TU WIEN Informatics

Specification and Operation of Privacy Models for Data Streams on the Edge

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Boris Alexander Sedlak, BSc
Matrikelnummer 01529846

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dr. Schahram Dustdar
Mitwirkung: Ilir Murturi, MSc

Wien, 23. März 2022

Boris Alexander Sedlak	Schahram Dustdar

TU WIEN Informatics

Specification and Operation of Privacy Models for Data Streams on the Edge

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Boris Alexander Sedlak, BSc
Registration Number 01529846

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dr. Schahram Dustdar
Assistance: Ilir Murturi, MSc

Vienna, 23rd March, 2022

Boris Alexander Sedlak	Schahram Dustdar

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Boris Alexander Sedlak, 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.

Wien, 23. März 2022

Boris Alexander Sedlak

Danksagung

Ich möchte zuallererst meinen Betreuern Ilir Murturi und Schahram Dustdar danken, die mich bei meinen Anstrengungen unterstützt und auf einen herausfordernden, aber spannenden Pfad geleitet haben. Kein Schritt war vergebens, und so münden etliche Monate an Arbeit schlussendlich in einem Resultat, auf das ich stolz bin.

Weiters möchte ich Laura Dusl danken, die mir geholfen hat meine Gedanken zu sortieren und auf das Wesentliche fokussiert zu bleiben. Vieles verdanke ich insbesondere meiner Familie und meinen Freunden, die mich auf meiner bisherigen Reise begleitet haben, ich stehe hier und heute durch eure Unterstützung.

Acknowledgements

My utmost thanks go to my advisors, Ilir Murturi and Schahram Dustdar, who would not let me walk the easy path, but instead guide me towards a result that I am proud of by today. It is not that I would have liked to take it easy, but the challenge encouraged me and would eventually pay off.

Thanks to Laura Dusl, who would be the one to help me sort my thoughts and stay focused. Thanks to my friends and family who supported me on my journey, I stand here and now because of all of you.

Kurzfassung

Die wachsende Anzahl an Internet of Things (IoT)-Geräten erzeugt eine erhebliche Menge an verschiedensten Daten, welche persönliche oder vertrauliche Informationen enthalten, die nicht für die Öffentlichkeit bestimmt sind. Üblicherweise werden Datenschutzrichtlinien zum Schutz von sensiblen Daten über Systeme durchgesetzt, die über eine Vielzahl an Ressourcen verfügen, wie am Beispiel Cloud Computings. Die schiere Menge an Datenströmen, heterogenen Geräten, und involvierten Netzwerken beeinträchtigt jedoch die globale Latenz und erhöht mit zunehmender Distanz von der Datenquelle die Gefahr, dass Daten abgegriffen werden. Aus diesem Grund müssen Datenströme auf dem IoT-Gerät oder anderen Geräten in naher Umgebung aufbereitet werden, um den Schutz von persönlichen Daten zu gewährleisten.

In dieser Arbeit präsentieren wir die Grundstruktur eines Systems, welches Datenströme innerhalb von Edge-Netzwerken zum Schutz von persönlichen Daten transformiert. Dies geschieht über leistungsfähige Geräte in der Nähe der Datenquelle. Jedes Mal, wenn ein IoT-Gerät sensitive Daten erfasst, wird der Datenstrom anhand einer Gruppe von Regeln umgeformt. Dieser Prozess ist über eine Reihe an *Auslösern* und *Transformationen* definiert (einem Datenschutzmodell), welche ein direktes Abbild der Datenschutzrichtlinien eines Interessenten darstellt. Unsere Arbeit beantwortet, wie Datenschutzrichtlinien in einem solchen Modell wiedergegeben und durch eine zugrundeliegende Verarbeitungsumgebung durchgesetzt werden können.
Als Nachweis für die Machbarkeit dieses Systems haben wir einen Prototyp entwickelt, der Video-Streams über ein Edge-Gerät leitet und dort transformiert. Die Ergebnisse haben gezeigt, dass das eingesetzte Gerät konstant in der Lage war menschliche Gesichter innerhalb von 15ms zu erkennen und zu anonymisieren. Demnach könnte dieser Video-Stream mit bis zu 60 Frames pro Sekunde (FPS) übertragen und transformiert werden, ohne Daten zu verlieren. Gekoppelt mit der niedrigen Latenz, die wir zwischen Stream-Rezipienten und Edge-Gerät gemessen haben, bietet unsere Arbeit vielversprechende Aussichten. Zukünftige Forschungsarbeit muss sich jedoch der Unterstützung von weiteren Datentypen widmen, um die allgemeine Eignung des Systems zu verdeutlichen.

Abstract

The growing number of Internet of Things (IoT) devices generates massive amounts of diverse data, including personal or confidential information (i.e., sensory, images, etc.) that is not intended for public view. Traditionally, predefined privacy policies are enforced in resource-rich environments such as the cloud to protect sensitive information from being released. However, the massive amount of data streams, heterogeneous devices, and networks involved affects the overall latency, and the possibility of having data intercepted grows as it travels away from the data source. Therefore, such data streams must be transformed on the IoT device or within available devices (i.e., edge devices) in its vicinity to ensure privacy.

In this paper, we present a privacy-enforcing framework that transforms data streams within edge networks. We ensure privacy close to the data source, using powerful edge devices to analyze the stream's content. Whenever an IoT device captures personal or confidential data, we transform the stream according to a predefined set of rules. How the stream is modified is defined precisely by a chain of trigger and transformation functions (a privacy model) that directly represents a stakeholder's privacy policies. Our work answers how to represent such privacy policies in a model and enforce transformations on the edge through an underlying processing environment.
To prove the presented concept, we developed a prototype that routes and transforms video streams over an edge device. Results showed that the evaluated device was capable of continuously detecting and anonymizing human faces within 15ms. Thus, we might transform this stream at up to 60 Frames Per Second (FPS) without dropping any frame. Our research yields promising results when paired with the low latency we experienced between stream consumer and edge device. However, future research must focus on streaming different data types to underline the universality of our framework.

Contents

Kurzfassung **xi**

Abstract **xiii**

Contents **xv**

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Problem Statement . . . 2
1.3 Research Questions . . . 5
1.4 Structure . . . 6

2 Background **7**
2.1 Edge Computing . . . 7
2.2 Data Streams . . . 8
2.3 Privacy Protection . . . 10

3 Related Work **15**
3.1 Stream Transformations . . . 15
3.2 Privacy Modelling . . . 18
3.3 Remaining Challenges . . . 19

4 The Framework **21**
4.1 Concept . . . 21
4.2 Stream Routing . . . 25
4.3 Pattern Detection and Transformations . . . 27
4.4 Privacy Model Specification . . . 32
4.5 Privacy Model Deployment . . . 41

5 Evaluation Methodology **45**
5.1 Overview . . . 45
5.2 Evaluated Scenario . . . 47
5.3 Experimental Setup . . . 49

6 Results **53**
6.1 Streaming Latency 53
6.2 Privacy Model Performance 55
6.3 Edge Device Deployment 61

7 Discussion **63**
7.1 Device Performance 63
7.2 Processing Boundaries 67
7.3 Proof of Concept 69

8 Conclusion **71**
8.1 Research Questions 72
8.2 Future Work 74

List of Figures **75**

List of Tables **77**

Bibliography **79**

CHAPTER 1

Introduction

1.1 Motivation

In recent years, the number of Internet of Things (IoT) devices has widely increased, while each device continuously generates massive amounts of various data. Several examples exist; for instance, a security camera in a smart factory that produces a constant stream of high-quality images or a sensor network in a car that opportunistically provides live information about road conditions. Traditionally, data produced in such scenarios is processed or aggregated at a central cloud server. Research literature shows that data can often be significantly large (e.g., images, audio, video streams, 3D content, etc.) and usually requires processing with low latency [1]. Nevertheless, the massive amount of data streams, heterogeneous devices, and networks involved causes high traffic and affects the overall latency [2].

One prominent approach that has emerged in recent years suggests utilizing distributed computation entities (i.e., perceived as edge devices) in proximity to end users, respectively at the edge of the networks [3]. Edge devices available in the vicinity of end users can be leveraged to process various data or workloads instead, and as such, the edge emerges as a central architectural entity. As a result, the key theme is that these processing elements in proximity to end users first aim to avoid user-perceived latency, reduce the need to transfer data to the cloud, and improve privacy by analyzing released information by users. The latter plays a core role in protecting and securing user data that is released without users' consciousness or information that can violate privacy policy requirements defined by a stakeholder (e.g., company, school, etc.). In this context, edge devices and, in general, edge architectures play an important role because they support network affairs and improve overall privacy protection. For instance, edge-based infrastructures provide a seamless opportunity for accelerating the development of data-centric tasks such as within crowdsensing platforms [4]. In crowd-based tasks, edge devices can act as intermediary entities to protect released information and enforce various privacy strategies to make

sure that sensitive data (i.e., user information, sensory data streams, etc.) is not released. Therefore, we need an edge-based mechanism to prevent sensitive information from being released and to ensure that third parties cannot identify individuals without their consent.

Edge-based infrastructures are heterogeneous environments with various devices featuring different capabilities, such as dedicated edge servers, network routers, telecommunication stations, or simple edge gateways that aggregate sensor data in healthcare or smart city environments. Because of the heterogeneous nature of edge-based infrastructures, we require a standard format for how data has to be transformed to ensure privacy; more precisely, one specification serves for all possible types of edge devices. Specifically, we refer to transformations as modifications of data or its metadata, an operation that arbitrarily combines or discards information. Transformations are derived from privacy policies, a set of rules on how to transform data to ensure the privacy of the data once it has passed through the edge device. In particular, we would eliminate the need to implement a policy anew for each device type by providing a universal environment on these devices that supports desired transformations. For instance, an IoT device captures personal or confidential data, which is forwarded to an arbitrary device on the network's edge, where the data stream is transformed. We might choose any edge device here since the uniform environment will analyze and transform the data stream according to the mutual, predefined set of privacy policies.

This paper presents a privacy-enforcing framework that transforms data streams on the edge. We enforce privacy close to the data source, using powerful edge devices to perform operations that would otherwise absorb resource-limited IoT devices whose software stacks are limited. We assume that a central, trusted entity manages edge devices and the privacy policies for the entire data pipeline, ensuring exactly how and when privacy is enforced. As soon as data streams flow through an edge device that supports such transformations, we have complete control over the information flow in our system. Whenever an IoT device captures personal or confidential data, an edge gateway in the device's vicinity analyzes and transforms the data stream according to a predefined set of rules. Such rules build up to privacy models, a representation of their privacy policies. Privacy models consist of a set of triggers and transformations that describe how data must be modified before it can be released to stream subscribers.

1.2 Problem Statement

The most intuitive examples of areas that can benefit from such applied privacy models are those within surveillance environments. Specifically, private information might be recorded unintentionally just because a person or object was present in the area at the time the data was collected. For example, we can imagine an office building where the meeting room has a camera for security reasons. It might not be desirable to capture the content of a given presentation or laptop screen; otherwise, confidential information might be leaked by somebody who gets hold of the live stream or a recording. The exact rules and restrictions for privacy can be extracted from the company's policies

that they have agreed on within their enterprise; those will be translated into a privacy model on how to transform data before it can be released to the public or recorded internally. Transforming written agreements directly into privacy models is itself an ongoing challenge, as described in [5] and [6], so we stick to manual conversion here. Once a company has come up with a model representing their privacy policies, it is a rule that laptop screens are blurred out in every video frame. This has to be ensured before anybody can consume the video stream. According to [6], a standard camera installed in the office cannot directly perform this video transformation. This task will be performed in the device's vicinity, i.e., on the network edge, before directly streaming the video to recipients.

Edge gateways provide sufficient computational resources to perform stream transformations close to the data source; this is especially important with the emergence of GPU-accelerated single-board computers. For instance, the NVIDIA Jetson [7], which can be installed close to IoT devices to perform fast image recognition. In Figure 1.1 it is depicted how a captured image from an IoT device is transferred to an edge device in its vicinity. The edge device transforms the data according to the rules defined in the privacy model before being streamed to an audience or being recorded internally.

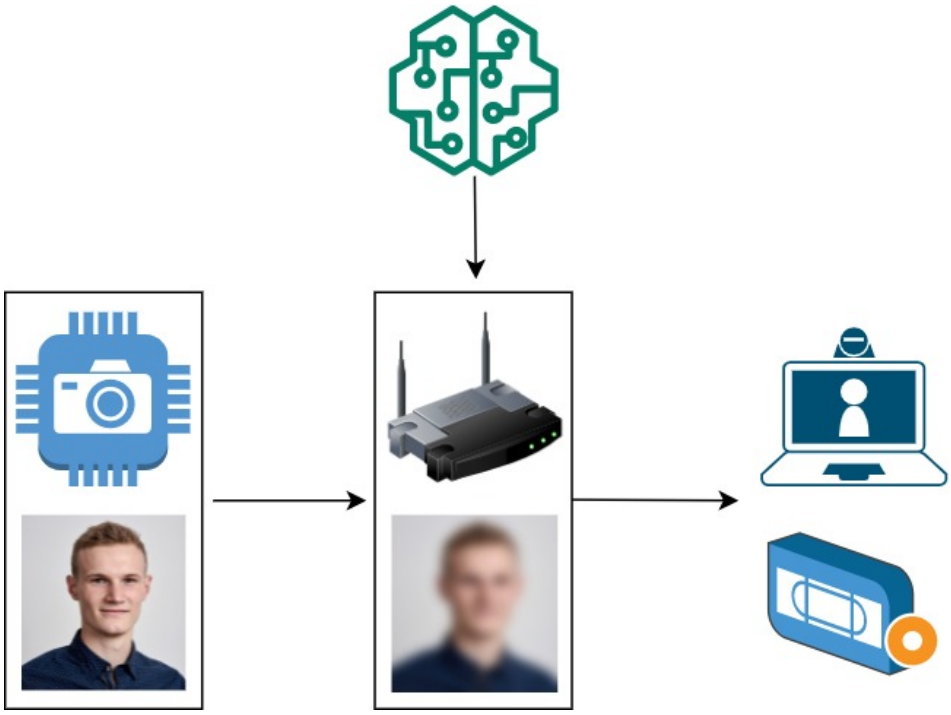

Figure 1.1: Transforming streamed data according to a privacy model.

Privacy models can describe all sorts of stream transformations; some of them might seem unrealistic from a state-of-the-art perspective because they can hardly be supported

with current technologies. For instance, detecting who is speaking and removing a certain audio layer from the stream, but doing the same for images by detecting an individual in a video frame seems perfectly possible. Machine Learning (ML) models have already shown that they can successfully identify celebrities after being trained on a sufficient number of images [8], so this can be the case for a company's employees as well. Suppose an ML model is regularly extended with new faces or voices that have been learned. In that case, the respective privacy model might as well need to be updated, and changes propagated to edge gateways. To maintain the interoperability of heterogeneous edge gateways, these devices have to support a consistent environment that allows them to frequently (re-)deploy models and perform the described transformations on handled data streams. In the event that the model is extended with novel transformations that are not yet supported by the edge device's environment, the environment has to be upgraded equivalently on all edge devices in the network. Hence, the environment that supports the transformations can be extended arbitrarily with new transformations by a company or an agency as they see fit. Still, the structure of the model itself should not change.

Streaming transformations do not have to be limited to audio and video data. In the context of a smart city, cameras and sensor networks can capture all kinds of information about vehicle movement within the road and parking network, like their exact speed, location, and car type. Edge gateways in these infrastructures can be located in a traffic junction's vicinity or in a parking garage, where sensor data is aggregated before being transferred to a central monitoring unit. To avoid any misuse of the collected data, like tracking an individual's movement, private information (e.g., car plate) is either not captured or removed by a transformation imposed by a deployed privacy model. However, it might still be possible to track this car's movement. Supposedly, there is a car type that is not common in the area, such as an outstanding sports car. The mere appearance of this car in any traffic junction or parking area allows for precise tracking of this car because the lack of equivalent cars makes it easy to track down this one car. For static data, it is well reported how to release a tuple only if there are sufficient similar values in the data set. Specifically, this is commonly known as k-anonymity [9]. However, it is possible to perform this anonymization on streams as well [10] by maintaining a history of processed tuples and releasing a tuple only if there have been sufficient similar tuples in the recent past.

Several examples exist where edge gateways can help protecting privacy of streamed data. Therefore, some of these examples might only emerge soon, even though the gateway could already provide the most common stream analysis methods and transformations. In this sense, we advocate that the edge gateway capabilities must be easily extendable according to stakeholder needs [11]. For instance, a precise anonymization technique (i.e., blurring a face) is just an extension that an edge gateway can support. A policy manager (PM) can then arrange these transformations in the privacy model. This calls for a modular separation of the transformation functions from the edge gateway, allowing developers to single-handedly define new transformations and analysis methods that fit the company's demands without touching the gateway's source code.

1.3 Research Questions

- RQ. 1: How can privacy requirements for edge networks be specified and represented as privacy models?

  Participants in data collection tasks want their data well protected from outer access so it will not be exposed or identified if not desired. This is commonly the content of privacy policies that companies agree on with end users. Edge architectures give various benefits for collecting data from IoT devices, but to enforce these policies from an edge perspective, they need to be represented as a privacy model that can be executed on edge devices. Here, the question is how to specify such a privacy model so that the privacy requirements of end users and enterprises are reflected while providing sufficient flexibility to express multiple user groups, data types, and transformation rules.

- RQ. 2: What architecture is the most efficient for the distribution execution of privacy models?

  From a functional perspective, the edge-based system is required to enforce the model's privacy policies on all tuples of the ongoing data stream, but the question is where and how exactly in the edge layer to perform the transformations and how to design the communication with IoT devices and central managing units. Chains of policies and time-consuming transformations might considerably slow down the throughput of the data stream. Therefore, the architecture itself must be as lightweight as possible and react quickly to changes in the distributed privacy model.

- RQ. 3: Is the presented architecture in fact able to transform a data stream according to a privacy model within a respective time span?

  After presenting an appropriate architecture, we will implement the framework in a rudimentary prototype and evaluate its performance. We do not set any benchmarks that the solution must fulfill, but want to investigate if the given prototype could in fact be applied as it is in the evaluated scenario. The decisive point will be whether the processing of privacy models is feasible at the same frequency as the data is incoming to the system. The focus will be on creating a notation of performance and transformation within the time frame and not optimizing the solution to make it fit the time frame, assuming that it fails to do so.

1.4 Structure

The rest of the paper is structured as follows: Chapter 2 provides essential background information that helps the reader to understand the context in which the thesis is embedded. In Chapter 3 we consider related work and outline the novel aspects of our work. Our main contribution is included in Chapter 4, where we first present an abstract concept of how to transform streams on the edge according to deployed privacy models, and then provide our implementation for video stream processing. The given prototype will be evaluated according to a set of privacy models and stream resolutions, Chapter 5 described the evaluated scenario, Chapter 6 the respective results that were measured, and Chapter 7 our interpretation of the results. We conclude our work in Chapter 8, where we furthermore answer the proposed research questions and outline future work directions.

CHAPTER 2

Background

By now we have presented a motivating example of the shortcomings in existing research and where in this domain we embed our work. We dedicate this chapter to introducing the fundamental concepts of the related research areas. For readers who are not familiar with these concepts, we recommend attending to the following sections before moving on. Nevertheless, readers with profound knowledge of these technical principles may skip this chapter since it will not contain novel information for them.

We will first discuss in Section 2.1 the principles of edge computing and how its network architecture varies from common cloud scenarios. Afterwards, we dive in Section 2.2 into streaming technologies and the conceptual difference to message brokers, in this context we will discuss the means that exist for transforming streams between the provider and recipient. Finally, we discuss in Section 2.3 common principles in the definition of data privacy agreements and how privacy is enforced in software engineering processes.

2.1 Edge Computing

The convergence of wireless networks, commodity sensors, and embedded systems has led to the evolution of the IoT. This technical innovation has led to an exponentially growing number of IoT devices, including connected vehicles, home automation systems, wearable devices, and smart city sensors. However, the overwhelming amount of data produced by these devices needs to be analyzed, stored, and protected, which has brought cloud architectures to their limits [12].

Traditionally, data is transferred to cloud servers for processing, where it is fairly easy to maintain and secure stored data. However, there are significant challenges within this approach, namely that devices with limited network capacities face slow responses from the cloud, whereas resource-constrained IoT devices need to access cloud facilities for larger storage and computation resources. To overcome these challenges,

cloud computing is experiencing a key move away from centralized resources towards distributed, decentralized architectures, called edge computing. The data processing units in edge computing are placed within the client's vicinity, on the edge of the network. As a result, the network latency is decreased and resource-restricted IoT devices have powerful edge environments available to outsource processing [6], [13].

The emergence of edge computing will not replace cloud computing, the two should rather complement each other to accelerate the digital transformation of the industry to a greater extent. Data that is handled by edge nodes still needs to be aggregated in the cloud to achieve in-depth analysis and obtain meaningful analysis results. The role of edge computing is thus to share the pressure of the cloud and take charge of tasks within the scope of the edge. A promising scenario of the two could be that cloud computing is concerned with big data analysis and output, passed to the edge side, and then processed and executed by edge computing [14].

Edge computing can be seen as an extension of cloud computing, but in fact, both have their own characteristics: The cloud can process a large amount of data and conduct in-depth analysis, thus grasping the "whole image", and it also plays an important role in non-real-time data processing, such as business decision-making. Edge computing, on the other hand, is limited to locally available data and operates on a small scale for intelligent real-time analysis. Thus, cloud computing is more practical for large-scale centralized data processing, whereas edge computing is more appropriate for small-scale intelligent analysis and local services. In terms of network resources, edge computing is responsible for handling data closer to its source. Therefore, data can be stored and processed at the local network edge without transferring the entire amount of IoT-generated data to the cloud. The reduction of the network burden greatly improves the utilization efficiency of the network bandwidth [14].

2.2 Data Streams

A data stream is a theoretically unlimited series of events generated by a source node, based on one or more of its attached data sources. To interpret a sensed value v_i correctly, we need the timestamp t_i of the observation and potentially, other meta information such as the device identifier or location. Informally, a sensor data stream is a continuously updating relation to which we can append tuples created by a sensor, i.e. the time series of updates of a sensor. A sensor stream describes thus a history of sensed values to which we can only append values. Once an event is senses, it cannot be erased from history. Within one stream, all sensors have the exact same schema, i.e. list of attributes [15].

When streaming media that originates from a camera or microphone, data is usually in a raw and uncompressed format. Encoding is the first step to reducing the file size to empower low-latency streams. Once an IoT device pulls the data from the webcam or microphone, it first gets encoded with a supported media codec; then it splits the information into packets that can be transferred to the stream recipients. There exist numerous frameworks for live-streaming of data, among it WebRTC [16], which will

be relevant later in this paper. The fundamental protocol that powers WebRTC is UDP, which does not ensure packet delivery or that the packets will arrive in order. Nevertheless, it is a suitable delivery protocol for live-streaming because it puts emphasis on the speed of transmission. In a TCP scenario we would resend a package that could not be transmitted correctly. However, in a live-streaming scenario it can be the case that a mislaid frame is no more required at a later stage. This can be different for other streaming solutions, like video-on-demand, where a lost video frame may, for example, serve as a movie's stream buffer [17].

Figure 2.1 gives an overview of the protocols that are used in the individual network layers for streaming data with WebRTC. The protocols that are responsible for establishing peer-to-peer connections are ICE, STUN, and TURN. DTLS is used to secure and encrypt transfers between participants. Finally, SCTP and SRTP are used to multiplex the streams and provide both congestion and flow control. On the contrary, WebSockets are built upon TCP and designed mainly for client-server communication [17].

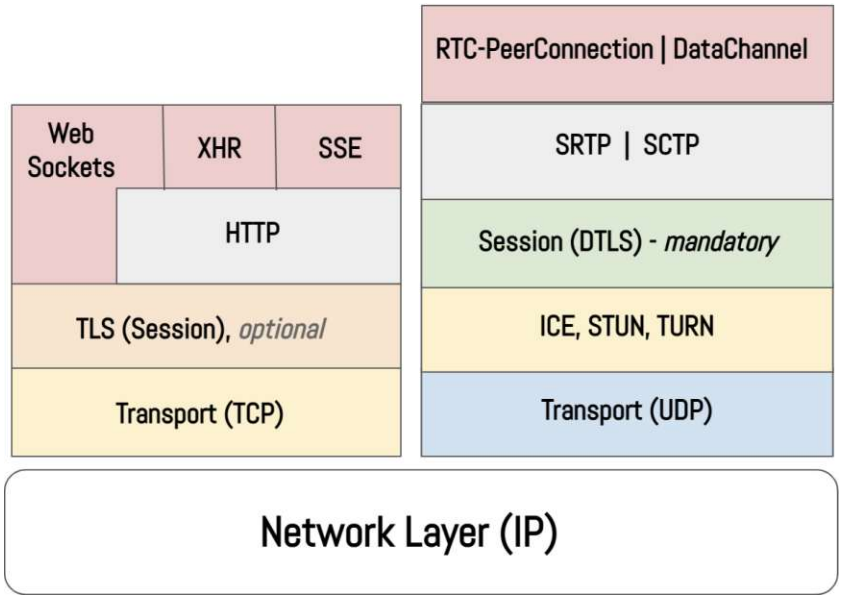

Figure 2.1: Network layers in a WebRTC live-streaming setup [17]

2.2.1 Stream Processing

If we wish to process or transform the content of the stream while keeping its nature, [18] presented 4 aspects that need to be respected: reactivity, continuity, adaptability, and efficiency. *Reactivity* means that the processing of the data item is started as soon as the sensing event occurred. *Continuity* requires a periodic recomputation of streaming applications to adapt to the context of the stream. *Adaptability* allows undergoing modifications of data processing plans as an answer to external events,

e.g. rapidly changing frequency of the stream or congestion of the internal processing queue. *Efficiency* means that data stream processing rates should be higher than data transmission rates. Stream tuples should thus be processed on time before a new tuple comes in, however, this is just an overall requirement, we may as well process tuples in batches.

Transferring big data as streams is itself a challenge because it requires breaking down massive amounts of information into chunks that can be processed continuously. The result can either be organized into systems like real-time data warehousing that deliver results based on numerous factors, or act directly based on a single event. Stream processing may include a variety of steps, among them data cleaning, query processing, stream-stream joining, stream-disk joining, data transformation, etc. A broad category of approaches, tools, and technologies have been developed so far to overcome the given challenges of stream processing. As an additional requirement, processed output must be provided with low latency, limited resources, accuracy, and within fractions of a seconds to make real-time reactions and decisions possible [19].

Applications such as medical devices, e-commerce systems, or trajectory trackers are examples of unbounded streams of data. Due to privacy regulations, these streams may need to be anonymized before they can be released for public usage. The process of anonymizing streams is different from traditional static anonymization in the sense that it is a continuous process that cannot be performed using a single full scan. To satisfy the principles of stream processing, each stream tuple must be published with a minimum delay once it fulfills the privacy requirements. A challenge in processing streams is the non-stationary nature of streams, where the change in the distribution of an attribute may cause a drift in the concept of the data. The result can be increasingly imprecise if we do not adapt the stream transformation process accordingly. A common strategy to anonymize streams and achieve the privacy goal of unlinkability (see Section 2.3) is to perturb the stream tuples by adding naive noise or applying the Laplace mechanism [9]. This concept is more commonly known as differential privacy, which aims to break the connection between data source and the data by shifting individual values slightly while maintaining indifferent mean values [20].

2.3 Privacy Protection

Privacy protection goals play an increasingly important role in ensuring information security regarding concepts or implementations of data processing systems.. The classic triad of confidentiality, integrity and availability – already used in the early 1980s – has endured decades of debates on security requirements: *Confidentiality* means that unauthorized access to information or systems is prevented. *Integrity* means that information or systems are protected from unauthorized or corrupting modifications. *Availability* means that information or systems are available exactly when needed [21].

In the context of privacy preservation, three additional goals, accompanying the principal security goals (confidentiality, integrity, and availability), have been advertised:

unlinkability, transparency, and intervenability. The goal of *Unlinkability* is to separate privacy-sensitive data from the rest of the data. It should be impossible, or at least infeasible, to find a connection between the privacy-relevant data and any other data not directly belonging to that set of data. This also implies that the privacy-sensitive data should not be usable in any context other than the one for which it was collected, thus tying the data to a specific purpose [22].

Even though data was seemingly anonymized, the richness of the data leads to "naming" an individual by a sometimes surprising collection of fields, or attributes [20]: popular combinations in this context are zip code, date of birth, and sex, or even the names of three movies and the approximate dates on which an individual watched these movies. This "naming" capability can be used in linkage attacks to match "anonymized" records with non-anonymized ones in a different dataset. By these means, the medical records of the governor of Massachusetts were identified by matching anonymized medical encounter data with (publicly available) voter registration records. These datasets in combination with a collection of anonymized viewing history records published by Netflix were sufficient to identify the governor; so even though data seams unlinkable in one context, by merging multiple datasets it can still be possible to draw conclusions about individuals.

Transparency enables involved parties to understand how privacy-relevant data is treated at any point in time and to reconstruct this information at will. Thus, there is no blackboxing allowed in how data is processed, as this would result in some of the parties being unable to understand how data is handled. One option to provide transparency is making the source code publicly available. However, the availability of source code demands that the end-user is able to understand the source code, which makes it an unrealistic method of providing transparency depending on the clients' knowledge.
Intervenability empowers all involved parties to interfere with the data processing and revoke their permission to process private data. This comprises system adjustments and countermeasures in the event that any part in the system does not operate as expected. Examples for such means are the ability for end-users to stop a running process to avoid further exposure of private information or allow investigation [22].

In the context of data privacy, it is necessary to distinguish between privacy protection goals and the measures and mechanisms that are used to achieve them. For example, pseudonymity is a mechanism to achieve the privacy goal of unlinkability, but there can be multiple approaches to ensure the same goal. Privacy agreements may equally contain abstract goals (e.g. must give evidence how private data is processed) and leave the implementational details to the developing team and stakeholders [23].

2.3.1 Definition of Privacy Policies

In order to fulfill the presented privacy goals, privacy requirements are used. These refine the goals by forming rules that directly apply to data or services. The privacy requirements are combined into a privacy policy, which contains all the rules that apply while handling the data or using the service. Privacy policies are the standard approach

used by service providers on the web. They consist of statements describing how the data provided by the end-user is handled and what kind of data is collected. Each statement contains obligations that the service provider has committed to fulfill. Statements further contains an action stating what type of processing is restricted by the statement. Every action is combined with a purpose, that defines whether the processing of the data is needed for service provisioning or whether it is done for marketing or analysis. Privacy policies are provided in textual form and are written in a language that includes legal terminologies, making them difficult to understand for end users who are not familiar with these areas [24].

Privacy policies and their content are themselves subject to regulations, e.g. the General Data Protection Regulation (GDPR) [25] of the European Union. The GDPR defines a set of rules that have to be followed by service providers; a major rule is to inform end-users about how their data is handled, which is done by defining privacy policies. Figure 2.2 provides a conceptual overview of the actuators that are included in the negotiation of privacy policies: The *Legislator* restricts the content of the privacy policy that the *Service Provider* has defined in combination with the offered *Service*. *End-Users* accept these policies to gain access to the *Service*, however, their privacy requirements might not be fully satisfied by the given policies and thus lead into inconsistencies [23].

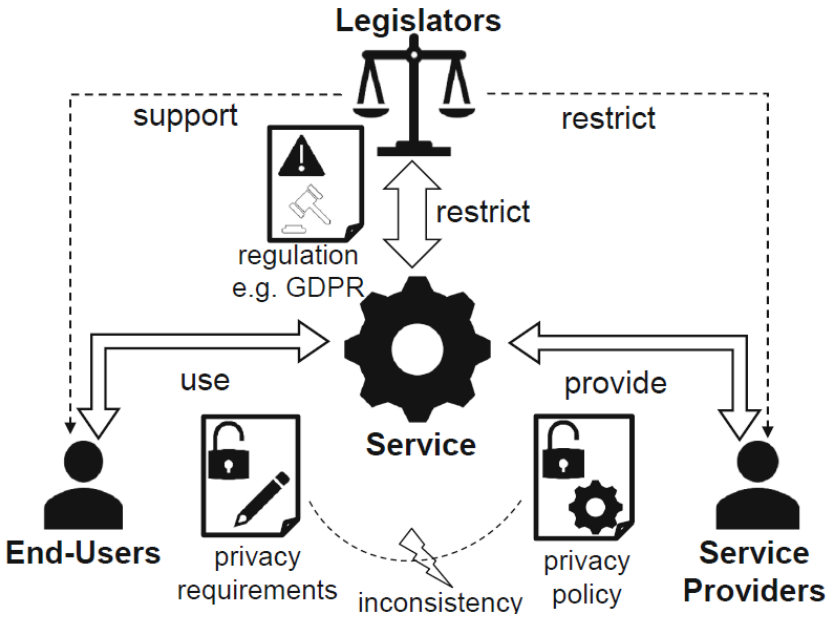

Figure 2.2: Concept of privacy regulations for services providers and end-users [23]

Sticky policies represent a special type of privacy policy in this context that uses a different approach than policies defined by service providers. Instead of leaving the decision and obligation of defining privacy policies to the service provider, end-users are required to define policies themselves for each set of data they provide to the service. Sticky

policies are thus data-oriented, whereas standard privacy policies are service-oriented. Data-orientation is achieved by sticking the policy to the data itself, this leads to the benefit that all parties involved in handling or storing the data can access the policy and are aware with how the data has to be treated [23].

Privacy requirements are not limited to textual representation; they can alternatively be given through privacy models, which are graphical or visual representations that describe the problem and the regulations that apply for treating data in the associated systems [26]. Requirement models, which include privacy models by definition, are not limited to a graphical representation; they can be encoded or described in a specific modeling language. The resulting models can be arbitrarily converted between their graphical representation and their encoded version. They provide a useful extension by visualizing how data is to be treated, which can improve the transparency of privacy agreements for end-users [27].

CHAPTER 3

Related Work

After discussing the required technical preliminaries in Chapter 2, we will now focus with more detail on the results that other researchers have already documented in the context of this thesis. We present their work on one hand to provide the reader with information about ongoing research topics in this field, but on the other hand to discuss their shortcomings and highlight the novel aspects of our work.

We will first focus on related topics on transforming streams on the edge, and then narrow down the required aspects of transforming privacy agreements to a machine-readable format. All the work presented in this chapter is merely outlined; additional information can be obtained through the provided references.

3.1 Stream Transformations

Anonymizing streaming data is one measure to protect the privacy of those who contribute the data. To that extent, the authors in [28] propose data collection schemes for IoT sensors that do not give evidence of the data source. By doing so it obfuscates the connection of an IoT sensor and the measured data; nevertheless, removing private information from the data is still a remaining challenge. In [29], the authors presented an edge-based system that removes private attributes from sensor data, in their work they propose a ML model that identifies which attributes to erase in the transformation step. However, they did not extend their approach to support advanced privacy policies besides single attributes that are to be removed. This is similar to the work of [30], 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 on role-based access schemes, where policies represented precise rules which role can consume what type of data. This approach seamlessly does not require any moderating entity in the architecture, but is very restricted to the type of privacy policies that it can represent.

A fundamental problem when enforcing differential privacy on streams is that data is non-stationary, meaning that structural attributes might evolve over time until privacy violations cannot be detected anymore. To that extent, the authors in [9] discuss how to monitor stationary shifts in the data and adjust the data release process accordingly. In our setup we assume that such a shift is not possible, otherwise we would need a solution like theirs to cope with the resulting issues. Their solution would ensure k-anonymity on the resulting data, which is also the content of [10]: 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 ensure an applicable latency, however, they would not consider stationary shifts in their work.

3.1.1 Transformations on the Edge

The authors in [6] discuss how role-based access control schemes can be expressed through privacy models, where they implemented an edge-based system to consume data over a message broker after anonymization. Their privacy models focused on a variety of access control restrictions combined with token-based authorization. Their work is clearly the one that contributes the most towards a complete framework for privacy enforcement, in Figure 3.1 we depicted their approach, which included IoT devices that are registered by organizations, which on the other hand define privacy policies. Policies are stores in a secure vault and provided through the database to enforce privacy on the computation nodes. However, they did not extend their approach for continuous data streams and focused on authorization of entities and role-based access schemes for data instead of transforming streams according to more advanced privacy policies.

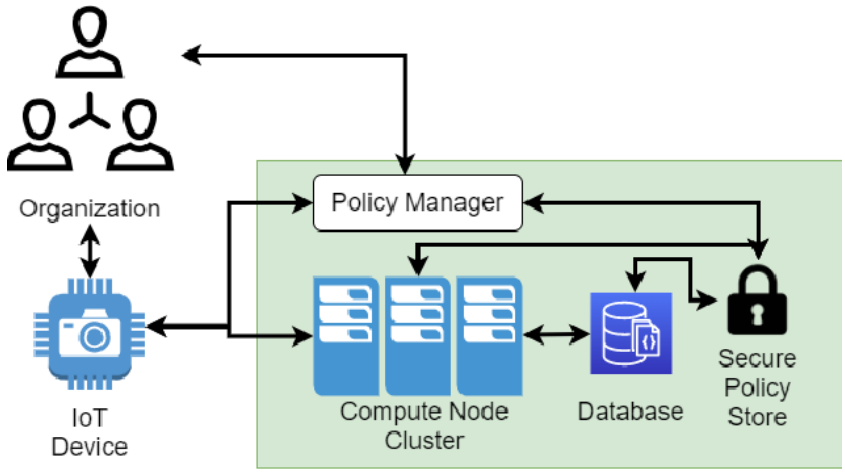

Figure 3.1: Access-control restrictions for sensor data on the edge (adapted from [6])

The authors in [31] describe how differential privacy can be improved through edge networks: They enforce privacy within the edge layer to reduce the amount of data that is transferred to the cloud; in case that an attacked gains access to the cloud systems, the damage is also decreases because the accessible data has already been transformed. More generally, in [32], the authors discuss privacy considerations for edge networks, including stream processing and anonymization techniques that gain momentum by the availability of edge resources. Their work provides an extensive evaluation of architectural manifestations by which they declare the edge a first-class entity for privacy protection. They also discuss how federated learning protects privacy by maintaining training data in the edge level, a thought that [33] pursue for image recognition with deep learning models. Furthermore, the authors assess the advantages of edge networks for ML and how a model can be trained and extended on the go, without involving a dedicated training phase. Still, the low latency in edge computing is also advantageous for numerous feedback mechanisms.
Because ML can be done on the network edge, according to [34], it might not even be necessary to have a full view of the data on the cloud side, hence, data that reaches the cloud might already be anonymized or filtered for public availability. An example for different roles that do not necessarily need to have a identical view of the data, certain data might not be desired in the cloud due to its increasing size and the already conducted learning phase on the edge. We do not yet include role-based consumption of streamed data in our framework, but we consider it a useful extension to provide different views of the data whenever required by a privacy agreement.

Especially important in the context of this paper, the authors in [35] present how video streams can be transformed in edge networks. If the latency of the received stream drops due to increased processing on the edge device, they scale up to several stream transformation workers. Maintaining a low and stable latency by such means is definitely a useful extension for stream transformations scenarios. Still, it requires an unavoidable overhead in communication and the means to scale edge devices horizontally. In our work we limit ourselves to the processing of streams on a single edge device without the possibility to outsource computation, however, this might provide a powerful extension through future research.

To underline that edge technologies are advancing, [36] have discussed the efficiency of using edge computing environments for data collection in IoT systems, which already leads towards the problem at hands because they point out that sensor data from IoT devices need to be protected from undesired access. Together with [14] they advertise edge computing facilities for latency-aware tasks. They give a broad outlook to fields that will benefit from processing IoT data at the edge without involving any central cloud server. As a minor disclaimer, all of them will need to be secured according to privacy requirements. In their existing work [37], they had already discussed the advantages of edge computing for collecting geospatial data, which is related to numerous papers in the context of data collection in crowdsensing [38]–[42]. There is a couple of open questions on privacy issues [14], because edge servers as new entities in networks still

need to be regulated. In this context, crowdsensing systems are a often mentioned topic [43] regarding how to collect data anonymously to protect the privacy of those who contribute data.

Privacy is a highly valued non-functional requirement for those who contribute data, something that became evident again through the COVID-19 pandemic, when a lot of countries were struggling to convince citizens to participate in contact tracing. According to [44] and [45], this was caused by the shortcoming that citizens did not feel their privacy protected well enough because secure collection schemes involving smartphones and edge facilities were only recently gaining momentum in development. Their work is dedicated to the problem of anonymous data collection, but also to continuously ensure privacy once data leaves the edge layer. Together with [46] and [47], they frequently discuss the importance of blockchains to ensure integrity of results and preserve privacy of data contributors. Our framework could also benefit from blockchain technologies for a multitude of security aspects, one of the secure storage of privacy models.

It is thus notable that there is a lot of active research on crowdsensing systems and how to ensure privacy within the streamed data, one of them is discussed by [48]: how to maintain confidence in the devices that provide data. This is also a challenge within our approach that we do not yet address at the current state, but it will be in our favor to prevent malicious devices from streaming data to our edge device. The author in [49] further discussed how to establish a reputation-evaluating system that maintains a list of trustful data providers and workers, which again contains results that can be directly related to our work.

3.2 Privacy Modelling

Numerous researchers have already advertised the network's edge for transforming data. Still, another field that this research covers, the one of how to specify privacy requirements in a privacy model, has found attention in the research of [50]. Their work was related to how it is possible to translate enterprises' policy agreements to machine-understandable file formats, a much-needed extension to our work because manual conversion of policies to privacy models is only applicable for basic use cases.
The authors in [51], and [27] also contribute in many regards towards the specification of privacy or security requirements. They investigated on a fundamental basis how it is possible to specify privacy requirements at all through modelling languages, to that extend they evaluated multiple established frameworks and compared the resulting privacy models. Their work on privacy specification can extend our approach in future work, nevertheless, it remains to provide an underlying privacy-enforcing environment that supports the transformations imposed by the privacy models.

Following the work of [23], end consumers suffer from a common "take-it-or-leave-it attitude", which either forces them to accept a given privacy agreement or find themselves unable to use the service. This grows increasingly inscrutable as privacy agreements are growing, not least due to extensive aspects that need to be included in the agreement

due to privacy standards like the GDPR. They recommend turning the tables and specify from a client's perspective how we would like our data to be secured and let services consume it by these regulations. This concept aligns closely to the one by [52], which puts the end user in charge of defining user-centric, sticky policies. However, the questions is if end users are actually committed to dedicate time to secure their data instead of relying on common standards. Adopting this concept to our approach, a client could provide the privacy model how the contributed data should be treated and the data stream would be transformed accordingly on the SFU.

To assist developers in the creation of privacy-preserving software, the authors in [53] focused on the interpretation of business data flow diagrams to identify vulnerable spots in applications. Resulting software would by design undertake more frequent privacy-checks and remove the burden from software engineers to be accountable for privacy aspects. Their work is lacking further evaluation for larger business processes, but provides an interesting approach of deriving privacy restrictions from data flow diagrams instead of privacy agreements.

3.3 Remaining Challenges

Through the provided research literature we showed that the edge has found a lot of attention for various tasks, including the enforcement of privacy policies. Up to now, research shows that transformations on the edge were dedicated primarily to removing certain attributes from streamed data or preventing the release of frames due to missing differential privacy. Nevertheless, it remains a step towards a general model to specify policies and enforce a variety of transformations expressed by the introduced model. As mentioned above, several research works exist on specifying privacy requirements for enterprises. However, these works focus on business processes and do not consider low levels aspects such as removing privacy-violating content in continuous data streams, and this is where the two areas diverge notably.

So as presented in the given literature, there is both existing publications on performing transformations on data streams in edge networks and publications on specifying privacy requirements in "machine language". This work will bridge the gap between those two areas, proposing a novel scheme on how privacy models, defined in a certain language, can help to secure privacy on the streamed data of managed edge devices by enforcing privacy policies.

CHAPTER 4

The Framework

In Section 1.2, we discussed the existing gap in research and where this thesis is located. We highlighted scenarios that will benefit from continuous stream transformations, where the methods for analyzing and transforming data are expressed in privacy models, which represent a set of privacy policies. In this chapter, we present in two phases our framework, which is dedicated to fulfilling these goals: Section 4.1 presents an abstract concept, where we give an overview of the capabilities and features that a solution must present to fulfill these requirements. At this point, we do not yet include any details of the implementation and focus on the fundamental aspects, among them the framework's architecture.

The rest of this chapter is dedicated to the prototype we developed as a proof of concept, our implementation of the abstract specification. We present how certain features were realized and discuss the possibilities that we have from a technical point of view when it comes to stream analysis and transformation functions. It contains valuable technical details and shows how we work with state-of-the-art technologies for stream processing. Nevertheless, the remaining chapters of this thesis do not assume that the reader is fully aware of these details; it is sufficient to know the concept. Section 4.2 discusses how data streams are provided and consumed by clients. Section 4.3 presents the possibilities we have to detect patterns in continuous streams, Section 4.4 is dedicated to how we specify privacy policies in privacy models, and finally Section 4.5 gives evidence of how we deploy privacy models on edge devices.

4.1 Concept

There are three significant parts of the proposed framework that need to be addressed: first, the structure of the privacy model and how rules and transformations are expressed. Afterwards, one should note how these models should be executed on the streamed data. And third, the structure of the edge network. Specifically, one should take note of how

data is streamed between an IoT device and an edge gateway. Furthermore, an interesting point is how the output data can be consumed after assuring that all privacy policies have been respected.

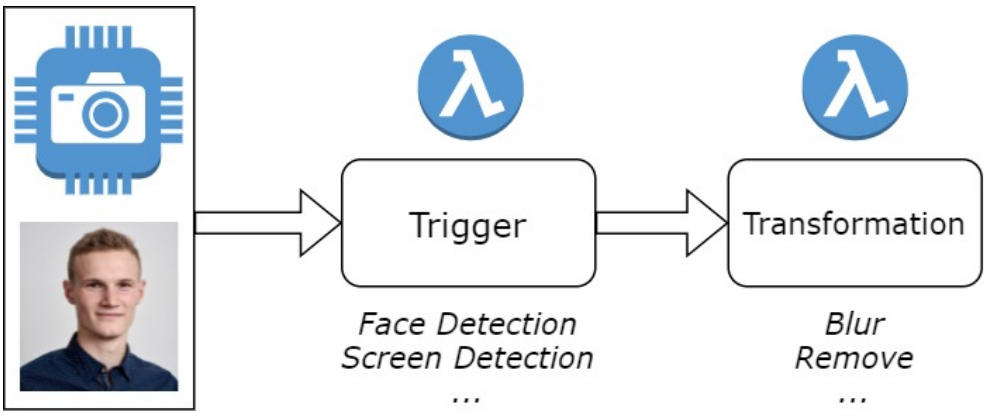

Figure 4.1: Defining triggers and transformations for the privacy model.

In a streaming scenario, the most natural representation of how a privacy model can be expressed is a flow diagram, expressing how data is step-by-step analyzed and then transformed in case a privacy-violating pattern was detected in the streamed data. Figure 4.1 shows how we define triggers and transformations on an image; such cause-and-effect rules can be chained so that one rule only applies once another has come true. The privacy model builds up to a set of such rules that are expressed in an acyclic graph, where the next step can always be determined by following the flow towards its end. The streamed data type already restricts what triggers and transformations are possibly defined in the privacy model, e.g., image recognition with a subsequent replacement does not make any sense if the stream contain car plates paired with locations. To assure that the model actually matches a valid grammar, we need to execute a model compiler on the edge device before putting the model into force.

Once the edge gateway starts receiving data over the streaming connection, it becomes evident what type of data is being transferred to the gateway. However, there might be certain rules that should only apply to certain data subtypes, like transformations that should solely be executed for road scenes and not for office cameras. In these cases, the stream's metadata can include dedicated tags so that the gateway is aware which of the available privacy models to apply. If we imagine that every frame from an office camera would be automatically scanned for car plates as well, this would also affect the latency of the stream negatively since the analysis consumes more time.

Some triggers and transformations do not require a state, like static analysis of images containing a specific pattern. These operations can easily be expressed as stateless lambda functions that are chained together in the aforementioned acyclic graph, passing the result on to the next step before eventually returning the result to the stream subscribers.

However, in some cases, we would need to keep a state similar to the example with k-anonymity in Section 1.2. In these cases, the environment has to supply temporary storage that lambda functions can address for such matters. AWS lambdas [54], and NVIDIA Deep Streams [55] are examples of how such functions can be combined by dragging and dropping them through a UI. This approach supposedly does not require any programming skills from the person responsible for defining the privacy model. The model graph itself is defined in a central cloud application from where it is deployed to all edge gateways. After compilation, the model is active and incoming data streams are analyzed and transformed according to the defined criteria. Lambda functions and the privacy model are maintained in distinct modules, separated from the stream processing. This allows the gateway to continuously transform data streams without interrupting active peer connections, also while receiving a new privacy model or lambda functions. Updates to the privacy model and the lambda functions must be as lightweight as possible to not impact the device and network performance, maintaining a stable stream latency.

Edge gateways must not only handle in- and outgoing data streams but also decode stream data and re-encode it after performing the transformations imposed by the privacy model. WebRTC [16] is primarily a protocol for continuous streaming of videos and other data between two peers. However, one peer can be extended with routing functionalities to receive and forward data streams between multiple clients; this role is called a Selective Forwarding Unit (SFU) and is assumed by the edge gateway. Figure 4.2 contains all the major parts of the architecture:

1. A cloud application, where a PM can define new lambda functions and policy models, which communicates with the edge gateways through their exposed REST endpoint for configuration.

2. Multiple IoT devices that stream data to the edge gateway through a supported protocol, including but not limited to WebRTC. If there were a serial connection to the gateway, it could use that for transfer.

3. Different types of stream consumers, which can be consumer devices or even recorders. Regardless of their purpose, they must all support the WebRTC protocol.

4. The core of the topology is the SFU, which is deployed on the edge gateway. It receives new models and functions through a REST endpoint without interrupting the ongoing stream connection. Incoming stream packages are decoded and transformed according to an active privacy model before being streamed to connected peers.

The SFU establishes connections to producer and consumer devices through a dedicated Session Description Protocol (SDP), which contains information about the session and the data type to be transferred. Once both peers have agreed on the content of the stream, they establish a peer-to-peer connection. WebRTC provides more than one channel to

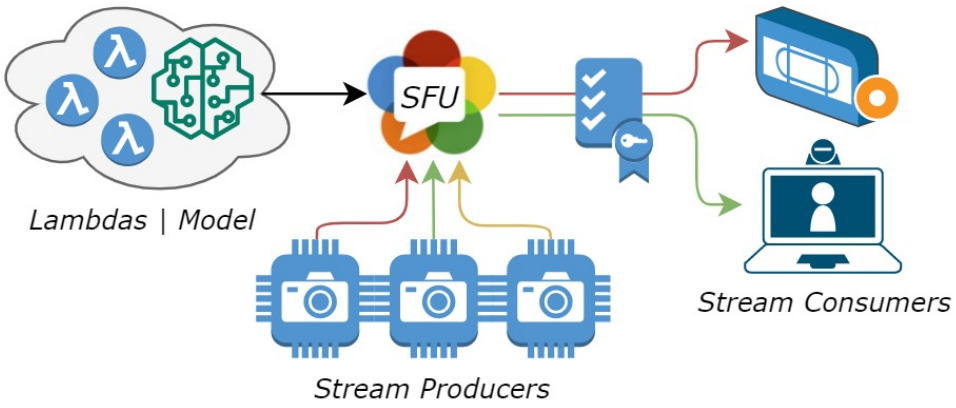

Figure 4.2: Conceptual framework for transforming streams with privacy models[1]

communicate between peers through a given connection [16]: a media stream that itself consists of two encoded tracks (audio and video) that do not need to have a relationship (i.e., originate from the same video file), and the data channel, a bidirectional connection that can exchange any data encoded in text or byte arrays.
In this framework, data is always transferred and analyzed in frames; this is the same for audio, video, or other streamed data. This means that privacy violations that would only emerge from analyzing multiple consecutive frames at once cannot be detected without storing intermediary results. The SFU waits for incoming frames on each channel, which are then analyzed and transformed according to the privacy model. Depending on the type of data that arrives, the SFU's environment has to provide different analysis methods. For instance, if we want to stream video frames over the SFU, we need to analyze incoming frames with a suitable environment that supports video operations, like OpenCV [56]. Through our privacy model we can then express a function that detects privacy violating patterns in the video frame (trigger). Once we detect e.g. a face in the video frame, the respective area is blurred (transformation) and the frame is routed to connected consumer devices. Extending the SFU with other environments makes it possible to analyze and transform data frames in various ways.

The bottleneck of this architecture is supposed to be the edge gateway, more precisely, decoding and encoding the packages and running the model's functions on the packages. As the chain of trigger and transformation function grows (see Figure 4.1 for reference), the latency of the stream grows equally. This requires monitoring to give feedback on how long the chain of transformations and the en-/decoding process takes. One mechanism that can be implemented to cope with increasing or unstable latency is an adaptive bitrate, regulating the quality of a video stream according to the network quality and the time required to process frames. In other words, if the network quality is low, the audio/video bitrate is decreased by the SFU and the IoT devices.

[1] WebRTC icon obtained from https://webrtc.org/ (visited on March 3rd, 2022)

Streamed data is encrypted by default when using WebRTC [57]. Essentially, WebRTC communicates through Transport Layer Security (TLS) (recall architecture in Figure 2.1), which is widely used to secure HTTPS connections. Therefore, any transferred stream is as secure as any other data sent or received by a web browser. Since WebRTC is a peer-to-peer connection between two agents, the data never passes through web or application servers. This drastically reduces opportunities to have the data intercepted.

4.2 Stream Routing

WebRTC itself is a protocol that has implementations in various programming environments (e.g. Python, Node.js), but for our prototype we wanted to decide on one consistent environment that supports all the features of the presented concept. We chose Python and the aiortc [58] environment for routing, not because we saw substantial advantage over Node.js implementations like the ones by mediasoup [59] or node-webrtc [60], but because of the image processing capabilities that Python provides for Section 4.3. Thus, we decided to use Python for the entire prototype and the evaluation, nevertheless, the concept can be implemented in other environments and with other frameworks as well. We did not evaluate every tool that we applied on a formal basis, but decided as we saw fit for each aspect.

4.2.1 Data Provision & Consumption

The primary responsibility of the SFU is to manage the peer connections to the provider and consumer devices, each of these clients must establish a peer connection through SDP. The provider and consumer clients can be implemented in any programming environment that supports the universal generation of SDP offers and responses - thus supports the SDP protocol. To evaluate at least two different environments we decided on a combination of them: we used the aiortc implementation [61] for the data provider, and for the consumer a native browser-based implementation [62] for WebRTC, written in JavaScript (JS).

Figure 4.3 shows how the SFU communicates with providers and consumers to establish peer connections. The SDP offers and answers contain a variety of information, among them the transport port, IP addresses, media codecs, and in which direction the data can be streamed. This information is vital, because not all browser APIs for WebRTC support all available media codecs. In our setup, the consumer can for example only connect through Google Chrome, because the WebRTC implementation utilizes beta features that are not fully available for other web browsers. A peer connection is already established after the SDP response is received from the SFU, but there is no data transferred yet. At this point, both actors are still free to choose what type of media they want to transfer over the stream; in our case, both clients are able to provide and respectively consume audio and video data from the SFU. This can also be done in parallel, which means that provider and consumer can add more than one media track for transfer. After the track has been added to the peer connection, video/audio frames start to flow, which have to

be routed by the SFU to its destination. Note that it is in both cases the producer or consumer who adds the media track, though in fact all parties can decide on which type of track to use, as soon as any part decided on the media track the peer can start streaming.

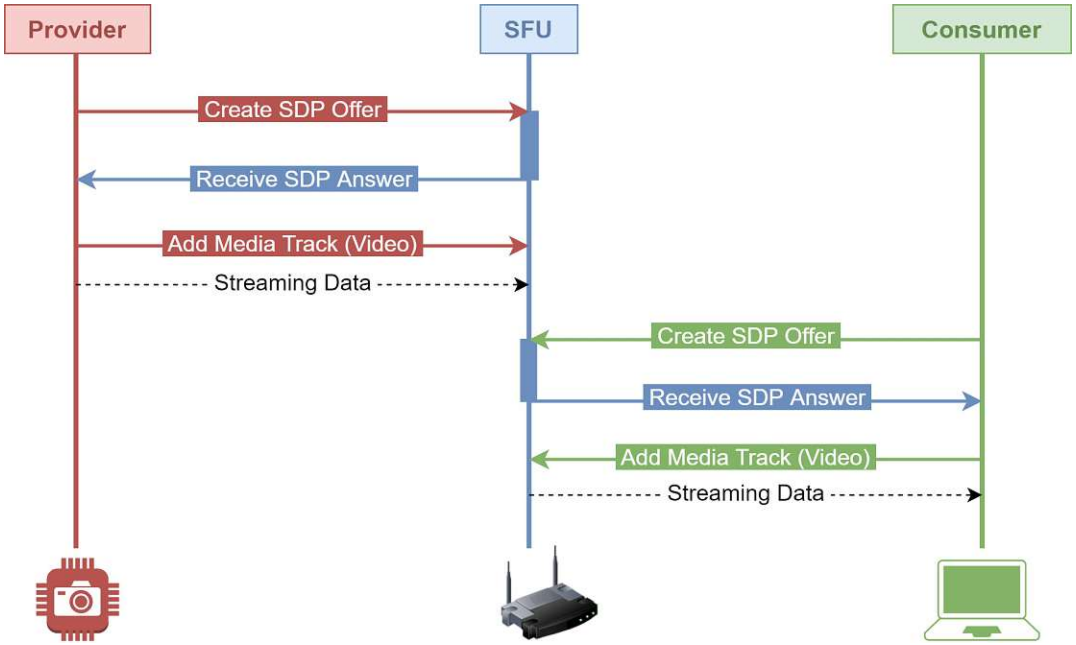

Figure 4.3: Establishing peer connections between provider, SFU, and consumer

Our implementation does not support multiple clients providing data, but it allows multiple clients to consume data of the same provider. Therefore, a streaming scenario might consist of multiple instances of consumers receiving the same data stream. To ensure that all consumers receive the data packages with the same frequency, aiortc provides a MediaRelay component [61] that consumers internally subscribe to after selecting a media track. Whenever a new frame from the provider is available, a copy of this frame is relayed to all subscribed consumers at once. By the same logic, we can execute arbitrary operations on the data frame, if we relay it later to all stream consumers we only need to perform the operations once. This is essential, because otherwise the computation load at the SFU would increase linear with the number of consumers, which would naturally happen with multiple provider streams that all need to be processed independently of each other.

The provider client we implemented can generate and stream both video and audio data, in either case, it uses Python libraries to access the local recording hardware: for audio data this is facilitated through the PyAudio library [63], and for video data a MediaPlayer implementation of aiortc [61] is used, which are actually both wrappers around the FFmpeg environment [64]. WebRTC supports a third channel by default as presented in the concept - the data channel - that can be used to stream any data

encoded as bytes, integers, etc. However, we did not utilize the data channel because we saw less challenge in transforming these types of streaming data since their processing can supposedly be done in little time compared to multimedia data. Our client can provide video data either by accessing the local webcam or replaying a video that was recorded before. For performance aspects this makes little difference if the recording has the same video resolution and frame rate in which it was recorded. However, there might be a slight difference for live webcam data, since the producer would still have to encode the data before releasing it, e.g. transform a video frame to the H.264 codec.
For both media types, FFmpeg accepts parameters to regulate the resolution and frame rate of the provided data, thus, allows tuning and optimizing the streaming throughput. For example, if the processing of video frames requires more time than available and we start to drop frames, we can decrease the resolution of the video stream or the frame rate to cope with this problem. This implies that the recording hardware connected to the client accept such parameters, which is not always guaranteed.

In fact, our provider client is a simple Python router that is operated through commands sent as HTTP requests. The interface is depicted below in Table 4.1: We can start / stop multimedia streams by connecting / disconnecting to the SFU server, which assumes that we have access to the SFU in the local network or know its global IP address. The SFU's address can be passed to the provider client on application startup, otherwise the SDP offer will fail. We can also calculate and persist stats about an ongoing peer connection, which will be important later for evaluating the latency between SFU and provider.

Method	Path	Params	Description
Post	/startVideo	live/recorded	Connect to SFU and provide video data
Post	/startAudio	{}	Connect to SFU and provide audio data
Post	/stopAll	{}	Disconnect all peer connections from SFU
Post	/calculate_stats	{}	Measure the RTT between client and SFU
Post	/persist_stats	{}	Persist all RTT to an external CSV

Table 4.1: API interface for provider client

4.3 Pattern Detection and Transformations

With a stable streaming connection between provider, SFU, and consumer, we would already like to transform the data at the SFU before relaying it, ideally by applying operations that we specify in a privacy model. However, we are still missing a couple of fundamentals: We will see soon that a transformation, like blurring a face, is actually not the most challenging part, but how to detect a pattern in an audio or video frame. In Section 4.3.1 and Section 4.3.2 we will first focus on how to detect a privacy-violating pattern within the frame, and then how to transform the frame to conform to a policy. We do not yet discuss how these triggers and transformations are expressed in a privacy

model, this will be addressed in Section 4.4, but we will have a detailed look how we process streaming data. The operations we present here will later serve for the privacy model, where we will chain them together as function, thoroughly encapsulating their implementation.

To detect patterns in streamed data, we make use of Deep Neuronal Networks (DNN), though we did not train any networks ourselves since this was not in the focus of our work. Instead, we apply existing models that other researchers have provided. Such models are traditionally created with ML frameworks like PyTorch [65] or Tensorflow [66], and by design lack interoperability between these environments. We do not want to limit ourselves to one type of model or provide multiple runtime environments to supports them, therefore, we rely on models converted to the Open Neural Network eXchange (ONNX) [67] format. A variety of models environments can be converted to the ONNX standard, which significantly increases the spectrum of models that we have available. The ONNX model zoo [68] provides a multitude of models that are free to use for educational purpose, it also shows the variety of patterns that can be detected: image or object classification, language processing, analyzing facial expressions, etc.

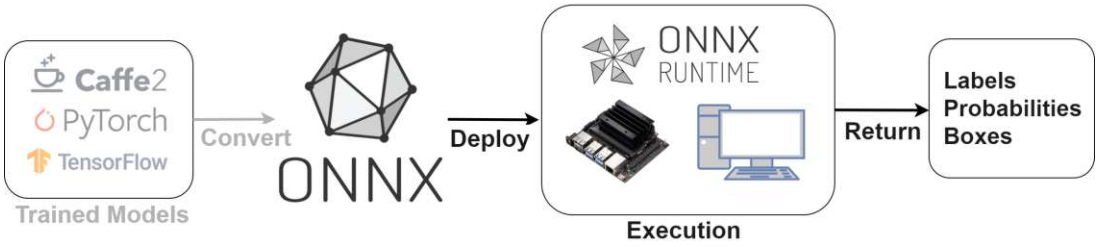

Figure 4.4: Converting and executing ONNX models for pattern detection[2]

In Figure 4.4 we have depicted how a model that was trained for example in PyTorch is converted to the ONNX standard. This model can then be executed with the ONNX runtime on edge devices like the NVIDIA Jetson or any conventional device that supports Python3. An important detail of the ONNX runtime is that it facilitates GPU-accelerated image processing on edge devices through NVIDIA Cuda® [69], which will later elevate our image processing capabilities to the next level. After execution, the model yields a set of labels, probabilities, and boxes, i.e. where (boxes) in the frame did we detect what (label) with what likelihood (probabilities). This is discussed in details for image and audio streams in the following sections, where we take a closer look how data frames are assembled and what type of operations we can apply on them.

[2]ONNX icon obtained from https://onnx.ai/, NVIDIA Jetson image obtained from https://www.nvidia.com/, TensorFlow icon obtained from https://www.tensorflow.org/, Caffe2 logo obtained from https://caffe2.ai/, PyTorch logo obtained from https://pytorch.org/ (all visited on Feb. 10th, 2022)

4.3.1 Video Streams

Videos frames that are received by the SFU contain a variety of metadata, among them image aspects (width, height, video codec) and a time base, which relates to the frame rate in which the video was recorded. A common video frame could have a resolution of 640x480 pixel and is encoded in yuv420p (H.264). Depending on these specs, certain image recognition model might not even be applicable because they require for example 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 with a precise format that the ONNX model can process. The ONNX model circus provides the accepted frame format alongside the trained models, in the below Table 4.2 we reference all the models that we used for our implementation.

Name	Description
Face Detection 320 [70]	Lightweight face detection model for edge devices
Face Detection 640 [70]	Same as above, but images as 640x480 for better results
Age Classification [71]	Returns age range (e.g. 25-32) and probability it matches
Gender Classification [71], [72]	Returns gender (male/female) and probability it matches
Car Plate Recognition [73]	Detects Vietnamese car plates in images

Table 4.2: ONNX models used for detecting privacy-violating patterns

As soon as we wrap our implementation around an ONNX model we start to call it a trigger function. With our set of triggers we can already detect a variety of privacy violations, for example a public video stream that may not contain faces of minors because a company is lacking parental approval. Once we detect a privacy violation within a video frame, e.g. a child between 8-12 years with a probability of 92%, we transform the area where this pattern was detected according to the policy.
We implemented 3 privacy-enforcing transformations that are all based on the OpenCV Python package [74], thus, directly editing the video frame data. Table 4.3 contains all transformation functions together with a short description. Notice that transformations are not limited to be executed after detecting a pattern, the Max Spec Resize can for example cap the workload for all following operations and optimize the streaming latency to the consumer.

Name	Description
Blur_Area_Pixelate [75]	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

Table 4.3: Overview of OpenCV transformation functions

Transformations are our countermeasure to ensure privacy once we detect a privacy defect. To continuously ensure privacy, we have to process every frame that is routed through the SFU before relaying it to consumers. Our implementation can easily be extended with other ONNX models and OpenCV transformations; as a company, 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 have to implement, thus maintaining a uniform function structure. This is essential for the next chapter, when we will chain together triggers and transformations that depend on a precise set of input parameters.

The presented transformation and trigger functions do not maintain any state, they represent static operations on the input data that return a set of labels, boxes, and probabilities, as shown in Figure 4.4. One limitation that emerges from this static nature is that we can only detect privacy violations that appear from a single data frame, we can not detect any movement or transition between frames. For example, if we would like to blur faces of individuals that left a store again within the first 5s, we would have to keep a state for $5 * fps$ frames. The first frame would only be release once the last 5s of frames have been processed and potentially transformed, which would naturally introduce a delay of 5s for a video surveillance stream.

It has been shown that video frames from the producer are received by the SFU with an overall stable frame rate. However, individual frames might arrive faster or slower. This becomes more evident later when we provide results on the measured latency in Figure 6.2. To re-stabilize the stream's frame rate and relay a frame exactly every $1/fps$ seconds, we implemented a frame queue that is situated between the SFU and the video transform track. Whenever a video frame is received from a producer, it is appended to a thread safe queue for retrieval by the transformation track. When a consumer stream is connected, another thread consumes the enqueued frames one by one and executes the configured trigger and transformation functions. The second thread retrieves and processes video frames according to the average provided frame rate, masking timing issues between individual frames.
The maximum length l_x of this frame queue is a decisive parameter, a high number of elements helps to cope with unstable stream latencies, but in the worst case it introduces a delay of x frames. Supposed that the trigger and transformation functions cannot be executed within a given time frame, i.e. they require more than $1000/fps$ milliseconds, the queue will start to fill up to a maximum of x frames. Every new frame that is retrieved will drop the oldest unprocessed frame before queuing up. As for now, the second thread that executes the functions will always come upon a queue with x elements it has to process, which results in a delay of x frames. This is an undesired scenario; in Chapter 6 & 7 we will show further how to monitor the execution time and avoid exceeding the given time frame.

4.3.2 Audio Streams

With the established WebRTC peer connection we can transfer audio frames to the SFU that were recorded by the provider client. Detecting patterns and transforming the streamed data follows the same rules as for video frames, although we might need to specify more clearly what is included in an audio frame since it is not self-explanatory. A common microphone could have a sampling frequency of 48000 Hz, record with one channel (i.e. mono), and have a resolution of 16 bits. The question at hand is now how to split the live audio recording into frames that can be streamed from providers to consumers; a common choice, according to [76] is to choose frames with a length of 10, 20, or 30 ms. We reason that a longer frame duration will increase the risk of a frame being dropped, because the frame's absence will create a stuttering effect in the received audio stream.

We kept to the idea of static trigger functions that do not maintain any state. However, there is only limited possibilities for detecting privacy violating patterns in audio frames of this length. Audio data might contain a variety of privacy violations, for example songs that are not supposed to be shared with subscribers, or discussions, where for example the names of individuals are to be removed. Both could actually be implemented in the Python environment, there are plenty of frameworks for speech recognition [77] and it is possible to detect songs (i.e. audio snippets) in a recording [78]. Still, they require substantially longer recordings, but aggregating audio frames to such an audio snippet would introduce an undesired delay in the SFU, therefore, we focus on a single pattern we can detect in frames of less than 30 ms: human voice detection.

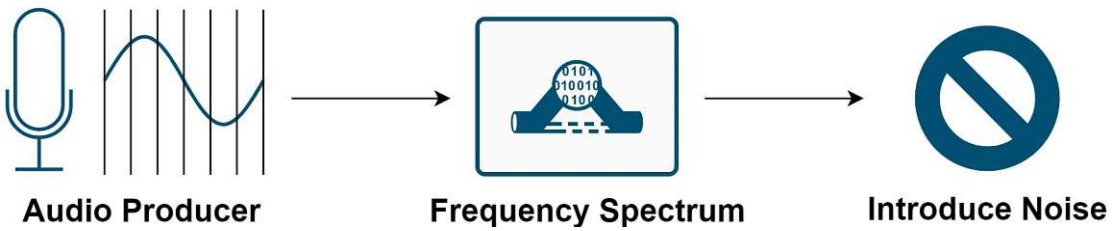

Figure 4.5: Analyzing audio frames to replace human voice with white noise

We decided to evaluate the py-webrtcvad [76] package to detect human voice activities by analyzing the frequency spectrum of a given audio frame. Our provider client breaks the live recording down into chunks of 20 ms, which are then streamed to the SFU. As for video frames, audio frames contain a variety of metadata, among it the sampling frequency, duration, and resolution. These parameters are needed for configuring the Voice Activity Detection (VAD) package, we might agree on them when establishing the audio stream, but we can also extract them on premise from the audio frame. The VAD package is then applied on the audio snippet and yield whether or not there were human voices detected in the audio frame. Although the package has shown to work as documented by the author, we see little more possibilities to cope with privacy issues as

to drop the frame, or replace it with a white noise sound. Figure 4.5 provides a quick overview of how the frequency spectrum is analyzed when receiving audio frames from the producer. If we detect human voices in the frame we instead route a frame containing white noise from the SFU to the consumer.

Because of the overall lack of triggers and transformation functions for audio frames, we decided not to continue working on audio tracks for the remaining chapters. This means that their performance will neither be evaluated. We conclude this section with the knowledge that it is feasible to detect and eliminate privacy violations within audio frames. They can likewise be broken up into data chunks and streamed over the SFU, from where they can be relayed to stream consumers. Nevertheless, we want to focus on video streaming at this point because we have already defined a variety of trigger and transformation functions that are sufficient for evaluating our idea of privacy models. Resuming this idea for audio frames in future work is only a question of the technical possibilities for audio processing; our privacy model is suitable equally.

4.4 Privacy Model Specification

By now we have a set of 4 trigger and 3 transformation functions that we can chain together to form a privacy enforcing model. However, we are still lacking a human-readable representation for specifying such a model, as well as a compiler and execution environment in the SFU that enforces policies. We decided to describe chains of functions in a textual representation, where individual links of the chain represent the trigger and transformation functions defined. 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 for a PM that is not fully aware of technical implementations hidden below the function names. We can imagine a common use case in which the faces of individuals are blurred on a surveillance camera, an equivalent model is given below:

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

The first link in the chain defines what kind of media sources our privacy chain should be applied to. In this case, the provided chain is used for 'video' streams. We implemented a whitelist mechanism here: in case that the SFU does not have any active privacy chains for a data type X, it cannot be consumed by any client. An empty chain that does not contain any triggers or transformations (e.g. video:{'tag':'webcam'}) is instead a configured rule that the data can be consumed without applying further operations.

The two remaining links in the chain describe the operations that are to be executed on every video frame that is processed by the SFU: First we detect whether or not there are human faces included in the video frame (Face_Trigger:{}), and for every face we detected we blur the respective region where it is contained (Blur_Area_Pixelate:{'blocks':5}). The curly braces contain parameters that are passed to the functions, the only given

parameter represent the number of pixelated squared in the blurred area. We will discuss in detail what parameters are accepted by trigger and transformation functions in the respective Sections 4.4.2 & 4.4.3, for now it is sufficient to note that they are a mean to configure these functions. It is a quick task to convert such a chain into a flow diagram and vice-versa: Figure 4.6 contains a graphical representation of the face blurring function we defined above.

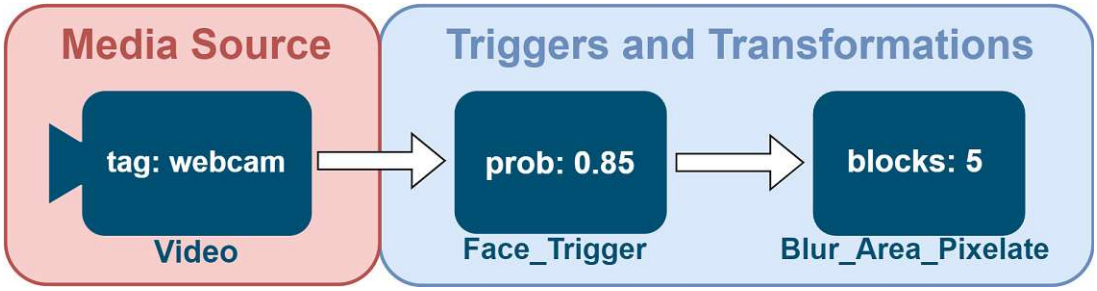

Figure 4.6: Graphical representation of a privacy chain for blurring faces

4.4.1 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 map for conversion must be available before compiling the privacy chain and maps a function name as string (e.g. 'Face_Trigger') to a Python function (e.g. Face_Trigger()) from Section 4.3.1. 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.

In a privacy chain we index the first element (i.e. the media source) with -1, the first function as 0, the second as 1, the third as 2, and so forth. To save execution time, we only process a chain up to the *nth* element that yields a result, so if a link n resolves to a trigger function that does not detect any pattern in the frame, we skip the remaining chain links $n+1, n+2, ...$

The last aspect of specifying privacy chains that we did not yet discuss is related to the "tag" parameter that is included in the above chain, as well as in Figure 4.6. Eventually, we need to differentiate between a privacy chain and a privacy model: the former is a chain of functions that describes how a frame is processed, while the latter is a collection of privacy chains. We might want to define chains for different media sources (e.g. audio, video, or data-channels), or multiple chains for the same media source. Multiple data sources are desirable when we stream for example both audio and video data over the

SFU, privacy operations can be executed in parallel for both streams before relaying them to audio / video consumers. But we can also express multiple privacy chains for one data source: when negotiating the peer connection between provider client and SFU the provider might supply a tag that characterizes the stream further (e.g. webcam or security camera). This tag is stored alongside the peer connection at the SFU; before streaming the data to a consumer the SFU browses its list of available privacy chains and selects the one most fitting, i.e. the one that has the same media source and a matching tag. In case that the is no chain with the same tag available or the stream does not deliver any tag we apply the last chain that has a matching media source.

For the deployment of privacy models, which will be discussed in Section 4.5, we need to clarify that we only deploy privacy models, a privacy chain on its own cannot be deployed. Privacy chains are always part of a privacy models and cannot be replaced or exchanged without re-deploying the entire privacy model and all available privacy chains. In the following 2 chapters we will now take a look at trigger and transformation functions we provided for definition of privacy chains. We already presented the implementation of these functions in Section 4.3.1, so we will now focus on method signatures and parameter configuration.

4.4.2 Trigger Functions

Below we provide an overview of the 4 trigger functions we implemented for function chains. Such a table could be provided to a PM in the fashion of a Javadoc and provide sufficient documentation on how to configure a privacy model before deploying it. The string representations of trigger functions is equivalent to the function names we defined in Section 4.3.1; the method signature refers to the expected parameters and the results the function returns. For some triggers we also provide a list of labels that the function returns (e.g. 'male' when using the Gender_Trigger), for these functions a PM must specify a label parameter in the privacy chain string, otherwise the privacy model will fail to compile.

For the trigger functions we introduce the term 'boxes' as a parameter and return type: a box is precisely a tuple of 2 points in the video frame that span a rectangle-shapes 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.

- **Face_Trigger**

  Description: Requires a video frame from a video source as input parameters and returns all the boxes where human faces are detected in the video frame. If we supply a set of boxes as parameters the model is only applied on this box areas.

Method Signature: frame, {options} → frame, [boxes]

Parameters:

prob float between 0.0 and 1.0, defines a threshold for the confidence that a face was detected, low values require a low confidence and high values a high certainty. Defaults to 0.7 float.

boxes np-array of boxes that can be used to restrict the model to process only the designated frame areas. Useful to accelerate the frame processing, defaults to [∅].

res enum that specifies whether the 320px model or the 640px model is applied. The processed video frame requires a minimum width of 320 or 640px respectively, otherwise the result is distorted because the image is resized internally to match the model requirements. The '640' model yields more accurate results than the '320' version, defaults to '320'.

■ **Age_Trigger**

Description: Requires a video frame from a video source as input parameters and returns all the boxes where a human face with the specified age range is detected. If we supply a set of boxes as parameters the model is only applied on this box areas.

Method Signature: frame, {options} → frame, [boxes]

Parameters:

prob float between 0.0 and 1.0, defines a threshold for the confidence that a face was detected, low values require a low confidence and high values a high certainty. Defaults to 0.7 float.

boxes np-array of boxes that can be used to restrict the model to process only the designated frame areas. Useful to accelerate the frame processing, defaults to [∅].

label enum that refers to the age range that detected human faces must belong to. Can be any of (0-2), (4-6), (8-12), (15-20), (25-32), (38-43), (48-53), or (60-100). Does not have a default value and must always be specified.

debug bool value that determines whether we print the age range over every human face that was detected. Must be True for printing debug information, defaults to False.

In the below Figure 4.7 we have depicted the result of the Age_Trigger in combination with the later introduced Blur_Area_Pixelate function. To visualize the result of the Age_Trigger we activated the *debug* parameter which prints the estimated

age over each face. We can see clearly that the model is not flawless in regards to the age range it provides, nevertheless, it worked correctly in 3/4 cases and provides an idea of the capabilities we have available.

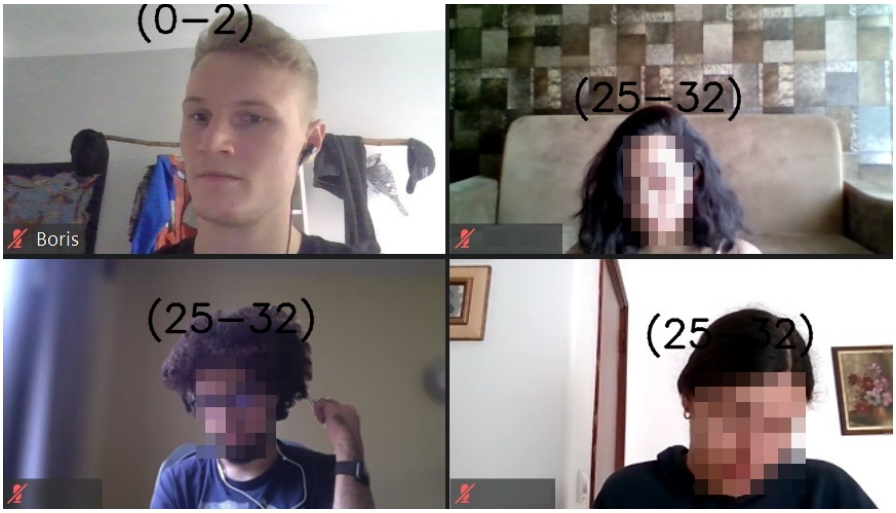

Figure 4.7: Detecting and anonymizing faces of humans with age between 25-32

■ **Gender_Trigger**

Description: Requires a video frame from a video source as input parameters and returns all the boxes where a human face with the specified gender is detected If we supply a set of boxes as parameters the model is only applied on this box areas.

Method Signature: frame, {options} → frame, [boxes]

Parameters:

prob float between 0.0 and 1.0, defines a threshold for the confidence that a face was detected, low values require a low confidence and high values a high certainty. Defaults to 0.7 float.

boxes np-array of boxes that can be used to restrict the model to process only the designated frame areas. Useful to accelerate the frame processing, defaults to [∅].

label enum that refers to the gender that detected human faces must belong to. Can be 'male' or 'female'. Does not have a default value and must always be specified.

debug bool value that determines whether we print the gender over every human face that was detected. Must be True for printing debug information, defaults to False.

■ **Car_Plate_Trigger**

Description: Requires a video frame from a video source as input parameters and returns all the boxes where a Vietnamese car plate is detected. If we supply a set of boxes as parameters the model is only applied on this box areas.

Method Signature: frame, {options} → frame, [boxes]

Parameters:

prob float between 0.0 and 1.0, defines a threshold for the confidence that a face was detected, low values require a low confidence and high values a high certainty. Defaults to 0.7 float.

boxes np-array of boxes that can be used to restrict the model to process only the designated frame areas. Useful to accelerate the frame processing, defaults to [∅].

We can chain the above trigger functions together arbitrarily to detect patterns in the video frame. All of them accept boxes as parameters to restrict the area the trigger processes, they can either be specified explicitly or passed between functions anonymously. This means that we can specify a sub chain of 2 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.

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

Notice that a trigger function c_n is as we mentioned not executed anymore 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 we release the frame without any transformation. If we do not want to pass box parameters internally we can also specify them as parameters, this overrules boxes that would have been passed internally. In the following section we will now provide the transformation functions so that detected patterns actually result in frame modifications.

4.4.3 Transformation Functions

We introduced the term of 'boxes' for trigger functions, limiting the area in a frame where we detect privacy violating functions. Boxes also play a central role for transformations because they require a box or a set of boxes to specify explicitly which areas in the video frame should be processed according to the transformation function. Even if we want to transform the whole video frame (e.g. blur the entire image) we need to provide the frame boundaries as a single box. The below list equally contains a list of functions names, method signatures, and parameters that can be used to tune functions.

■ Blur_Area_Pixelate

Description: Requires a video frame from a video source and a set of boxes as input parameters, returns the video frame with all boxes' contents blurred. Returns the unprocessed image if no box was specified.

Method Signature: frame, {options} → frame

Parameters:

blocks Int that describes a grid, where each cell is blurred on its own. So for a parameter value of 3 we divide the boxes' areas into $3 * 3 = 9$ cells, where we calculate for each cell an average color in which the cell is filled. Must be a positive number, defaults to 1.

boxes np-array of boxes that is required to point out the designated areas that should be transformed. Defaults to an empty set [∅], which indicates that no areas will be blurred.

In the below Figure 4.8 we show the effect that the 'blocks' parameter has on the blurred images. A mediocre value like 5 blocks produces a prettier result, but with an increasing number of blocks the processing time rises equally; this will be evaluated later. Ironically, a high number of blocks results in a decreased anonymization effect, as we see in the last image with a number of $30 * 30 = 900$ blocks calculated.

Figure 4.8: Blurred face with a number of $1 * 1$, $5 * 5$, or $30 * 30$ blocks calculated

■ **Fill_Area_Boxes**

Description: Requires a video frame from a video source and a set of boxes as input parameters, returns the video frame with all boxes' contents filled with color. Returns the unprocessed image if no box was specified.

Method Signature: frame, {options} → frame

Parameters:

boxes np-array of boxes that is required to point out the designated areas that should be transformed. Defaults to an empty set [∅], which indicates that no areas will be blurred.

color Blue-Green-Yellow encoded color that is used to fill the boxes. Defaults to (255,0,0), which represents a pure blue color.

stroke Integer that defines how thick the borders of the rectangle are drawn. A value of 1 represents a border of 1px, 10 results in a border of 10px. Any negative value will result in the whole rectangle being filled, defaults to -1.

■ **Max_Spec_Resize**

Description: Requires a video frame from a video source as input parameters, returns the video frame with its dimensions resized according to the provided parameters. Returns the unprocessed image if no parameters were specified or the image already complies with the boundaries.

Method Signature: frame, {options} → frame

Parameters:

max_width Positive Int that denotes the maximum width that an image can have, if the video frame exceeds the boundaries it is resized to the parameter value. Maintains the proportion of the image, decreasing the height as well. Does not have a default value and is applied before the max_height rescaling.

max_height Positive Int that equally denotes the maximum height that an image can have. Also maintains the aspect ratio of the image, does not have any default value, applied after the max_width rescaling.

4.4.4 Privacy Model Examples

We fully introduced our set of trigger and transformation functions that can be used to map a variety of privacy requirements to privacy models. To give a better understanding of how these functions can be chained we provide 2 further use-cases with examples for privacy models:

1. We are monitoring traffic activities through a video stream to detect traffic jams and indicate to authorities whether the monitored junction is overcrowded or has a low capacity utilization. There should be a live stream as well as the possibility to record the stream for later analysis. To avoid any abuse of personal data we anonymize faces and car plates that show up in the video stream.
   A: We install a surveillance cameras in the concerned traffic junction and an edge device in its vicinity. Video streams are provided to the SFU and can be consumed through a recording device and a live monitor in parallel. We set up the below model, which blurs personal information (i.e. faces and car plates) in the video frames:

$$video : \{\} \rightarrow Face\_Trigger : \{'prob' : 0.9\} \rightarrow Blur\_Area\_Pixelate : \{'boxes' : 3\}$$
$$\rightarrow Car\_Plate\_Trigger\{'prob' : 0.9\} \rightarrow Blur\_Area\_Pixelate : \{'boxes' : 3\}$$

2. Due to Covid-19 Restrictions we need to monitor the number of visitors that enter the changing rooms. We already implemented a chip card system that counts the number of persons that move in and out, but we also want to install a surveillance camera. To protect personal privacy we want to blur faces in front of the male/female changing rooms. However, individuals moving into the wrong rooms (e.g. males into the female changing room) should not be anonymized because of a given violation of usage restrictions.
   A: We install a camera in the hallways in front of each changing room together with an edge device in their vicinity. Due to the current limitations that only one device can provide a stream at the same data a security guard would have to actively switch between them, which is not a fully convenient scenario. The security cameras have an API interface like the provider device in Section 4.2 and it is possible to remotely start or stop video streams. Each camera has a tag it provided together with the stream, which is either 'm-change', or 'f-change', based on this tag the male or female faces are blurred and the other gender left unchanged. Note that the below is a combination of 2 privacy chains that form a privacy model, the two chains are separated with a line break '\n' where the SFU selects the appropriate chain based on the tag.

$$video : \{'tag' :' m - change'\} \rightarrow Gender\_Trigger : \{'label' :' male','prob' : 0.8\}$$
$$\rightarrow Blur\_Area\_Pixelate : \{'boxes' : 3\}\backslash n$$
$$video : \{'tag' :' f - change'\} \rightarrow Gender\_Trigger : \{'label' :' female','prob' : 0.8\}$$
$$\rightarrow Blur\_Area\_Pixelate : \{'boxes' : 3\}$$

4.5 Privacy Model Deployment

To deploy a privacy model on a SFU and thus enforce privacy policies on streamed data, we have 2 possibilities: The first one is to supply a model directly on application startup through the application parameters when starting the Python SFU server. To update the model and add, for example, a new privacy chain, we can, on the one hand, re-deploy the entire SFU, which is, on its own, not dynamic enough as a deployment strategy. Therefore, the second option is to update the model through the API that the SFU exposes, submitting the string-encoded privacy model through an HTTP Post request.

When deploying the SFU, we expose a variety of parameters that can be passed to the Python server, where they are resolved through an ArgumentParser. All accepted arguments are listed in Table 4.4, along with a brief explanation. The applicable arguments align closely to the server example of aiortc [79], which served us as an initial template. The SSL parameters only concern the HTTP(S) API that is used for configuration and establishing connections through provider and consumer clients, independent of streaming data through WebRTC.

Argument	Default	Description
– host	0.0.0.0	Host address for Python HTTP server
– port	4000	Port for Python HTTP server
– cert-file	None	SSL certificate file (for HTTPS), requires key-file as well
– key-file	None	SSL key file (for HTTPS), requires cert-file as well
– privacy-model	None	Privacy model encoded as String

Table 4.4: Overview of configurable arguments for the SFU

If there is no privacy model supplied on SFU startup, by default, no client can consume the stream. We already explained that this is due to the whitelist mechanism we implemented. To directly allow stream provision and consumption on startup we need to specify a suitable privacy model on premise. This decreases the time until a client can consume the stream because we do not have to first update the empty model through a separate HTTP request after startup.
In the below Table 4.5 we give an overview of the 4 HTTP commands that the SFU exposes for clients and PMs. One of them returns the JS-based consumer client, which we decided to provide automatically together with the SFU instead of using a separate application as for the Python-based producer client. It is possible to consume the stream through any web browser (e.g. mobile or laptop devices), we just provide the webpage and scripts through the SFU server. The browser-based consumer client uses the available */consume* route to establish a connection through SDP and display the stream's content directly on the page.

Method	Path	Params	Description
Get	/	{}	Access the browser-based consumer client
Post	/provide	SDP-Info	Connect a provider client through SDP
Post	/consume	SDP-Info	Connect a consumer client through SDP
Post	/privacyModel	Model-String	Update the active privacy model

Table 4.5: API interface for SFU server

When updating the privacy model through the provided API, changes are reflected immediately in ongoing video streams because we pass the new model directly down to every active VideoTransformationTrack. This assumes that the new privacy model was compiled successfully; otherwise, the model is rejected and the HTTP request results in a status code 400. If we did not update active stream connections this way, all connected clients would be disconnected and would have to re-connect through SDP, resulting in a number of frames being lost in this gap.
Another advanced feature that we discussed for the framework's concept in Section 4.1 is that we wanted to deploy asynchronous trigger and transformation functions during runtime as well, so that we could provide new privacy models containing novel functions. We did not implement this for now because we wanted to focus on our set of functions and evaluate them reasonably. It turned out that ONNX models are considerably large (ours are between 4Mb - 112Mb) and updating them without any impact on the active peer connections is itself a challenge.

4.5.1 Processing Environment

By now we have provided information on all the major components that our prototype consists of. However, one aspect that is required for deploying the SFU is missing: the requirements for the processing environment on the edge device. We will not specify minimum requirements for hardware specifications because this would force us to compare the performance on multiple edge devices, which is out of our focus. What we can provide instead is an overview of the software requirements that the SFU has for the underlying programming environment.

We summarized the requirements for the available programming environment in Table 4.6. If there are different versions listed in the *Minimum* and *Recommended* row, this is because we upgraded dependencies and were able to assert that the SFU would work correctly regardless of the update. In the case of Cuda and cuDNN, this was possible due to a major upgrade of the edge device's OS; for the others it were upgrades published throughout the development progress. In Section 5.3.1 we will focus on how Cuda and cuDNN are installed within the environment. The opencv and python-av dependencies are essential tools for decoding and transforming video frames, whereas the ONNX runtime is concerned with running trigger functions on the frames to detect privacy violations.

	Python	Cuda	cuDNN	aiortc	opencv	python-av	ONNX
Minimum	3.9	10.2	8.2	1.2.1	4.5	8.1.0	1.9
Recommended	3.10	11.4	8.3	1.3.1	4.5	9.0.1	1.10

Table 4.6: Software requirements for the programming environment

Whether we manually install the required tools through pip or provide them through an equivalent Docker image (as we will do for evaluating the performance in AWS in Section 5.3) does not make a difference. For now, we only provided an overview of the most important dependencies we used. The complete list of dependencies is included in the respective *requirements.txt* file, which was used to build the Docker image.

We conclude this chapter after having presented all of the major details for implementing our prototype. We discussed how clients can provide and consume data through the SFU, how we can detect privacy violating patterns in video and audio streams through ONNX models, how these models can be wrapped in static functions that are configured through parameters, how these functions can be chained together to form privacy models, and finally, how we can deploy these privacy models on a SFU. It remains to evaluate our prototype in Chapter 5, where one evaluated aspect will also be how to deploy our SFU implementation on an edge device.

CHAPTER 5

Evaluation Methodology

We provided an abstract concept in Section 4.1 that enforces privacy requirements on continuous data streams through privacy models. We dedicated the rest of the chapter to implementing this concept; by doing so we already made a big step evaluating our solution because it can be seen as a proof of concept. Nevertheless, our goal was not only to match the requirements that we imposed on such a solution, but we also set the goal of evaluating the performance of the implementation through benchmarking. This originates from RQ.3 in 1.3, by which we want to provide an answer regarding the actual applicability of a prototype.

We start this chapter by giving an overview of the aspects that we want to evaluate for the proposed framework. This step is not limited to raising metrics for the SFU's performance, but includes how well our prototype matches the requirements we specified in the problem statement and the respective concept. Later in Section 5.2 we introduce the scenario that we will evaluate and reason how we came to the decision; in Secion 5.3 we present the hardware that will be used for processing and routing the video streams.

5.1 Overview

To evaluate the performance of our solution we will focus on two central metrics that concern the streaming capabilities of the implementation we presented. We define the overall streaming latency as the time it takes to transfer a frame from the provider, 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 RTT between provider to SFU and from the SFU to the consumer is supposedly related to the network architecture in which we implement the solution: a naive assumption would be that the latency is lower if we install the SFU close to the provider/consumer client instead of a cloud center. We do not actually expect a cloud setup to yield better

results than an edge architecture, but we will evaluate how big the difference of both architectures is for our WebRTC streaming setup.
We estimate that the fractions of the RTT, where frames travel from producer → SFU or from SFU → consumer, do not diverge significantly, at least not based on the edge or cloud architecture. Nevertheless, we will evaluate both shares, but due to time constraints only measure the latency for producer and consumer clients once each. This means that we will solely provide results on the latency from producer → SFU for the cloud setup, and from SFU → consumer for the edge setup.

If the SFU in our architecture were only a streaming platform that forwards frames from providers to (multiple) consumers, we would not expect it to considerably affect the RTT. However, a central aspect of the SFU is the processing of frames according to the deployed privacy model, therefore we need to measure how big its impact is exactly. A core feature of our framework is the customizability of privacy models through a PM. Therefore, we will compare the overall processing time of privacy models in different use-case scenarios.
Ascertaining the overall processing time is a good indicator of whether a whole chain can be processed within the given time boundaries, but we can thereby not reason about individual processing times for applied functions. This will be evaluated in more detail for the implemented triggers and transformations, where we will also assess whether different parameter settings have an impact on the function's performance. Based on these results, a PM can estimate whether an entire chain can be processed in time by aggregating the individual chain links' execution times.

When we extend the SFU with new privacy-enforcing functions, these would have to be evaluated equally to give evidence of their performance. This concerns new video transformations or ONNX models, as well as entirely new media types, like audio operations. The environment we provide to evaluate the performance of video operations can server for future evaluation of audio triggers and transformations. This means, that it is equally possible for us or other researchers to evaluate new functions with the given framework.

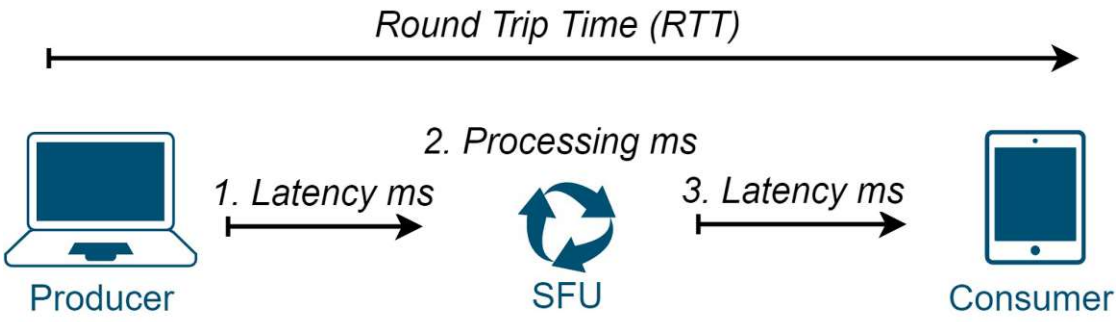

Figure 5.1: Overview of the metrics evaluated for the presented prototype

In Figure 5.1 we visualized how the overall RTT is related to the latency between SFU, provider, and consumer (Label 1 & 3). Their share of the RTT is evaluated through metrics together with the SFU's performance (Label 2). We only need to conduct a single session to measure all three of them, the latencies and the time required to process the frames are exported automatically as Comma Separated Values (CSV) sheet to a dedicated directory. These documents will be addressed in Chapter 6 & 7 to give evidence of the measured metrics.

We embedded this thesis in the context of edge computing, where powerful devices can perform privacy transformations close to the data source. To support a portable network architecture, the resulting SFU server must be equally deployable on a variety of edge devices, one of them the NVIDIA Jetson. In Table 4.6 we specified the requirements for deploying the SFU on an arbitrary device, though this is only related to the programming environment that must be available, not the hardware specifications. Whether a device is convenient for processing a certain privacy chain depends precisely on the underlying technologies in the SFU.

We presented a set of trigger and transformation functions that all operate on video stream data, the reason was not the overall demand for video streams, but that it provides a superior number of ONNX models and well rehearsed usage scenarios. Within this thesis it was our goal to provide a universal environment for privacy models, in a future work we can progress further into one direction (e.g. video transformations) and compare the different results of edge devices to select the best one for a specific use case. We will present how well the deployment works for the NVIDIA Jetson Xavier NX and the underlying GPU-acceleration because deploying the SFU on an edge device is part of the proof of concept, the evaluation of individual functions will be carried out on a different device in the edge network.

An insightful information would also be how well our initial prototype fares compare to alternative versions when we replace certain tools or technologies in the architecture. Going on step further, we could also compare our solution to an entirely new prototype that is based on another programming environment (e.g. Node.js) and use only tools from this environment. However, for this thesis we want to focus on setting up the means for measuring the identified metrics and give results for the given prototype, those can be used for high-level comparison in future work.

5.2 Evaluated Scenario

We assume a streaming scenario in which a client provides a webcam stream to the SFU, which is consumed by one (or more) clients running in a web browser. The provider and consumer clients are the ones we introduced in Section 4.2.1 and expose the presented API for starting and stopping stream consumption and provision. Especially with the COVID-19 pandemic, the usage of video conferencing tools (e.g. Zoom) has increased widely for education and working environments [80], thus we decided on a scene in which supposedly a variety of private details were recorded over the last few years without our

awareness. We choose this scenario in alignment with [35], where the authors evaluated their prototype for processing low-latency video analytics on the edge. Even though our framework is not primarily designed to support low-latency stream processing, we want to evaluate whether the performance of our stream is sufficient for relevant areas, like Augmented Reality (AR) and Virtual Reality (VR).

To protect personal privacy in the webcam stream, we, in our role as PM, have decided to blur all individuals' faces that are detected when routing and processing the stream over the SFU. The possible number of people that appear in the video is infinite, though the bound resolution of the image causes increasingly smaller faces to become unrecognizable in any case. The principal scenario will be to detect faces and blur them. Within this scenario, we want to vary the privacy chain by adding a trigger to detect only a certain gender ('male') or age range ('25-32') or to resize the image before processing it. The result will be anonymized first with the given Blur_Area_Pixelate function and later as comparison just replace faces with Fill_Area_Box. Since the aesthetic appearance of pixelated faces is still relevant to us, we will also tune the size of the pixelated grid and analyze whether a higher number of pixels affects the execution time.

We extend the privacy chain with more and more functions because, in our role as PM, we want to ensure privacy in a multitude of ways by detecting every possible privacy violation. We use as many triggers as applicable in the given time frame without affecting the streaming latency negatively, thus, fully utilizing the capacities we have to maximize privacy preservation. The cap for exceeding the time frame will always be the moment when frames are starting to get dropped, as explained in Section 4.3.1.

5.2.1 Resulting Privacy Models

We provide a visual representation of the privacy models that we want to evaluate in Figure 5.2: The individual privacy chains were created based on the given scenario, where one sequence of colored arrows represents one chain that will be evaluated. For example, the orange chain describing *Model #1* reads as follows: receiving data from a video source, applying a face trigger, and finally blurring all faces that were detected. We are interested in how the different privacy chains will compare to each other as well how long the individual functions will take. We already defined two suitable metrics to evaluate that, namely the overall RTT (i.e. total streaming latency) and the execution time of function in the SFU.

An additional factor that we want to evaluate within our scenario is the effect of the *blocks* parameter of the Blur_Area_Pixelate function. We already provided in Figure 4.8 an overview of the resulting images with different *blocks* values, nevertheless, we want to detect if the different values do in fact have an effect on the performance. To that extent, we will vary the *blocks* parameter in *Model #1* and measure the transformation's performance with values of {1, 50}. We perform this evaluation as an example of whether parameter values might affect the SFU's performance. Clearly we cannot suspend the possibility for other functions even if we do not detect a difference here.

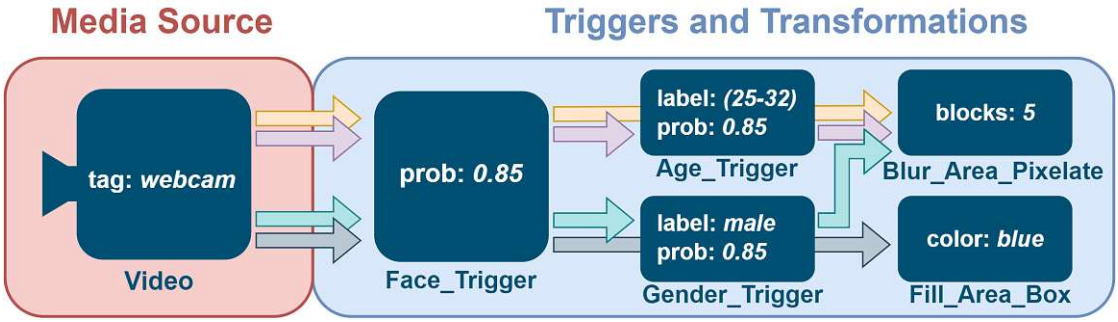

Figure 5.2: Evaluated privacy models in different colors (orange, purple, green, and grey)

Our evaluation goes further than comparable research on privacy policy enforcement [6], [30], where streams where 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 that a PM has available.

5.3 Experimental Setup

We already explained that in our evaluated scenario, we vary a limited set of trigger and transformation functions within the principal privacy model, in combination with different parameter values. As a result, we want to measure the overall execution time of the model as well as the performance of individual functions. The question at hand is how to provide a consistent streaming environment in which outer influences can not affect the streaming performance by any means. As discussed in Section 4.2.1, we have the possibility to replay a recorded video as an alternative to providing a live stream. As long as the resolution and the FPS of the recorded video match a comparable live stream, we claimed that the only difference is the already carried out encoding of the captured frames. Because the encoding would occur on the provider client in any case and thus have no effect on the performance of privacy chains on the SFU, we conclude that a video is a suitable choice to prevent outside influences between measurement sessions.

We provide two prerecorded videos with different resolutions that will be used to evaluate the performance of the SFU. The second video originates from the first one. The difference is just that its resolution and frame rate were decreased. Therefore, there are no substantial differences in the video's content, the only theoretical difference might originate from the content of frames that were removed by decreasing the FPS. Table 5.1 provides a summary of the specifications mentioned for the recorded videos.

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 quantity of frames per second), the available time frame for processing is decreased. By evaluating video streams with two different frame rates, we intend to ascertain this fact, or otherwise disprove our assumption.

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

Table 5.1: List of two recorded videos that are used to evaluate the prototype

5.3.1 Hardware Setup

It is not within the focus of this thesis to select the best edge device for a given use case. Therefore, we do not evaluate the SFU's performance on the NVIDIA Jetson for convenience reasons. Instead, we will only deploy to the NVIDIA Jetson to conclude the proof of concept, the evaluation of the performance will be carried out on a different device. The hardware specifications of the device we used are depicted in the below Table 5.2; the device has a NVIDIA graphic card installed which supports NVIDIA Cuda for GPU-accelerated graphic processing of video streams. We followed the official instructions and resources from NVIDIA for installing the Cuda [81] and cuDNN [82] environment on our device.

OS Version	CPU	RAM	GPU	Cuda Version	cuDNN
Windows 10	AMD FX-6300	16 GB	NVIDIA GTX 960	11.4.3	8.2.2.26

Table 5.2: Hardware specifications of the device used to evaluate the prototype

We recall that it was never a requirement of this thesis to evaluate whether video processing is enhanced by harnessing GPU resources in the SFU. However, by evaluating video operations as the most advanced in terms of available ONNX models, it became part of the proof of concept: The principal goal was to start from a common privacy-concerning use case (e.g. blurring faces) and gradually define an abstract model that can be deployed on an edge device, where it is maintained by a PM. However, when building privacy models for video streams, our use case of blurring faces ran into performance issues using the given hardware setup, thus, exceeding the given time frame and resulting in an inconvenient streaming result. By evaluating the SFU's performance with or without GPU-acceleration, we intend to show that certain improvements can decide whether a use case is feasible in a given environment.

We advocate that for future research, it must be a paradigm to not accept a given time frame and work towards finding triggers and transformations that it can fit, but to seek out well-established methods to speed up processing within the time frame and accomplish the most beneficial scenarios. Whether a certain model fits, is of course a question that requires previous analysis of metrics. The given evaluation is a first step towards continuous feedback.

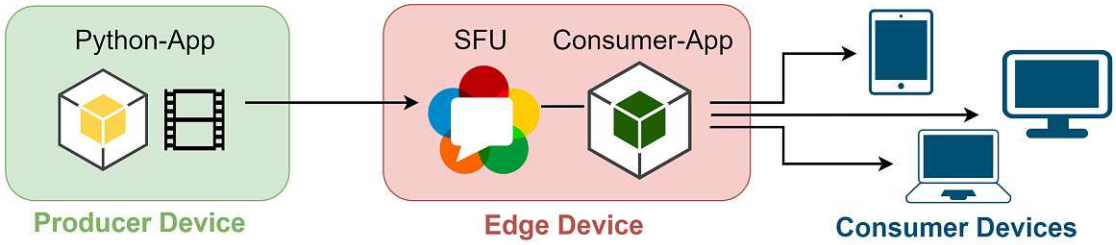

Figure 5.3: Streaming architecture for evaluating the prototype

All the entities that are applied for the evaluation scenario are presented in Figure 5.3, it contains our Python producer client application from Section 4.2 that connects to the SFU through SDP. Once a stable connection exists the producer client starts streaming the prerecorded videos from Table 5.1 with the respective frame rate and resolution. The SFU processes the received stream according to the supplied model from Figure 5.2, the transformed stream can then be consumed by a set of three consumer devices. We decided to use three different devices:

1. A smartphone, which is of special importance because we discussed the rise of crowdsensing in combination with edge computing. A useful choice if we want to consume streams on the go. We used the Samsung Galaxy S9 (SM-G960F/DS) as 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, which is a special case because we do not transfer frames between different devices. This means that stream packages are routed over the localhost without using additional network facilities.

The consumer devices connect to the SFU through a web application that is hosted on the edge device. For clarification, the consumer app is merely supplied through the SFU but is executed on the 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 5.2, this is the environment we will use to evaluate the performance of the privacy models and individual functions. In another step, we evaluate the deployment on the NVIDIA Jetson Xavier NX, but the performance is only evaluated with the above setup.

The RTT is influenced by two major factors: the performance of the SFU and the latency of the peer connection between the SFU and the producer and consumer clients. We evaluate the SFU's performance with the given privacy models and scenario, for

the remaining share of the RTT we will measure differences between a cloud and edge environment. With the above setup, we already have a running edge environment. For our cloud setup, we will deploy the SFU on AWS. To simplify the deployment on AWS, we created a Docker image of the SFU that includes all the respective dependencies. The image was shared through Docker Hub [83] and can be pulled and executed on an arbitrary Docker instance.

The AWS server we used for evaluating the latency for cloud setups was an EC2 instance, namely the t2.micro [84], which has a single vCPU and 1 GiB of RAM. The cloud setup is evidently not as powerful as the device we use for evaluating on the edge, but this is negligible since we do not compare the performance of processing the privacy model on the SFU. Instead, we exclusively compare the latency of producing and consuming video frames to / from the SFU. The performance of the privacy model is supposedly equivalent on two devices with the same hardware specifications, regardless of whether they are deployed in the cloud or on the edge.

CHAPTER 6

Results

In this chapter, we will present the results of the evaluated scenario from Section 5.2. This includes the specified metrics for the RTT and the performance of the SFU, which is contained in Section 6.1 and 6.2. At the end we conclude the proof of concept by deploying the SFU on the NVIDIA Jetson in Section 6.3. We focus on a value-free presentation of the results and try to provide only sufficient information to understand how the data was raised. In Chapter 7 we will then give our point of view regarding the applicability of the SFU in its current state. This will include a comparison of the presented concept with the proof of concept, the given prototype. All the graphics that are presented in this chapter were created with the Python Pandas [85] and seaborn [86] packages.

6.1 Streaming Latency

Instead of presenting the whole RTT for the streaming setup, we focus for now on the latency between the producer client to the SFU and from the SFU to the consumer. The latencies were first measured for the presented edge architecture and later with our cloud setup in AWS. We streamed the *Video #1* (see Section 5.3) from the producer which has a frame rate of 30; this is relevant because the frame rate defines how often the latency can be calculated: We do not measure the latency ourselves but access it through the peer-to-peer connection's statistics that the SFU maintains. With every frame that is transmitted to/from the SFU, these statistics are updated and can then be accessed by us for evaluation. Therefore, it is impossible to evaluate the latency with a higher frequency as the provided frame rate, this means that we cannot measure the latency more often than 30 times a second.

We measured the latency over a total time of 60 seconds where we accessed once a second the latency, thus we end up with 60 values for each session that we can compare to each other. Within the SFU we always had our privacy model *Model #1* executed, though

this supposedly does not make any difference for the latency from the producer or to the consumer. The consumer client was always executed on the same device from Figure 5.3, but for the consumer we varied our list of three consumer devices to detect possible differences. Consumer and producer streams were started simultaneously and the results captured from the very beginning of the connection, instead of from a long-running instance. We wanted to capture a possible unstable streaming latency at the beginning of the established connection, within 60 seconds we would supposedly already arrive in a stable state where we have less fluctuation in the latency.

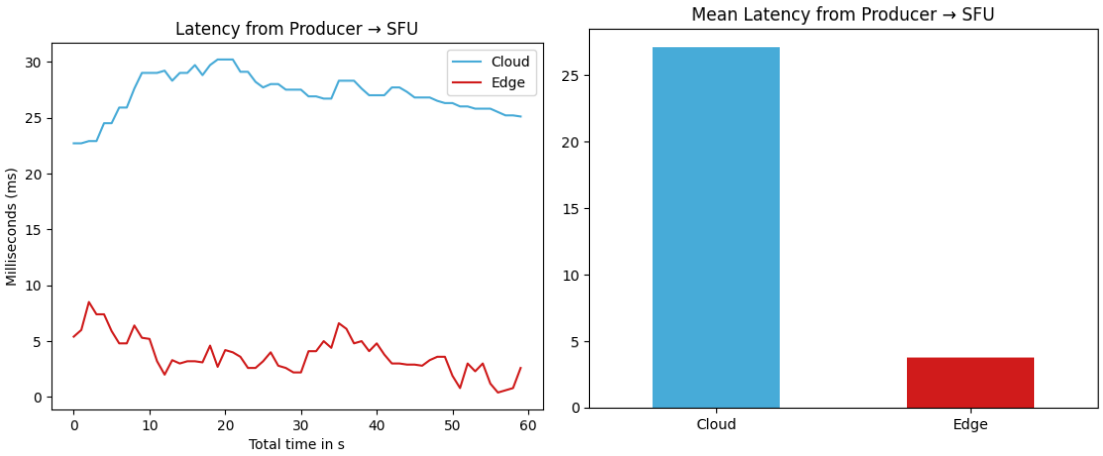

Figure 6.1: Producer latency over 60s for cloud and edge setups with mean values

In Figure 6.1 we visualized the latency of the edge and cloud setup in comparison over 60 seconds, the average values that are provided in the right subgraphic are for the same respective time frame. We can tell from the rather ragged shape of the curve that it was measured no more than once per second, if we would have evaluate the latency more often we would have gotten a more fine-grained result. Nevertheless, the development of the latency is fairly well visible with the given setup.

By now, we have evaluated the latency between the producer client and the SFU to detect differences between the edge and cloud environment. For the next step, we evaluated the latency between multiple consumer devices and the SFU deployed on the edge. The 3 devices are the ones we described in the experimental setup in Section 5.3. We recall that the first consumer device was a smart phone, the second a laptop, and the last consumer was executed locally on the SFU's device. The result is visualized in Figure 6.2, we tracked the latency once again over 60 seconds and provided the respective mean values in the right subgraphic.

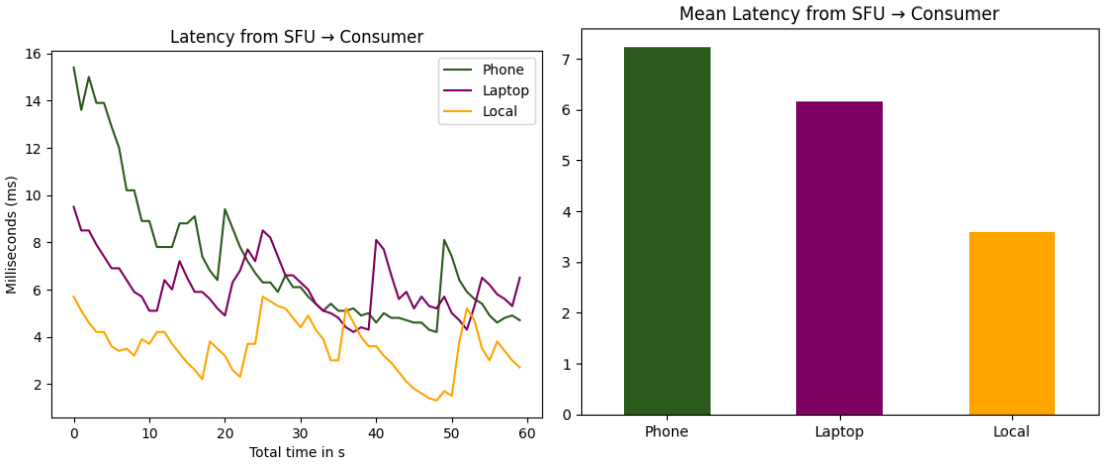

Figure 6.2: Latency of different consumer devices over 60 seconds with mean values

6.2 Privacy Model Performance

We will now focus on the central share of the RTT that the SFU entails - the performance of the trigger and transformation functions. In Figure 5.2 we presented the four privacy models that we will evaluate, for each of them we will provide the overall execution time alongside the individual function's performance. The individual functions and the overall execution time will be evaluated on the SFU with the GPU-acceleration enabled, only for Section 6.2.3 we will visualize for one privacy model the different performances with/without GPU support.

An important detail is that the last third of the produced stream contains a printed photo with three additional human faces. The increasing number of human faces might impact the performance. To give a precise idea of what the transformed stream looks like under these circumstances, we created Figure 6.3. We mention this now 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 over the course of the video stream.

6.2.1 Triggers and Transformations

In the below Figure 6.4 we have visualized 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 we wanted a higher resolution for evaluating the computation time, we evaluated the processing time once every frame, instead of only every 30 frames like in Section 6.1. However, to avoid an increasingly dense x-axis, we stopped the replay of the demo video after 10 seconds. We evaluate 10

Figure 6.3: Resulting stream with multiple faces transformed

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 the last one transferred.

In Figure 6.5 we used the exact same setup to visualize the performance for processing *Video #2*. As presented in Section 5.3, the differences are both the resolution as well as 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.

The given two visualizations provide a continuous representation over the individual processing times to monitor how the processing varies over time. In the below Figure 6.6, we added an additional box plot that describes the overall distribution of the values[1]. 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.

[1]Order of functions for *Video #1* & *#2* reads as follows on x-axis:
M1: Face_Trigger, Blur_Pixelate, Overall; *M2*: Face_Trigger, Blur_Pixelate, Overall, Age_Trigger; *M3*: Face_Trigger, Blur_Pixelate, Overall, Gender_Trigger; *M4*: Face_Trigger, Overall, Fill_Area.

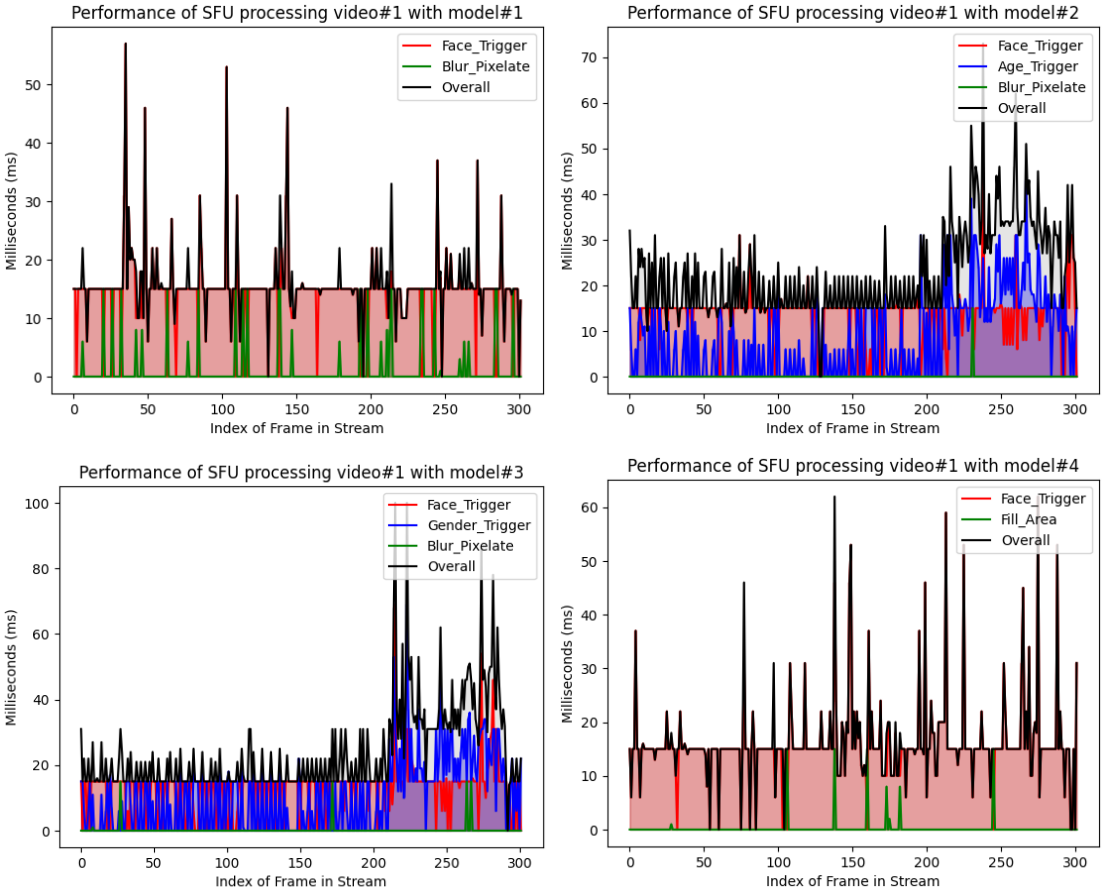

Figure 6.4: Individual performance of functions for *Video #1* and the 4 privacy models

6.2.2 Overall Performance

We already included the overall performance of the evaluated privacy chains in the presented figures in Section 6.2.1, alongside their individual function times. To give a more precise idea of the overall chain performances we included them exclusively in the below Figure 6.7. The left subgraphic contains the computed result for *Video #1* and the right one the results for *Video #2*, both were applied on 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 a different frame rate.

6.2.3 Parameter Tuning & GPU Acceleration

By now we have evaluated the performance of individual functions and the overall time that privacy model execution requires. As a result, we have already provided an overview of the approximate times that functions take when streaming the provided demo videos.

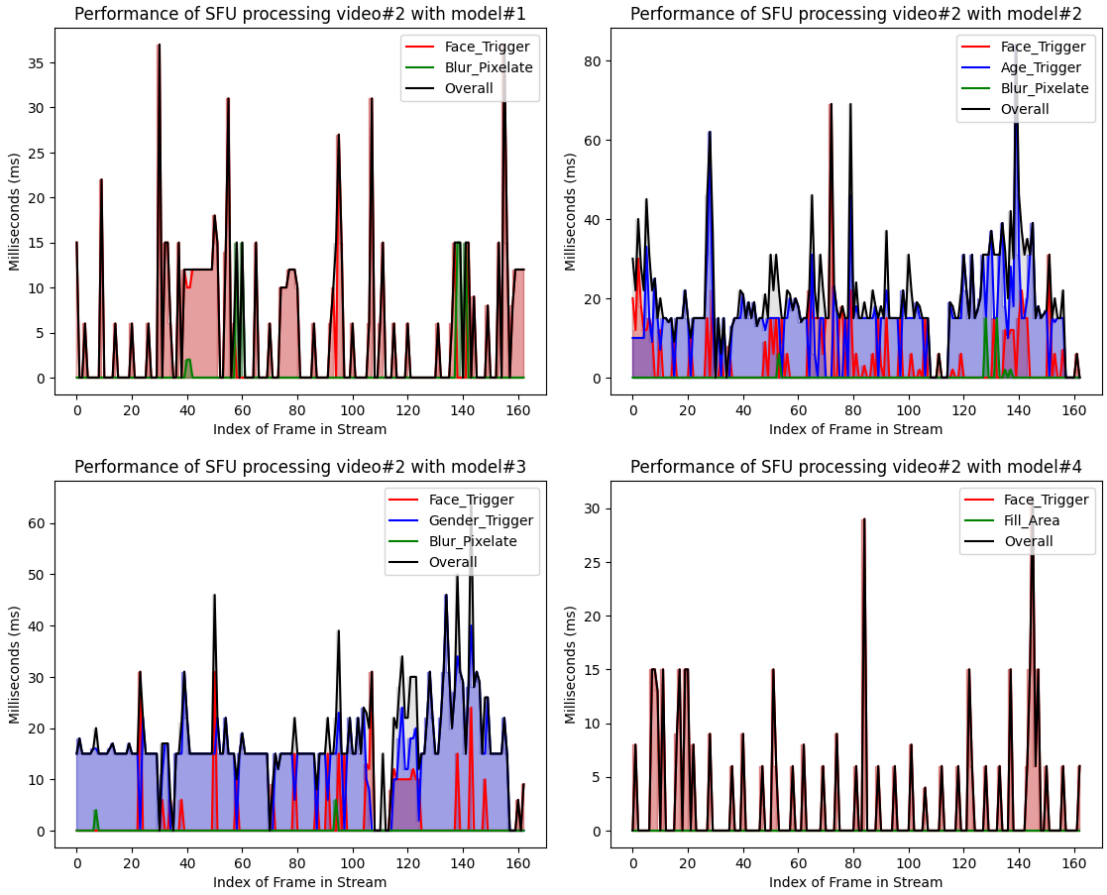

Figure 6.5: Individual performance of functions for *Video #2* and the 4 privacy models

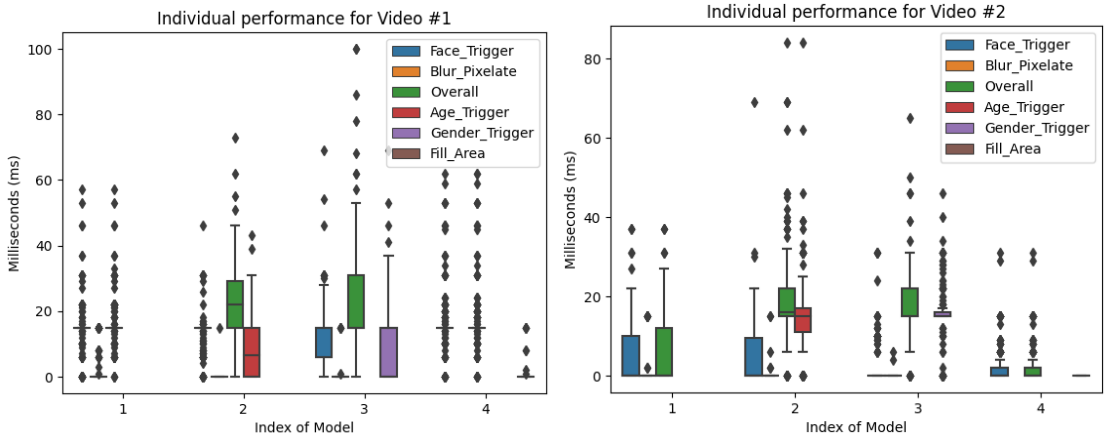

Figure 6.6: Statistical distribution of function's performances for *Video #1 & # 2*

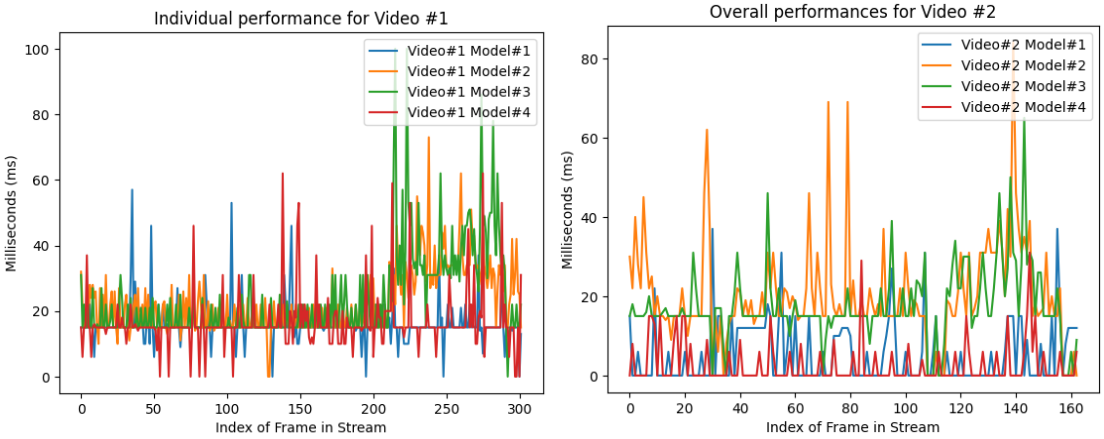

Figure 6.7: Overall performance of evaluated privacy models on *Video #1* & *#2*

For now, it remains to investigate further the configuration of privacy functions and their tuning, as well as how dependent they are 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 therefore with different settings to highlight diverging results. We will again only process the first 10 seconds of the video to increase the detail on the x-axis. With the given video, we will iterate once more over exactly 300 frames and measure the performance of the individual function and the overall processing time.

We visualized in Figure 6.8 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 4.8 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 6.4, the last third of the frames processed contains likewise an image with multiple human faces.

To evaluate the effect that the GPU-acceleration has, we deactivated the acceleration manually according to the scenario description. This time, we evaluated the four privacy models again, but only *Video #1*, so that we can compare the results to the four subgraphics in Figure 6.4. The results without GPU-acceleration are provided through Figure 6.9. The x-axis contains again all frames that are 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 overall time that elapsed for processing the privacy chain. Performance was evaluated on the exact same device with the same hardware specifications for the entire chapter to avoid external factors interfering with the results. To deactivate the GPU-support, we did not have to manually remove hardware parts, but we implemented a configuration parameter that allows switching between the available providers (CPU or GPU) for the ONNX model.

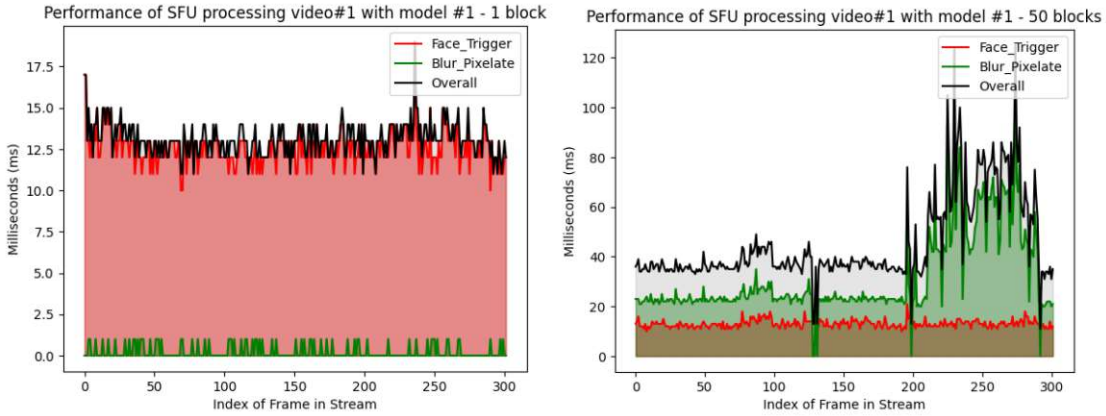

Figure 6.8: Performance of Blur_Area_Pixelate for *blocks* parameter values {1, 50}

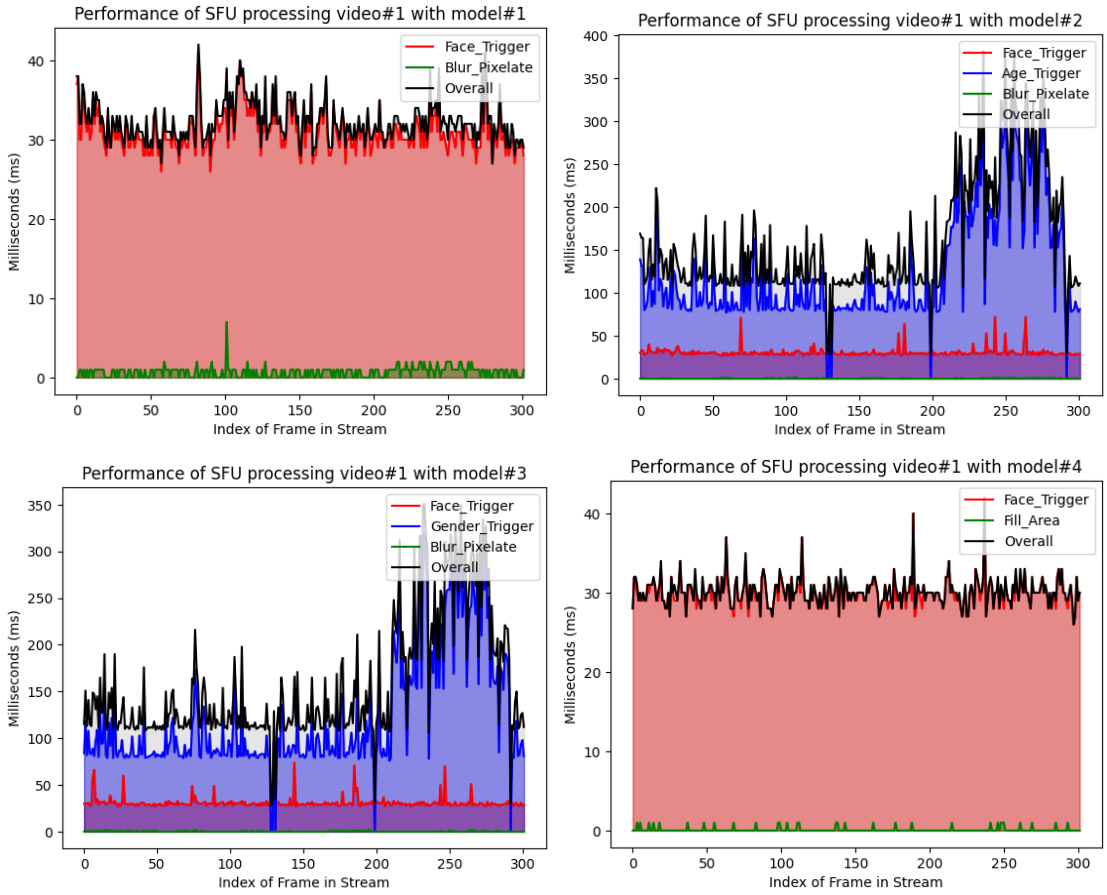

Figure 6.9: Individual performance of functions for *Video #1* without GPU-acceleration

6.3 Edge Device Deployment

In Chapter 4, we provided an abstract concept for a streaming framework that enforces privacy on a continuous stream of data. We dedicated the rest of the chapter to implementing this framework and thus providing a proof of concept. However, since the central idea of our work is to deploy the SFU on an edge device in the vicinity of the producer device, we conclude this proof by deploying the SFU on the NVIDIA Jetson Xavier NX.

For deploying to the NVIDIA Jetson, we flashed its memory card with Ubuntu version 20.04, which by default has Python 3.8.2 installed and is therefore below the recommended version from 4.6. We upgraded the Jetson's Python version to 3.9.10 and installed all the Python packages it requires through pip, except for the ONNX-runtime. The runtime for the ONNX models needs to be obtained manually from the Jetson Zoo [87] webpage, for Python 3.9.x we can install the *onnxruntime 1.10.0* version, which supports the accelerated processing of images over the Jetson's GPU. This is the exact same version that we applied for evaluating the performance in this chapter. The main difference is, that it is optimized for the Jetson's architecture, which does not support the default release of the ONNX runtime. With this, we already have the required environment for deploying the SFU on the Jetson. We cloned our GitHub repository and the included source code instead of using the Docker image we provided for AWS. We started the SFU with the default application parameters from Table 4.4, except for the privacy model, where we passed the string representation from the evaluated *Model #1*

The SFU is by now up and running and can be accessed over the Jetson's local network address with the default port of *4000*. We connected a producer and consumer client instance through SDP to the SFU and started streaming webcam images through the producer API in Table 4.1. The stream was transformed according to the deployed privacy model, thus blurring all detected faces in the video frames. We included a snapshot of the transformed stream in Figure 6.10. The stream was consumed by accessing the web consumer that is automatically deployed together with the SFU.

We conclude this chapter after having provided suitable results for the evaluated scenario and hardware setup. To complete the proof of concept, we deployed the SFU on the NVIDIA Jetson and achieved comparable transformations on an edge device. However, we did not provide any details on its performance. This will remain to evaluate for future work, for now we focused on whether it is possible at all to deploy to a commonly available edge device.

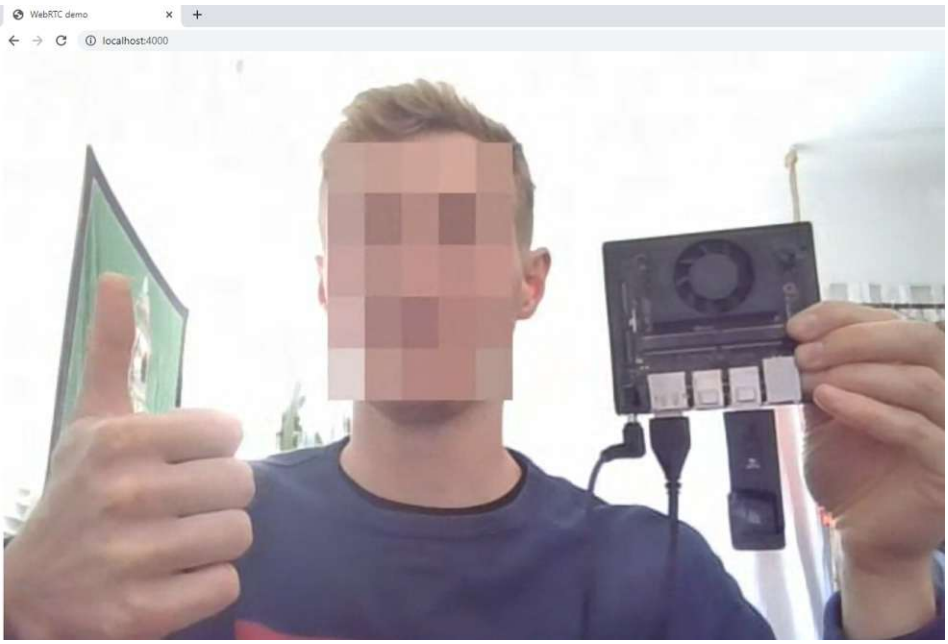

Figure 6.10: Consuming a stream after transformation on the NVIDIA Jetson Xavier NX

CHAPTER 7

Discussion

The central aspects of the conducted evaluation were the performance of the SFU when processing the given privacy models and the extent to which our prototype matches the abstract concept that we proposed. The former will be addressed in Section 7.1 and 7.2, where we discuss the resulting latency and performance and draw the boundaries for processing time. This drives us especially towards the answer to the third proposed research question that we stated, whether we consider the SFU applicable in its current state. The latter, whether the prototype fulfills the role of a proof of concept, will be discussed in Section 7.3. We will compare the ideas and requirements that we established to the status quo and discuss critical aspects of the given prototype that need to be improved or revised in future work.

7.1 Device Performance

We introduced the metrics that we wanted to evaluate in Figure 5.1, which were the latency from the producer towards the SFU, the performance of the SFU when processing privacy models, and the latency from the SFU towards the consumer. We will start interpreting the latencies that we measured for the given edge and cloud setup.

An important aspect of our raised metrics was that the results were obtained from streaming the exact same demo videos from the provider to the SFU, maintaining an even stream content for all conducted evaluations. Thus, we could segregate the effects of different models, functions, and parameters to decrease the risk of external factors distorting the results. However, from the raised metrics, we can only argue about the given results and not about critical situations that might come up in live situations.

7.1.1 Producer and Consumer Latency

The latency was measured in two independent sessions, once from the producer to the SFU, and again from the SFU to the consumer, both over a time frame of 60 seconds, where we evaluated the latency once every second, i.e., every 30th frame. When streaming to our SFU deployed in AWS, the producer client was able to maintain a latency which would not fluctuate more than 15% from the mean latency (26ms), thus frames were transferred in between 23-30ms. This can be observed from Figure 6.1, which contains the progress of the producer's latency and the respective mean values. We did not detect any particular pattern in the latency increasing or decreasing gradually over time, nor for the edge setup, which was oscillating slowly between higher and lower peak latencies. Precisely, the latency was fluctuating between 2 and 8 ms with the SFU deployed in the local network, which is in absolute terms an equivalent range. Based on these results, we see a significant difference in the latency depending on whether the SFU is deployed in our cloud or edge environment, with the cloud setup yielding a mean latency that dwarfs the edge-based one by a factor of 7.

Streaming latency is of special importance when building Augmented Reality (AR) systems. According to [88], the eye starts to notice a delay from up to 20-30ms. If we assume that a producer could be a camera mounted on an employee's head and the consumer a pair of AR glasses like the Microsoft HoloLens [89], worn by the same employee, we could already run into an unbearably high latency if we transform the video stream in our cloud setup. For now, maintaining a low latency by deploying the SFU on the edge, close to the data source, allows us to further pursue this use case. However, it still remains to analyze the latency and performance of the SFU and consumer.

For the latency between the SFU and the consumer, we only evaluated the edge setup. The mean latency of the consumer devices in Figure 6.2 aligns very closely with that of the producer in the edge setup. In fact, the mean latency moves between 3-7ms for the three different consumer devices. A naive assumption was that the latency for producing and consuming the video stream is equivalent. This can be partly affirmed: The producer's device was a laptop with an integrated camera, so it can be compared with the laptop consumer (purple bar). From the given results, a laptop was able to provide and consume with a latency of 4ms and 6ms respectively, which would still provide a significant difference. However, if we trace the consumers' results from the beginning of the measurement to the end, we can observe a decreasing pattern in the first third, indicating that the latency might improve when consuming over a longer time period. This should be affirmed by measuring over a longer time period to avoid any possibly unstable latency in the early stages of the peer-to-peer connection.
In the given Figure 6.2 we can also observe that different devices might run into higher or lower latencies when consuming the video stream, even though their physical distance is equivalent. In particular, the phone device (green bar) had an initially very high latency; the yellow bar was a special case because the consumer was executed on the same device as the SFU, which yielded the lowest latency with an average of 4ms.

Catching up with the example of an AR headset consuming the video stream, we claim that the smartphone could best be compared to the headset. We add its 6ms to the mean latency of 4ms for the producer and end up with 10ms that the mere transfer of the video frames requires in our edge setup. By the abovementioned reasoning, this gives us a margin of 10-20ms for the processing of privacy models in the SFU. It remains to discuss what types of video streams (e.g. resolution and frame rate) and which privacy models we can process within that time frame. This is still a native benchmark that we would like to comply with. For other use cases, we might very well tolerate significantly higher latencies, but for now, we want to discuss further whether we can stay within this boundary.

7.1.2 Privacy Model Processing

The actual performance of the SFU is visible for the first time in Figure 6.4, where we can obtain numerous details about the processing of *Video #1* and all 4 evaluated privacy models. The stream resembles a webcam input with a resolution of 1280x720px and a frame rate of 30, which could contain sufficient details for a surveillance camera or even a head-mounted camera. 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 peaks in execution time, which were mostly owed 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 was entirely dependent on the result of the Face_Trigger, as we can observe from the box plot in Figure 6.6. We could not detect any impact on the overall performance depending on whether we applied the Blur_Area_Pixelate or Fill_Area_Box function; both had a negligible share ($\sim$ 0ms) of the model's average performance with barely any outliers.

The other two models, *#3 & #4*, would not present such a stable latency over the full course of the stream. While the Face_Trigger maintained a mostly stable latency of around 15ms for these models, the Age- and Gender_Trigger on the other hand were the source of numerous peaks in the processing time. The frequency of how these peaks occur forms almost a periodical pattern in the first two thirds of the measurement, it seems to alternate whether the overall latency is increased by the second trigger function. This is completely different to the results of *Video #1* in Figure 6.9, where the second trigger was a comparably constant factor when processing streams without GPU. A notable factor is also that the overall processing time almost doubled in the last third of the stream, when 3 new faces were introduced (as visualized in Figure 6.3). We reason thus, that the Age- and Gender_Trigger are in fact dependent on the number of faces shown in the video frame. This limits the applicability of these triggers for scenes, where a multitude of faces are to be shown and anonymized.
If we recall that we could tolerate a mean latency of 10-20ms for our scenario using AR glasses, we would already get close to the cap with mean latencies of around 15ms for our first and fourth model. Within these constraints, the other two models would no longer

be applicable. We will show now that we can decrease the processing time significantly by decreasing the quality of the video stream.

Therefore, we compare the results for processing the first video (Figure 6.4) to the performance of the second video in Figure 6.5. The difference between these two in terms of video specifications is the resolution and frame rate. The second video, which has a resolution of 640x320px, would contain 4.5 times fewer pixels in a single frame than the first video. We notice that the mean latency for processing *Video #2* with *Model #1* and *#4* dropped drastically to a value close to 0ms. To be more precise here, we would have to increase the scale on the y-axis. Only in sporadic cases would the Face_Trigger cause the performance to run into peak values of around 10-15ms, which is very different from the image we get from *Model #2* and *#3*. For these two models, which used an additional trigger function, we got a surprisingly stable latency dominated by this second trigger function. This is totally different from the results for *Video #1*, where the Age- and Gender_Trigger would only show their impact in occasional peak values, not as a predominant mean value over the full course.
The overall performance dropped drastically for the first and fourth model, whereas for the second and third model we did not observe such extreme improvements. This is well visible in Figure 6.6, where the second and third model would decrease from a mean performance of 25ms to 15ms by switching to the lower-resolution *Video #2*.

The same impression also emerges from Figure 6.7, where we observe how the overall performance of *Model #1 & #2* (red and blue line) dropped continuously by switching to a lower video resolution. Under these circumstances, we could definitely use these two models for our use case with the AR glasses, where our global latency would have to stay somewhere between 20-30m. However, the question is whether the resolution of the video stream would still be sufficient for the given use case. Regardless, we have shown that we can optimize the performance by varying the quality of the stream. We might do that as long as the stream still fulfills the requirements for video resolution, assuming there are such.

7.1.3 Processing Environment & Tuning

As an example of how we can tune parameters, we varied the number of blocks in the Blur_Area_Pixelate function to investigate whether different parameter settings can have an effect on the model's performance. Figure 6.8 contains the results for processing *Video #1* and *Model #1* with 1 or 50 blocks. We remember from Figure 4.8 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 on average similar results to applying Blur_Area_Pixelate with 5 blocks in Figure 6.4. However, a notable fact is that the streaming session with 5 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 main question, whether a higher *blocks*

parameter produces a higher processing time, can be clearly answered. There is a significant difference in the two subgraphics. Firstly, the Blur_Pixelate grew to become a factor with around 20ms in the overall performance. Secondly, it showed an increasingly high load during the section of the video where multiple faces are visible, whereas in previous research we had not detected that this function is influenced by multiple faces. We conclude that an incorrectly set parameter value might very well blast an initially efficient privacy chain. In our case, we recorded a mean performance below 15ms with 1 block. With an increasingly high parameter, we almost doubled the mean processing time and ran into undesired peaks. Thus, parameters can be a critical factor for the performance of the privacy model.

We wanted to ascertain whether we can accelerate the processing significantly by providing a GPU for processing the ONNX models, as this would put the selection of edge devices further into focus. In Figure 6.9 we presented the results for processing *Video #1* without GPU-acceleration. Under these circumstances, *Model #1 & #4* would both process the streamed *Video #1* with a mean latency of around 30ms, twice as much as with GPU-support. However, there were fewer high peaks in the processing time, generating a fairly stable streaming latency. The performance of *Model #2* and *#3* shows a tremendous difference between the setup with and without GPU. While we would measure a mean processing time of around 25ms with GPU support, the processing would rocket upwards to around 150ms on average with no GPU being available for processing.
By providing an environment with GPU support, we decreased the processing time for these two models by a factor of 6. Without GPU-acceleration we would have come nowhere close to a latency that facilitates real-time use cases such as the AR glasses. The gap between the two setups is more than significant and encourages us to dive into how to accelerate the processing of certain ONNX models. However, from the given results, we cannot provide a constant factor by which the processing improves by switching between these two environments. We have seen that this is very dependant on the functions that are applied in the privacy model. For example, the Age- and Gender_Trigger would shrink 3 times more than the Face_Trigger by using a GPU setup.

This brings us to the conclusion that upgrading an edge device with an advanced computation environment can be a comparably big factor for accelerating the processing of data frames. Nevertheless, not all functions benefit equally from that environment, some not at all, like the transformation functions in our prototype. Whether it makes sense to undergo a (potentially costly) upgrade depends entirely on the privacy model that is to be deployed and the functions that it contains. Thus, it needs equal evaluations for upcoming trigger and transformation functions in combination with different types of data streams to give PMs evidence on how to optimize their processing chain.

7.2 Processing Boundaries

Until now, we have focused on interpreting the measured values that we presented in Chapter 6. In this section, we will now advance a step towards discussing the stream's

frame rates and how they define the boundaries of processing time. Depending on the number of frames that we transfer per second, minute, or hour, the available time for computation changes accordingly.

Our producer client streamed two types of videos in our evaluated scenarios, one with a FPS of 30 and the other with 16 FPS. The FPS in fact had no impact in regards to how long the processing of video frames took. The differences we recorded between Figure 6.4 and 6.5 were caused by the decreased video resolution. What changes, however, is the time that the computation of video frames might take before frames are dropped. We recall from Section 4.3.1 that we implemented a frame queue, with its size corresponding to the incoming FPS.
When streaming *Video #1* over the producer, the SFU receives new video frames every 33ms. Our producer client sends 30 frames equally distributed over 1000ms, thus we end up with $1000/30 = 33ms$. In another thread, the SFU pulls new frames for processing from the frame buffer at the same frequency as the FPS, i.e., every 33ms. In the case that the processing of *Model #2* takes on average 25ms, the SFU would process the frame, forward it to consumers, and stay idle for 8ms out of 33ms. By default, the frame queue is empty, but if we suppose that we have a sudden peak of 49ms in processing, the SFU will be occupied for longer than desired, and the incoming thread will place a new frame in the frame buffer while the processing of the old video frame is still ongoing. After finishing the processing of 49ms, the SFU immediately starts to process the old frame that is present in the queue, which takes 26ms. Nevertheless, during that time, another frame comes in, which needs to be processed. We overcome this issue by gradually achieving results below 33ms in processing, which empties the frame queue. This is the scenario if we have a mean latency lower than the frequency at which we receive new frames, which allows us to compensate for occasional peak values without losing video frames. However, if we have a mean processing time that always exceeds the time frame we have, we will start to lose data.

For now, we assume that the processing of our privacy model takes around 30ms, as for example for *Model #4* without GPU support in Figure 6.9. There is no issue if we stream at a frame rate of 30 FPS since the SFU might take up to 33ms for processing until a new frame comes in. If we would like to stream a video with 60 FPS, we would decrease the time available for processing to 16ms, and thus insert on average around 2 frames into the frame queue while only consuming one. As soon as the frame queue fills up, the SFU consumes one frame every 30ms, but on the other hand inserts 2 every 33ms, which will in most cases replace one existing frame. Since we only finish the processing of a frame every 30ms we would end up with a frame rate of 33 FPS that is streamed to connected consumers, while the remaining 27 frames are dropped every second. In this case, the consequence is dropping a stream's frame rate from 60 to 33 FPS, which might not be dramatic in some cases. In contrast, imagine the impact if the mean processing time takes on average 60 or even 120ms and the frame rate is still 30 FPS. Does the resulting stream still serve its purpose with only 16 or 8 FPS?

Even if it is acceptable to have the stream's FPS decreased by the SFU, we would recommend decreasing the producer's frame rate as soon as the stream starts to drop frames. In the event that the frame buffer is filled, the SFU will drop the oldest frames from the frame buffer, which are, after all, still arbitrary frames of the video stream. By decreasing the frame rate willingly, we are aware of this issue and decide on a lower stream quality instead of dropping frames by chance. A naive rule to decide on an applicable FPS is $1000ms \ / \ processing\ time = possible\ FPS$. Together with a little margin upwards for peak performances, the SFU should be able to process a mean processing time close to that value. During development, we implemented monitoring utilities for the produced and consumed performance in the SFU through the console. Here it requires advanced tools to give developers and PMs more detailed feedback on the live performance of privacy models.

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 6.4 and 6.5. 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. This is strongly related to the adaptive bitrate we discussed in Section 4.1, which is an advanced mechanism to react to insufficient performance.

7.3 Proof of Concept

In this thesis, we started with a motivating example in Section 1.2 and presented in Section 4.1 an abstract framework of how to transform streams on the edge. Afterwards, we decided on an appropriate programming environment in which we would eventually implement the framework. To provide information about the framework's performance, we presented how to evaluate it in Chapter 5; Chapter 6 would contain the respective results, and this chapter their interpretation. Since the presentation of the abstract concept, the remaining work has been dedicated to evaluating the proof of concept of the framework, which we will conclude right here.

We will not provide a full list of all the features that the initial concept included, but rather focus on the ones that were not (sufficiently) implemented from our point of view and that need to be addressed further in future work:

- In the presented concept, we wanted to re-deploy privacy models and new trigger and transformation functions without any impact on ongoing streaming connections. We provided the means to redeploy privacy models on the go without impacting on active peer connections. However, ONNX models exceeded our expectations in regards to size and programmatic complexity. Every ONNX model needs a specific input format for the video frame, which needs to be adjusted precisely for every function. We consider it still possible to update or add entirely new functions and transformations during live performance, but we would need to address that in more detail in future work if we see sufficient demand for it.

- We discussed in the concept the idea of lambda functions that would not maintain a state, which builds up to a chain of static functions where we pass parameters (e.g., video frames) between them. In addition, we wanted to provide temporary storage to keep track of recurring patterns, for example, to ensure k-anonymity by using structured attributes. Thus, functions would not be truly static anymore, even though temporary storage can be provided through the application context. We ultimately did not implement such a storage because the functions and use cases we developed did not necessitate it, but future research may very well push us in this direction.

- The initial concept was not limited to multimedia streams; we wanted to keep it general enough to support a wide range of data types that can be streamed and transformed through our streaming pipeline. Throughout the thesis, we developed the example of video streams further and further because it provides a high variety of ONNX models and simple use cases that helped us narrow down the aspects of the prototype. In Section 4.3.2, we showed that it is equally possible to stream audio frames over the SFU and detect whether they contain human speech. However, we did not evaluate it further due to a lack of further analysis methods. To prove that our framework is equally applicable for primitive data types (e.g., integer streams), we will address these fields further in future research.

After developing and evaluating our prototype, the last part that remained to show for the proof of concept was the deployment of the SFU on our edge device of choice: the NVIDIA Jetson. This last step was documented in Section 6.3, where we listed the required steps to configure the device and the ONNX runtime environment. We did not evaluate the performance on the NVIDIA Jetson; however, in Figure 6.10 we visualized the transformed stream after enforcing *Model #1* on the webcam input. The resulting stream would equally contain anonymized faces and run stable with 30 FPS, so we see the deployment on the Jetson as having succeeded.

The results we obtained from evaluating our solution provided us with a variety of information about the prototype's performance. We could document that certain privacy models (e.g. *Model #1* & *#2*) would achieve a stable latency of around 15 ms when processed with GPU support in their current state. This would allow us to stream and transform videos at up to 60 FPS while keeping sufficient overhead for peak times. On the contrary, *Model #3* & *#4* would run into comparably large processing times when applied with/without GPU and thus drop the streams frame rate undesirably.
To sum up, in the evaluated state, we see certain combinations of privacy models and processing environments that are very well applicable in live situations. However, without advanced monitoring capabilities, it is difficult to detect if a model exceeds the given time frame for processing. This will be improved through future work; in fact, the setup we established for evaluating processing times already provides most of the required functionality for a live monitoring unit.

CHAPTER 8

Conclusion

We have presented a privacy-enforcing framework that describes how data streams are transformed on edge networks, close to the IoT device producing the data. The two primary components of the proposed work were how to specify policies in a privacy model through chained lambda functions and how to enforce these rules on streamed data. Within our architecture, data is provided and consumed through a SFU, which is responsible for routing data and, most importantly, analyzing and transforming frames according to a privacy model. Due to the absence of a central cloud server, we have a small number of peers handling the stream, which considerably reduces the chance of having the data intercepted. Additionally, as soon as the stream leaves the SFU, the potential damage of having data leaked is significantly lower since information that is considered confidential has already been removed.

To answer whether our framework can be used for transforming live streams, we developed a prototype dedicated to video streaming. Results showed, that solely by moving the SFU from the cloud to a device in the producer's vicinity, we decreased the streaming latency between SFU and producer from 28ms to 4ms. We used four privacy models to evaluate the performance of the SFU; within the privacy models, we varied the quality and the submission speed of the stream - decreasing the video resolution would drastically speed up the processing, while decreasing the frame rate would extend the time available for processing. In the given state, our prototype was able to detect and blur human faces within 15ms for a video resolution of 720p, we might thus stream and transform content at up to 60 FPS without dropping any frame. To accelerate the processing of video frames, we further evaluated the usage of GPU-resources. Any setup with a GPU would clearly outperform the one without. We concluded the proof of concept by deploying the SFU on the NVIDIA Jetson Xavier NX, where we accessed the underlying GPU-resources to transform video streams.

8.1 Research Questions

- RQ. 1: How can privacy requirements for edge networks be specified and represented as privacy models?

  We introduced the notation of trigger and transformation functions as a means to model cause-effect relations: whenever we detect a certain privacy-violating pattern within a frame (trigger), we enforce privacy by manipulating the frame content accordingly (transformation). These functions can be linked together, forming increasingly longer chains where the next link is only processed if the one before has come true.

  The question at hand is what kind of privacy requirements can be specified with this logic; is it sufficient to represent enterprises' privacy agreements? We provided a couple of possible use cases that were based on detecting faces, genders, car plates, etc. and blurring the respective area. This can, in fact, be sufficient for surveillance cameras that are installed in traffic junctions or offices, but complex privacy requirements would hardly be manageable with this logic. The key factor here is that we are heavily bound to the trigger functions that we have at our disposal for analyzing frames, in our prototype we applied pre-trained ONNX-models for detecting patterns. If we would have liked to model more advanced use-cases, we would either have to find a suitable DNN-model or train one ourselves; since most trained models (e.g. PyTorch or Tensorflow) can be converted to ONNX models, we would not have to extend the framework with a new runtime environment.

  We did not investigate a desirable automatic conversion of privacy agreements into privacy models because we focused on simple use cases that could be converted manually. Once future research arrives at such a state, the presented framework can be extended with new trigger and transformation functions to support the resulting privacy models.

- RQ. 2: What architecture is the most efficient for the distribution execution of privacy models?

  In this thesis, we presented a framework for transforming streams on the edge according to privacy models. The framework's architecture consisted of a single producer that streamed data to the SFU, which is responsible for routing the incoming stream and, most importantly, analyzing and transforming the stream according to the active privacy model. The resulting stream can be received by a unrestricted number of consumer devices. We assumed that the SFU was deployed on the edge device, and producer and consumer clients would be executed on other devices in the network.

  We evaluated the presented architecture in different setups and it showed that the latency between SFU and producer would drop drastically by moving computational resources from the cloud to the edge. In regards to latency, we see the edge setup as clearly superior, but there are lots of improvements one can undertake to

improve the architecture. For instance, in our presented prototype we evaluated the transformation of video streams; to accelerate the processing of privacy models we investigated the usage of graphic processors. The results clearly showed a drastic speed-up when harnessing GPU resources. However, the questions of which architecture is the "most efficient" is very dependent on the use case Audio or integer streams might, for example, not at all benefit from a GPU.

Performance, which is naturally related to efficiency, can be evaluated through a variety of metrics. We selected those that we considered the most relevant (latency & performance of SFU) and came up with clear results for the evaluated environments. Those results are clearly part of the answer, but the efficiency of the architecture is influenced by a wider range of factors that will have to be addressed in future research.

- RQ. 3: Is the presented architecture in fact able to transform a data stream according to a privacy model within a respective time span?

  In this case we can give a clear answer - yes. We conducted a variety of evaluations for our prototype: we varied the privacy model, the quality of the video stream, the SFU's destination, the accessibility/absence of GPU-resources, and the effect of function's parameters. Every combination of these factors yielded to a (slightly) different performance of the SFU, some of them (e.g. high frame rate and no GPU support) would run into latencies that would exceed the time frame for computation by a multiple. Nevertheless, within the results we would also find a large share of combinations that were well applicable in our experimental setup - signifying that we would process video frames within the available time frame. The decisive factor here is that the processing of frames can be carried out before a new video frame is received, which would otherwise inevitably lead to dropping frames.

  The answer we provided was based on the results we measured for our exact experimental setup. Depending on the available environment and the privacy model, the outcome might be different. If we notice that we cannot process the frame within the given time frame, we can decrease the streams quality: most notably for video streams was that decreasing the video resolution would drastically reduce the processing time, whereas decreasing the FPS would extend the available time frame for processing. This in mind, we have a set of parameters through which we can influence the performance of the SFU. However, the most important choice will maintain the selection of the edge device on which we deploy the SFU. Devices that do not provide sufficient computation power will preclude most use cases in advance.

8.2 Future Work

We consider this paper an initial step towards a general framework for specification and operation of privacy models for data streams on the edge. The evaluation of our prototype provided promising results for our approach, nevertheless, it led to a multitude of open challenges that should be addressed in future work:

- Our presented framework is designed to work independently of the transferred data type, so we should equally be able to stream and transform video-, audio-, or integer streams. Through our prototype, we proved this concept for video streams, whereas we decided not to evaluate privacy models further for audio streams, at least not in this work. This is what remains to show for future work, where we will extend and evaluate our prototype with other data types.
- Within the conducted evaluation, we varied a multitude of parameters that influenced the performance of the SFU. However, we would only evaluate the performance on one device; deploying the SFU on the NVIDIA Jetson served at this point only to conclude the proof of concept. To emphasize that the selection of the edge device is a decisive factor for the performance we will have to evaluate devices with unlike computational power, thus giving evidence of what type of device can process what kind of privacy model.
- We evaluated the performance of the presented trigger and transformation functions. However, we cannot evaluate these and coming functions on every type of edge device to give evidence of the privacy models that are feasible to process. To that extent, we require a live monitoring component that provides the PM with information about whether the current privacy chain runs into performance problems and what share the individual applied functions have in that overall performance.
- In our architecture, the SFU, which was deployed on exactly one edge device, is responsible for carrying out trigger and transformation functions. What we did not discuss until now is whether we can scale up the performance vertically/horizontally through a higher number of threads or a virtualization layer that processes requests internally over a multitude of physical devices. This, and many more ideas, can be pursued to improve the performance.
- The aim of our work was to improve privacy in continuous data streams, however, there are numerous aspects in regards to security that we have not discussed in our work by now. For instance, how to authorize PM when submitting new privacy models or even new lambda functions, how to ensure the integrity of a received privacy model, how to secure the SFU to avoid malicious providers connecting, and so forth. Those questions and more are the focus of future work.

List of Figures

1.1 Transforming streamed data according to a privacy model. 3

2.1 Network layers in a WebRTC live-streaming setup [17] 9
2.2 Concept of privacy regulations for services providers and end-users [23] . . 12

3.1 Access-control restrictions for sensor data on the edge (adapted from [6]) 16

4.1 Defining triggers and transformations for the privacy model. 22
4.2 Conceptual framework for transforming streams with privacy models . . . 24
4.3 Establishing peer connections between provider, SFU, and consumer . . . 26
4.4 Converting and executing ONNX models for pattern detection 28
4.5 Analyzing audio frames to replace human voice with white noise 31
4.6 Graphical representation of a privacy chain for blurring faces 33
4.7 Detecting and anonymizing faces of humans with age between 25-32 . . . 36
4.8 Blurred face with a number of $1 * 1$, $5 * 5$, or $30 * 30$ blocks calculated . . 38

5.1 Overview of the metrics evaluated for the presented prototype 46
5.2 Evaluated privacy models in different colors (orange, purple, green, and grey) 49
5.3 Streaming architecture for evaluating the prototype 51

6.1 Producer latency over 60s for cloud and edge setups with mean values . . 54
6.2 Latency of different consumer devices over 60 seconds with mean values . 55
6.3 Resulting stream with multiple faces transformed 56
6.4 Individual performance of functions for *Video #1* and the 4 privacy models 57
6.5 Individual performance of functions for *Video #2* and the 4 privacy models 58
6.6 Statistical distribution of function's performances for *Video #1* & *# 2* . . 58
6.7 Overall performance of evaluated privacy models on *Video #1* & *#2* . . . 59
6.8 Performance of Blur_Area_Pixelate for *blocks* parameter values {1, 50} 60
6.9 Individual performance of functions for *Video #1* without GPU-acceleration 60
6.10 Consuming a stream after transformation on the NVIDIA Jetson Xavier NX 62

List of Tables

4.1 API interface for provider client 27
4.2 ONNX models used for detecting privacy-violating patterns 29
4.3 Overview of OpenCV transformation functions 29
4.4 Overview of configurable arguments for the SFU 41
4.5 API interface for SFU server 42
4.6 Software requirements for the programming environment 43

5.1 List of two recorded videos that are used to evaluate the prototype 50
5.2 Hardware specifications of the device used to evaluate the prototype . . . 50

Bibliography

[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, October 26-28, 2021*, F. J. D. Mayo, M. Marchiori, and J. Filipe, Eds., SCITEPRESS, 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] W. Shi and S. Dustdar, "The promise of edge computing", *Computer*, vol. 49, no. 5, pp. 78–81, 2016. DOI: 10.1109/MC.2016.145.

[4] O. Alvear, C. T. Calafate, J.-C. Cano, and P. Manzoni, "Crowdsensing in smart cities: Overview, platforms, and environment sensing issues", *Sensors*, vol. 18, no. 2, p. 460, 2018.

[5] A. K. Massey, J. Eisenstein, A. I. Antón, and P. P. Swire, "Automated text mining for requirements analysis of policy documents", in *2013 21st IEEE International Requirements Engineering Conference (RE)*, ISSN: 2332-6441, Jul. 2013, pp. 4–13. DOI: 10.1109/RE.2013.6636700.

[6] 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.

[7] NVIDIA, *Embedded Systems for Next-Gen Autonomous Machines*, en-us, 2021. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/ (visited on Jan. 2, 2022).

[8] N. Kasim, N. Rahman, Z. Ibrahim, and N. N. Abu Mangshor, "Celebrity Face Recognition using Deep Learning", *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 12, pp. 476–481, Nov. 2018. DOI: 10.11591/ijeecs.v12.i2.pp476-481.

[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] S. Dustdar and I. Murturi, "Towards Distributed Edge-based Systems", in *2020 IEEE Second International Conference on Cognitive Machine Intelligence (CogMI)*, IEEE, 2020, pp. 1–9.

[12] L. Sanabria-Russo, D. Pubill, J. Serra, and C. Verikoukis, *IoT Data Analytics as a Network Edge Service*, eng, Conference Name: IEEE International Conference on Computer Communications, Paris (France), Apr. 2019. DOI: 10.1109/INFCOMW.2019.8845207.

[13] M. Satyanarayanan, "The Emergence of Edge Computing", *Computer*, vol. 50, no. 1, pp. 30–39, Jan. 2017, Conference Name: Computer, ISSN: 1558-0814. DOI: 10.1109/MC.2017.9.

[14] 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.

[15] S. Nittel, "Real-time sensor data streams", en, *SIGSPATIAL Special*, vol. 7, no. 2, pp. 22–28, Sep. 2015, ISSN: 1946-7729. DOI: 10.1145/2826686.2826691.

[16] WebRTC, *Real-time communication for the web*, en, 2021. [Online]. Available: https://webrtc.org/ (visited on Jan. 2, 2022).

[17] M. Iyengar, *WebRTC Architecture Basics: P2P, SFU, MCU, and Hybrid Approaches*, en, Mar. 2021. [Online]. Available: https://medium.com/securemeeting/webrtc-architecture-basics-p2p-sfu-mcu-and-hybrid-approaches-6e7d77a46a66 (visited on Feb. 7, 2022).

[18] J. R. Getta and E. Vossough, "Optimization of data stream processing", *ACM SIGMOD Record*, vol. 33, no. 3, pp. 34–39, Sep. 2004, ISSN: 0163-5808. DOI: 10.1145/1031570.1031577.

[19] E. Mehmood and T. Anees, "Challenges and Solutions for Processing Real-Time Big Data Stream: A Systematic Literature Review", *IEEE Access*, vol. 8, pp. 119 123–119 143, 2020, Conference Name: IEEE Access, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3005268.

[20] C. Dwork and A. Roth, "The Algorithmic Foundations of Differential Privacy", en, *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3-4, pp. 211–407, 2013, ISSN: 1551-305X, 1551-3068. DOI: 10.1561/0400000042.

[21] H. Federrath and A. Pfitzmann, "Gliederung und Systematisierung von Schutzzielen in IT-Systemen", de, *Datenschutz und Datensicherheit*, vol. 24, 2000.

[22] H. Zwingelberg and M. Hansen, "Privacy Protection Goals and Their Implications for eID Systems", en, in *Privacy and Identity Management for Life*, J. Camenisch, B. Crispo, S. Fischer-Hübner, R. Leenes, and G. Russello, Eds., ser. IFIP Advances in Information and Communication Technology, Berlin, Heidelberg: Springer, 2012, pp. 245–260, ISBN: 978-3-642-31668-5. DOI: 10.1007/978-3-642-31668-5_19.

[23] N. Gol Mohammadi, J. Leicht, N. Ulfat-Bunyadi, and M. Heisel, "Privacy Policy Specification Framework for Addressing End-Users' Privacy Requirements", en, in *Trust, Privacy and Security in Digital Business*, S. Gritzalis, E. R. Weippl, S. K. Katsikas, G. Anderst-Kotsis, A. M. Tjoa, and I. Khalil, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2019, pp. 46–62, ISBN: 978-3-030-27813-7. DOI: 10.1007/978-3-030-27813-7_4.

[24] M. Pollmann and D.-K. Kipker, "Informierte Einwilligung in der Online-Welt", de, *Datenschutz und Datensicherheit - DuD*, vol. 40, no. 6, pp. 378–381, Jun. 2016, ISSN: 1614-0702, 1862-2607. DOI: 10.1007/s11623-016-0618-6.

[25] Official Journal of the European Union L 119, "Regulation 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC", en, pp. 1–88, May 2016. [Online]. Available: http://eur-lex.europa.eu/legal-content/%20EN/TXT/?uri=OJ:L:2016:119:TOC (visited on Mar. 11, 2022).

[26] C. Kalloniatis, E. Kavakli, and S. Gritzalis, "Methods for Designing Privacy Aware Information Systems: A Review", Jan. 2009, pp. 185–194. DOI: 10.1109/PCI.2009.45.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] T. Wang, Y. Mei, W. Jia, X. Zheng, G. Wang, and M. Xie, "Edge-based differential privacy computing for sensor–cloud systems", en, *Journal of Parallel and Distributed Computing*, vol. 136, pp. 75–85, Feb. 2020, ISSN: 0743-7315. DOI: 10.1016/j.jpdc.2019.10.009.

[32] 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.

[33] Y. Mao, J. Feng, F. Xu, and S. Zhong, "A Privacy-Preserving Deep Learning Approach for Face Recognition with Edge Computing", *HotEdge*, 2018.

[34] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Zomaya, *Edge Intelligence: The Confluence of Edge Computing and Artificial Intelligence*. Sep. 2019.

[35] 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.

[36] 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.

[37] X. Cao and S. Madria, *Efficient Geospatial Data Collection in IoT Networks for Mobile Edge Computing*. Oct. 2019. DOI: 10.1109/NCA.2019.8935061.

[38] P. Bellavista, D. Belli, S. Chessa, and L. Foschini, "A Social-Driven Edge Computing Architecture for Mobile Crowd Sensing Management", *IEEE Communications Magazine*, vol. 57, no. 4, pp. 68–73, Apr. 2019, Conference Name: IEEE Communications Magazine, ISSN: 1558-1896. DOI: 10.1109/MCOM.2019.1800637.

[39] Y. Du, "Collaborative Crowdsensing at the Edge", Ph.D. dissertation, Jul. 2020.

[40] Z. Chen, C. Fiandrino, and B. Kantarci, "On blockchain integration into mobile crowdsensing via smart embedded devices: A comprehensive survey", en, *Journal of Systems Architecture*, vol. 115, p. 102 011, May 2021, ISSN: 1383-7621. DOI: 10.1016/j.sysarc.2021.102011.

[41] M. Marjanović, A. Antonić, and I. P. Žarko, "Edge Computing Architecture for Mobile Crowdsensing", *IEEE Access*, vol. 6, pp. 10 662–10 674, 2018, Conference Name: IEEE Access, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2799707.

[42] M. Li, Y. Li, and L. Fang, "ELPPS: An Enhanced Location Privacy Preserving Scheme in Mobile Crowd-Sensing Network Based on Edge Computing", in *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, ISSN: 2324-9013, Dec. 2020, pp. 475–482. DOI: 10.1109/TrustCom50675.2020.00071.

[43] F. Khan, A. Ur Rehman, J. Zheng, M. A. Jan, and M. Alam, "Mobile crowdsensing: A survey on privacy-preservation, task management, assignment models, and incentives mechanisms", en, *Future Generation Computer Systems*, vol. 100, pp. 456–472, Nov. 2019, ISSN: 0167-739X. DOI: 10.1016/j.future.2019.02.014.

[44] R. Rivest and D. M. Zissman, "Smartphone-Assisted, Privacy-Preserving COVID-19 Contact Tracing", vol. Massachusetts Institute of Technology, no. Infection Prevention and Control, Protection, p. 1, 2020.

[45] H. R. Schmidtke, "Location-aware systems or location-based services: a survey with applications to CoViD-19 contact tracking", en, *Journal of Reliable Intelligent Environments*, vol. 6, no. 4, pp. 191–214, Dec. 2020, ISSN: 2199-4668, 2199-4676. DOI: 10.1007/s40860-020-00111-4.

[46] M. A. Alsahli, A. Alsanad, M. M. Hassan, and A. Gumaei, "Privacy Preservation of User Identity in Contact Tracing for COVID-19-Like Pandemics Using Edge Computing", *IEEE Access*, vol. 9, pp. 125 065–125 079, 2021, Conference Name: IEEE Access, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3110762.

[47] D. Chatzopoulos, S. Gujar, B. Faltings, and P. Hui, "Privacy Preserving and Cost Optimal Mobile Crowdsensing Using Smart Contracts on Blockchain", in *2018 IEEE 15th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, ISSN: 2155-6814, Oct. 2018, pp. 442–450. DOI: 10.1109/MASS.2018.00068.

[48] L. Ma, X. Liu, Q. Pei, and Y. Xiang, "Privacy-Preserving Reputation Management for Edge Computing Enhanced Mobile Crowdsensing", *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 786–799, Sep. 2019, Conference Name: IEEE Transactions on Services Computing, ISSN: 1939-1374. DOI: 10.1109/TSC.2018.2825986.

[49] M. Kadadha, H. Otrok, R. Mizouni, S. Singh, and A. Ouali, "SenseChain: A blockchain-based crowdsensing framework for multiple requesters and multiple workers", en, *Future Generation Computer Systems*, vol. 105, pp. 650–664, Apr. 2020, ISSN: 0167-739X. DOI: 10.1016/j.future.2019.12.007.

[50] A. Manna, A. Sengupta, and C. Mazumdar, "EDPRL: A Language for Specifying Data Privacy Requirements of Enterprises", in *2019 Second International Conference on Advanced Computational and Communication Paradigms (ICACCP)*, Feb. 2019, pp. 1–6. DOI: 10.1109/ICACCP.2019.8882883.

[51] J. Vilela, J. Castro, L. E. G. Martins, T. Gorschek, and C. Silva, "Specifying Safety Requirements with GORE languages", in *Proceedings of the 31st Brazilian Symposium on Software Engineering*, ser. SBES'17, New York, NY, USA: Association for Computing Machinery, Sep. 2017, pp. 154–163, ISBN: 978-1-4503-5326-7. DOI: 10.1145/3131151.3131175.

[52] R. Joshaghani, S. Black, E. Sherman, and H. Mehrpouyan, "Formal specification and verification of user-centric privacy policies for ubiquitous systems", in *Proceedings of the 23rd International Database Applications & Engineering Symposium*, ser. IDEAS '19, New York, NY, USA: Association for Computing Machinery, Jun. 2019, pp. 1–10, ISBN: 978-1-4503-6249-8. DOI: 10.1145/3331076.3331105.

[53] H. Alshareef, S. Stucki, and G. Schneider, "Transforming Data Flow Diagrams for Privacy Compliance (Long Version)", *arXiv:2011.12028 [cs]*, Nov. 2020.

[54] 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.

[55] NVIDIA, *DeepStream SDK*, en-US, 2021. [Online]. Available: https://developer.nvidia.com/deepstream-sdk (visited on Jan. 2, 2022).

[56] OpenCV, *OpenCV*, en-US, 2021. [Online]. Available: https://opencv.org/ (visited on Jan. 13, 2022).

[57] Mozilla, *Using WebRTC data channels - Web APIs*, en-US, 2021. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Using_data_channels (visited on Jan. 1, 2022).

[58] aiortc, *About aiortc*, en, 2021. [Online]. Available: https://aiortc.readthedocs.io/en/latest/ (visited on Jan. 13, 2022).

[59] mediasoup, *mediasoup v3*, original-date: 2014-12-12T12:00:36Z, Feb. 2022. [Online]. Available: https://github.com/versatica/mediasoup (visited on Feb. 8, 2022).

[60] node-webrtc, *node-webrtc/node-webrtc*, original-date: 2013-07-02T18:21:45Z, Feb. 2022. [Online]. Available: https://github.com/node-webrtc/node-webrtc (visited on Feb. 8, 2022).

[61] aiortc, *API Reference*, 2021. [Online]. Available: https://aiortc.readthedocs.io/en/latest/api.html (visited on Feb. 7, 2022).

[62] Mozilla, *WebRTC API - Web APIs | MDN*, en-US, 2021. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/ (visited on Feb. 8, 2022).

[63] PyAudio, *PyAudio: PortAudio v19 Python Bindings*, 2022. [Online]. Available: http://people.csail.mit.edu/hubert/pyaudio/ (visited on Feb. 8, 2022).

[64] FFmpeg, *About FFmpeg*, 2022. [Online]. Available: https://www.ffmpeg.org/ (visited on Feb. 8, 2022).

[65] PyTorch, *PyTorch About*, en, 2022. [Online]. Available: https://www.pytorch.org (visited on Feb. 8, 2022).

[66] Tensorflow, *TensorFlow About*, es-419, 2022. [Online]. Available: https://www.tensorflow.org/?hl=es-419 (visited on Feb. 8, 2022).

[67] ONNX, *ONNX About*, 2022. [Online]. Available: https://onnx.ai/ (visited on Feb. 8, 2022).

[68] ——, *Model Zoo for pretrained Models*, Feb. 2022. [Online]. Available: https://github.com/onnx/models (visited on Feb. 8, 2022).

[69] NVIDIA, *CUDA Toolkit*, en-US, Jul. 2013. [Online]. Available: https://developer.nvidia.com/cuda-toolkit (visited on Feb. 8, 2022).

[70] 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).

[71] asiryan, *Age and Gender Classification using Convolutional Neural Networks*, en, 2021. [Online]. Available: https://github.com/onnx/models (visited on Feb. 9, 2022).

[72] 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.

[73] winter2897, *Real-time Auto License Plate Recognition with Jetson Nano*, en, 2021. [Online]. Available: https://github.com/winter2897/Real-time-Auto-License-Plate-Recognition-with-Jetson-Nano (visited on Feb. 9, 2022).

[74] O.-P. Heinisuo, *opencv-python: Wrapper package for OpenCV python bindings.* 2021. [Online]. Available: https://github.com/skvark/opencv-python (visited on Feb. 9, 2022).

[75] 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).

[76] J. Wiseman, *wiseman/py-webrtcvad*, original-date: 2016-04-23T05:03:52Z, Feb. 2022. [Online]. Available: https://github.com/wiseman/py-webrtcvad (visited on Feb. 10, 2022).

[77] R. Python, *The Ultimate Guide To Speech Recognition With Python – Real Python*, en, 2021. [Online]. Available: https://realpython.com/python-speech-recognition/ (visited on Feb. 10, 2022).

[78] J. Balikuddembe, *Audio Recognition: Identify Songs with python*, en, Jul. 2019. [Online]. Available: https://medium.com/@baliksjosay/audio-recognition-identify-songs-with-python-551003c42f16 (visited on Feb. 10, 2022).

[79] aiortc, *Server Example*, Feb. 2022. [Online]. Available: https://github.com/aiortc/aiortc/blob/c8f0d63feca60c7c46ce2a23c38d017f400c9350/examples/server/server.py (visited on Feb. 16, 2022).

[80] Z. Kristóf, "International Trends of Remote Teaching Ordered in Light of the Coronavirus (COVID-19) and its Most Popular Video Conferencing Applications that Implement Communication", *Central European Journal of Educational Research*, vol. 2, pp. 84–92, Jul. 2020. DOI: 10.37441/CEJER/2020/2/2/7917.

[81] NVIDIA, *CUDA Installation Guide for Microsoft Windows*, en-us, 2022. [Online]. Available: https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/index.html (visited on Feb. 20, 2022).

[82] ——, *cuDNN Installation Guide for Microsoft Windows*, en-us, 2022. [Online]. Available: https://docs.nvidia.com/deeplearning/cudnn/install-guide/index.html (visited on Feb. 20, 2022).

[83] Docker, *Docker Hub - Container Image Library | Docker*, en, 2022. [Online]. Available: https://www.docker.com/products/docker-hub (visited on Mar. 15, 2022).

[84] Amazon, *Amazon EC2 T2-Instances – Amazon Web Services (AWS)*, de-DE, 2022. [Online]. Available: https://aws.amazon.com/de/ec2/instance-types/t2/ (visited on Mar. 15, 2022).

[85] Pandas, *Python Data Analysis Library*, 2022. [Online]. Available: https://pandas.pydata.org/ (visited on Feb. 28, 2022).

[86] M. Waskom, "seaborn: statistical data visualization", *Journal of Open Source Software*, vol. 6, no. 60, p. 3021, Apr. 2021, ISSN: 2475-9066. DOI: 10.21105/joss.03021. [Online]. Available: https://joss.theoj.org/papers/10.21105/joss.03021 (visited on Mar. 2, 2022).

[87] eLinux.org, *ONNX Runtime - Jetson Zoo*, 2021. [Online]. Available: https://elinux.org/Jetson_Zoo#ONNX_Runtime (visited on Mar. 2, 2022).

[88] B. Sterling, "John Carmack's Latency Mitigation Strategies", en-US, *Wired*, 2013, Section: tags, ISSN: 1059-1028. [Online]. Available: https://www.wired.com/2013/02/john-carmacks-latency-mitigation-strategies/ (visited on Mar. 3, 2022).

[89] Microsoft, *Microsoft HoloLens | Mixed-Reality-Technologie für Unternehmen*, de-de, 2022. [Online]. Available: https://www.microsoft.com/de-de/hololens (visited on Mar. 3, 2022).