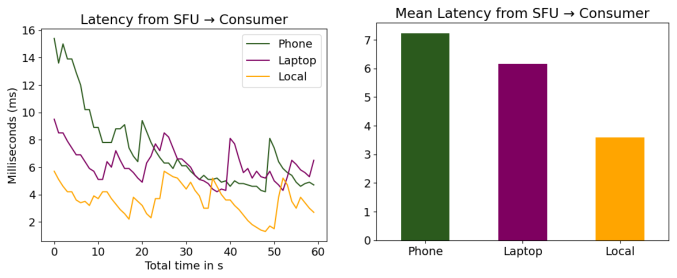


Fig. 11: Latency of different consumer devices over 60 seconds with mean values.

In Figure 11, we evaluated the latency between multiple consumer devices and the SFU deployed on edge. The three devices used are the ones described previously in the hardware setup. We recall that the first consumer device was a smartphone, the second a laptop, and the last consumer was executed locally on the SFU's device. We tracked the latency over 60 seconds and provided the respective mean values in the right sub-graphic. We deduce from the results that the WIFI-equipped laptop has lower latency than the smartphone running over 5G, indicating that a fast transition from wireless transmission to wired one can slightly improve the latency. Directly consuming the data locally on the edge device (yellow line) is unarguably the fastest way because there is no need to transfer it to another device, however, there might only be limited use cases where this is applicable.

6.4 Privacy Model Performance

In Figure 8, we introduced the four privacy models that are used for evaluating the SFU's performance. For each of them, we provide the overall execution time alongside the individual functions' performance. The time to process frames on the SFU will be evaluated with the GPU-acceleration enabled, only for Section 6.6 we will visualize for two privacy models the different performances with/without GPU support.

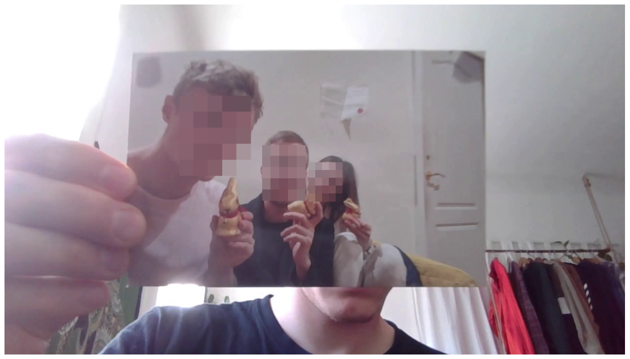


Fig. 12: Resulting stream with multiple faces transformed.

An important detail is that the last third of the produced stream contains a printed photo with three additional human faces. To give a precise idea of what the transformed stream looks like under these circumstances, we included Figure 12. We mention this because the larger number of human faces may lead to a different result for the trigger and transformation functions. On the other hand, the frame size stays mostly unchanged, so we would not expect the latency to vary throughout the video stream.

6.5 Triggers and Transformations

Figure 13 visualizes the performance on the SFU when processing the four defined privacy models, while *Video #1* is streamed from the producer. The colored lines refer to the individual functions' performances, while the black line always represents the accumulated chain processing time. Because a higher resolution is desired for evaluating the computation time, we evaluated the processing time once every frame, instead of only every 30 frames like in Section 6.3. However, to avoid an increasingly dense x-axis, we stopped the replay of the demo video after 10 seconds. We evaluate 10 seconds of *Video #1*, which was recorded and replayed by the producer at its default frame rate of 30. By multiplying 30 $FPS * 10\ Seconds$ we end up with exactly 300 values for individual performances and overall execution time. On the x-axis of the given figure, we iterate over the processed frames and the performance of the functions for frame X, frame 1 is the first streamed frame, and frame 300 is the last one transferred.

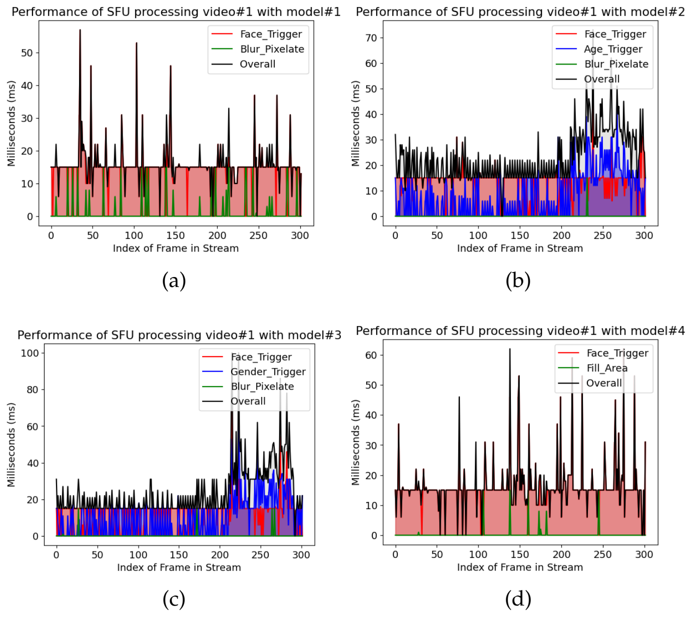


Fig. 13: Individual performance of functions for *Video #1* and the 4 privacy models.

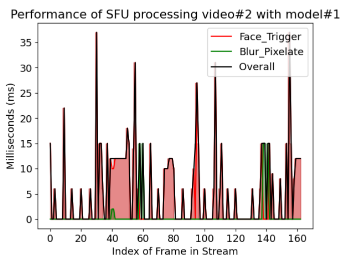
In Figure 14, we use the exact same setup to visualize the performance for processing *Video #2*. As presented in Section 6.2, the differences are the resolution and the decreased frame rate for the second video. We included the overall performance in combination with the individual results to give a precise idea of how the accumulated processing time is based on the distinct triggers and transformations. Since *Video #2* and the resulting video stream only have a frame rate of 16, the overall number of values is reduced equivalently to 160 instead of 300.

Figure 15 describes the overall distribution of the values. We used a standard representation where the interquartile range around the median is 50% and the upper and lower whiskers occupy a further 45% of the distribution. The remaining 5% of the values are represented as outliers, i.e., dots beyond the upper and lower whiskers.

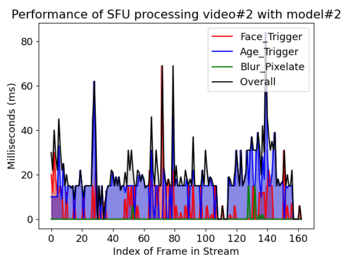
In Figure 16, we show the overall chain performances. The left subgraph contains the computed result for *Video #1* and the right one the results for *Video #2*. Both were applied to the exact same set of privacy models. We iterate again over the whole set of values that we captured when processing every frame of the stream. The x-axis of the two figures is not identical because the two videos have different frame rates. We deduce that decreasing the frame resolution significantly reduces the time required to process a video frame. However, the result depends on the privacy chain and the contained trigger and transformation functions. The time required to process *Model #1* & *#4* becomes almost negligible, while *Model #2* and *#3* only receive a small performance boot by decreasing the stream's resolution.

6.6 Parameter Tuning & GPU Acceleration

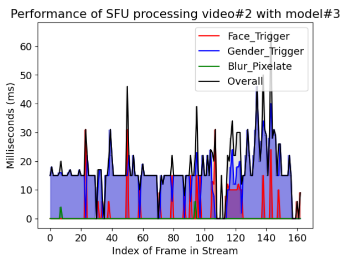
Another critical aspect to further evaluate is the configuration of privacy functions and their tuning, as well as their



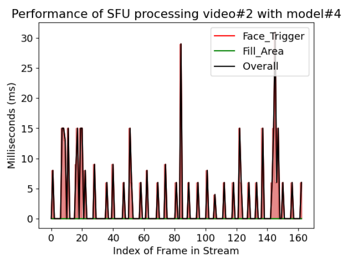
(a)



(b)

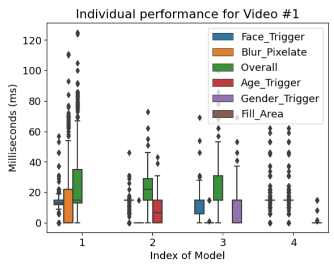


(c)

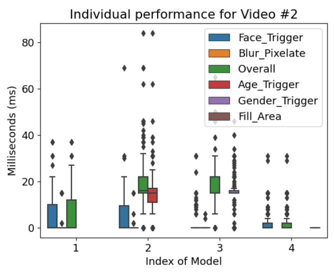


(d)

Fig. 14: Individual performance of functions for *Video #2* and the 4 privacy models.



(a)



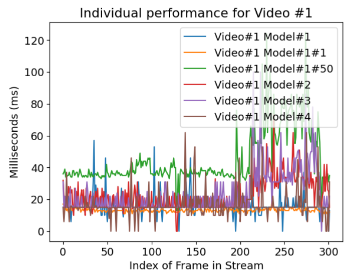
(b)

Fig. 15: Statistical distribution of function's performances for *Video #1* & # 2.

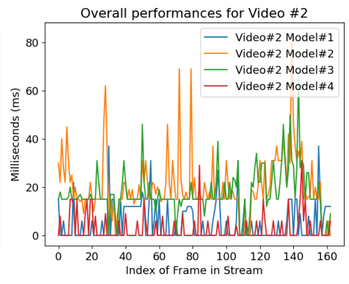
dependence on their environment for accelerated processing. To segregate the effects of different parameter values and the GPU-acceleration, we will only evaluate *Video #1* and the first two privacy models, but with different settings to highlight diverging results. We will only process the first 10 seconds of the video to increase the detail on the x-axis. With the given video, we will iterate over 300 frames and measure the performance of the individual function and the overall processing time.

Figure 17 visualizes the results of the SFU's performance with the tuned *blocks* parameter of the Blur_Area_Pixelate function, as described in the evaluated scenario. The left subfigure contains the performance with *blocks: 1* and the right one with *blocks: 50*. We recall that according to Figure 6 an increasingly large *blocks* parameter will not improve the anonymization effect of the transformation. The tuned parameters can be compared with the unmodified results of *Video #1* and *Model #1* in Figure 13; the last third of the frames processed contains an image likewise with multiple human faces.

The results without GPU-acceleration are provided

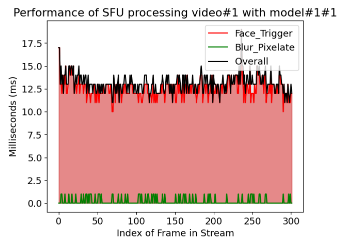


(a)

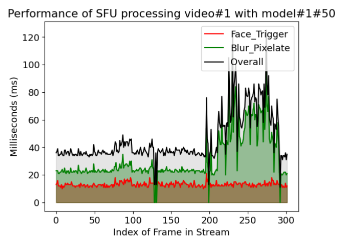


(b)

Fig. 16: Overall performance of evaluated privacy models on *Video #1* & #2.



(a)



(b)

Fig. 17: Performance of Blur_Area_Pixelate for *blocks* values $\{1, 50\}$.

through Figure 18. The x-axis contains all frames processed on the SFU when streaming *Video #1*. The colored lines likewise represent the individual performances of the trigger and transformation functions; the black line depicts the time elapsed for processing the privacy chain. Performance was evaluated on the same device with the exact hardware specifications to avoid external factors interfering with the results. To deactivate the GPU support, we implemented a configuration parameter that allows switching between the available providers (CPU or GPU) for the ONNX model.

6.7 Discussion

From Figure 13, we can obtain numerous details about the performance for processing *Video #1* with the evaluated privacy models. The stream resembled a video source with a resolution of 1280x720px and a frame rate of 30 FPS, which could contain sufficient details for a surveillance camera or an AR device similar to Section 3. We observe that the overall performance for processing the privacy chains is relatively constant for *Model #1* & *#4*, which applied a Face_Trigger without consecutive Age- or Gender_Trigger. Both models sporadically showed high peak execution time, mainly due to abnormalities in the execution of the Face_Trigger. Nevertheless, over the full measurement, both models maintained a latency close to their mean value of 15ms. The overall latency depended entirely on the result of the Face_Trigger, as we can observe from the box plot in Figure 15. We could not detect any impact on the overall performance depending on whether we applied the Blur_Area_Pixelate or Fill_Area_Box function.

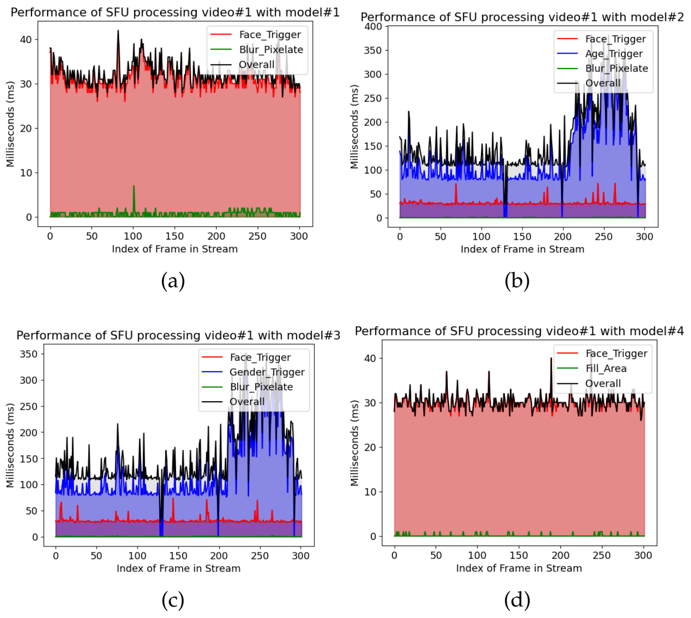


Fig. 18: Individual performance of functions for *Video #1* without GPU-acceleration.

We showed how tuning parameters (i.e., we varied the number of blocks in the Blur_Area_Pixelate function) affect the model's performance. Figure 17 contains the results for processing *Video #1* and *Model #1* with 1 or 50 blocks. We remember from Figure 6 that a higher number of blocks is not necessarily a guarantee of better anonymization; thus, we see it more as a general experiment on parameter tuning. We can observe that using only one block provides similar results to applying Blur_Area_Pixelate with five blocks in Figure 13. However, a notable fact is that the streaming session with five blocks had more peak times caused by the Blur_Area_Pixelate function, so we suspect that a lower parameter reduces the frequency of peaks, which should be affirmed with further evaluations. The quality of the stream is critical to the resulting performance, as we can see in the different results of the evaluated video streams in Figure 13 and 14. We may either decrease the time it requires to process a frame by decreasing its quality (e.g., resolution), or increase the available time frame for the computation by decreasing the FPS of the stream. It further showed, by comparing Figure 13 and Figure 18, that a GPU provides significantly lower latency for video trigger functions, while transformation functions maintained unaffected.

7 Conclusion & Future Work

We have presented a privacy-enforcing framework for transforming data streams on edge. Our first contribution introduced a novel approach that enables domain experts to specify high-level privacy policies in a human-readable way. The introduced edge-based runtime mechanism controls and interprets privacy policies deployed on edge devices. Throughout various experiments, we show the applicability and feasibility of the approach. The latency and network overhead are within acceptable boundaries, while the introduced method offers several benefits over a centralized cloud-based system. Our prototype provided promising results and led to a multitude of open challenges that should be addressed in future work: (i) We proved the abstract concept for video stream; nonetheless, it remains to extend and evaluate our prototype with other data types (e.g., audio streams) to underline the universality of our framework (ii) We evaluated the performance on a single-edge powerful device (i.e., deploying the SFU on the NVIDIA Jetson). In future work, we plan to evaluate privacy models with different complexities and show the framework's applicability on low-powered edge devices.

Acknowledgment

Research has partially received funding from EU Horizon Framework grant agreements No. 871525 (FogProtect), No. 101079214 (AIoTwin), and No. 101070186 (TEADAL).

References

[1] I. Murturi, C. Jia, B. Kerbl, M. Wimmer, S. Dustdar, and C. Tsigkanos, "On provisioning procedural geometry workloads on edge architectures," in *Proceedings of the 17th International Conference on Web Information Systems and Technologies, WEBIST 2021*, pp. 354–359.

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[3] I. Murturi and S. Dustdar, "A decentralized approach for resource discovery using metadata replication in edge networks," *IEEE Transactions on Services Computing*, 2021.

[4] I. Murturi, A. Egyed, and S. Dustdar, "Utilizing ai planning on the edge," *IEEE Internet Computing*, vol. 26, no. 2, pp. 28–35, 2022.

[5] B. Sedlak, I. Murturi, and S. Dustdar, "Specification and operation of privacy models for data streams on the edge," in *2022 IEEE 6th International Conference on Fog and Edge Computing (ICFEC)*, 2022, pp. 78–82. DOI: 10.1109/ICFEC54809.2022.00018.

[6] A. Wang, J. Shen, C. Wang, H. Yang, and D. Liu, "Anonymous data collection scheme for cloud-aided mobile edge networks," en, *Digital Communications and Networks*, vol. 6, no. 2, pp. 223–228, May 2020, ISSN: 2352-8648. DOI: 10.1016/j.dcan.2019.04.001.

[7] O. Hajihassani, O. Ardakanian, and H. Khazaei, "Anonymizing Sensor Data on the Edge: A Representation Learning and Transformation Approach," en, *arXiv:2011.08315 [cs]*, Aug. 2021, arXiv: 2011.08315.

[8] C. Lachner, T. Rausch, and S. Dustdar, "Context-Aware Enforcement of Privacy Policies in Edge Computing," Jul. 2019, pp. 1–6. DOI: 10 . 1109 / BigDataCongress.2019.00014.

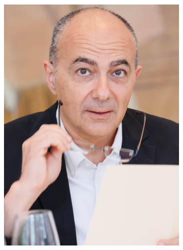
[9] M. Khavkin and M. Last, "Preserving Differential Privacy and Utility of Non-stationary Data Streams," in *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, ISSN: 2375-9259, Nov. 2018, pp. 29–34. DOI: 10.1109/ICDMW.2018.00012.

[10] N. Jha, T. Favale, L. Vassio, M. Trevisan, and M. Mellia, “Z-anonymity: Zero-Delay Anonymization for Data Streams,” en, *2020 IEEE International Conference on Big Data (Big Data)*, pp. 3996–4005, Dec. 2020. DOI: 10.1109/BigData50022.2020.9378422.
[11] P. Baniya, G. Bajaj, J. Lee, A. Bastani, C. Francis, and M. Agumbe Suresh, “Towards Policy-aware Edge Computing Architectures,” in *2020 IEEE International Conference on Big Data (Big Data)*, Dec. 2020, pp. 3464–3469. DOI: 10.1109/BigData50022.2020.9377982.
[12] C. Tsigkanos, C. Avasalcai, S. Dustdar, and S. Dustdar, “Architectural Considerations for Privacy on the Edge,” en, *IEEE Internet Computing*, vol. 23, no. 4, pp. 76–83, Jul. 2019, ISSN: 1089-7801, 1941-0131. DOI: 10.1109/MIC.2019.2935800.
[13] Y. Mao, J. Feng, F. Xu, and S. Zhong, “A Privacy-Preserving Deep Learning Approach for Face Recognition with Edge Computing,” *HotEdge*, 2018.
[14] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, “LAVEA: Latency-aware video analytics on edge computing platform,” in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, ser. SEC ’17, New York, NY, USA: Association for Computing Machinery, Oct. 2017, pp. 1–13, ISBN: 978-1-4503-5087-7. DOI: 10.1145/3132211.3134459.
[15] M. M. Peixoto and C. Silva, “Specifying privacy requirements with goal-oriented modeling languages,” in *Proceedings of the XXXII Brazilian Symposium on Software Engineering*, ser. SBES ’18, New York, NY, USA: Association for Computing Machinery, Sep. 2018, pp. 112–121, ISBN: 978-1-4503-6503-1. DOI: 10.1145/3266237.3266270.
[16] P. Maiti, J. Shukla, B. Sahoo, and A. K. Turuk, “Efficient Data Collection for IoT Services in Edge Computing Environment,” in *2017 International Conference on Information Technology (ICIT)*, Dec. 2017, pp. 101–106. DOI: 10.1109/ICIT.2017.40.
[17] K. Cao, Y. Liu, G. Meng, and Q. Sun, “An Overview on Edge Computing Research,” *IEEE Access*, vol. 8, pp. 85 714–85 728, 2020, Conference Name: IEEE Access, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2991734.
[18] X. Cao and S. Madria, *Efficient Geospatial Data Collection in IoT Networks for Mobile Edge Computing*. Oct. 2019. DOI: 10.1109/NCA.2019.8935061.
[19] Y. Du, “Collaborative Crowdsensing at the Edge,” Ph.D. dissertation, Jul. 2020.
[20] A. Sabbioni, L. Rosa, A. Bujari, L. Foschini, and A. Corradi, “A Shared Memory Approach for Function Chaining in Serverless Platforms,” in *2021 IEEE Symposium on Computers and Communications (ISCC)*, ISSN: 2642-7389, Sep. 2021, pp. 1–6. DOI: 10.1109/ISCC53001.2021.9631385.
[21] linzai, *Ultra-Light-Fast-Generic-Face-Detector-1MB*, Feb. 2022. [Online]. Available: https://github.com/Linzaer/Ultra-Light-Fast-Generic-Face-Detector-1MB (visited on Feb. 9, 2022).
[22] asiryan, *Age and Gender Classification using Convolutional Neural Networks*, en, 2021. [Online]. Available: https://github.com/onnx/models (visited on Feb. 9, 2022).
[23] R. Rothe, R. Timofte, and L. V. Gool, “DEX: Deep EXpectation of Apparent Age from a Single Image,” en, in *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*, Santiago, Chile: IEEE, Dec. 2015, pp. 252–257, ISBN: 978-1-4673-9711-7. DOI: 10.1109/ICCVW.2015.41.
[24] A. Rosebrock, *Blur and anonymize faces with OpenCV and Python*, en-US, Apr. 2020. [Online]. Available: https://www.pyimagesearch.com/2020/04/06/blur-and-anonymize-faces-with-opencv-and-python/ (visited on Feb. 9, 2022).

Boris Sedlak is currently a Ph.D. student of the Distributed Systems Group at TU Wien, Austria. He received his B.Sc. in Media Informatics at the University of Applied Sciences in St. Pölten, and his M.Sc. in Software Engineering & Internet Computing at the TU Wien. He was working four years in the field of software engineering before focusing on his current research interests: edge computing, data transformations, and software architecture.

Ilir Murturi is currently a Postdoctoral researcher with the Distributed Systems Group, TU Wien, Vienna. He received his M.Sc. degree in Computer Engineering from the University of Prishtina, Kosova. He received his Ph.D. from TU Wien in 2022. Previously, he worked as a University Assistant at TU Wien and the University of Prishtina. His current research interests include the Internet of Things, Edge Computing, EdgeAI, and privacy in distributed, self-adaptive and cyber-physical systems.

Praveen Kumar Donta(SM’22) is a Postdoctoral researcher in the Distributed Systems Group, TU Wien, Austria since July 2021. He received his Ph.D. from the Department of CSE in IIT (ISM), Dhanbad, India in June 2021. He was a visiting Ph.D. student during his Ph.D. at the University of Tartu, Estonia. He received his M.Tech and B.Tech from JNTU Anantapur, India in 2014, and 2012. His current research on Learning-driven distributed computing continuum systems.


Schahram Dustdar (F’16) is Full Professor of Computer science heading the Research Division of Distributed Systems at the TU Wien, Austria. He is founding Co-Editor-in-Chief of the new ACM Transactions on Internet of Things (ACM TIoT) as well as Editor-in-Chief of Computing (Springer). He is an Associate Editor of IEEE Transactions on Services Computing, IEEE Transactions on Cloud Computing, ACM Transactions on the Web, and ACM Transactions on Internet Technology, as well as on the editorial board of IEEE Internet Computing and IEEE Computer. Dustdar is Recipient of the ACM Distinguished Scientist Award (2009), the ACM Distinguished Speaker ward (2021), the IBM Faculty Award (2012), an Elected Member of the Academia Europaea: The Academy of Europe, where he is Chairman of the Informatics Section, as well as an IEEE Fellow. In 2021 Dustdar was elected President for Asia-Pacific Artificial Intelligence Association (AAIA).