

**BRNO UNIVERSITY OF TECHNOLOGY**  
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

**FACULTY OF INFORMATION TECHNOLOGY**  
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

**DEPARTMENT OF INTELLIGENT SYSTEMS**  
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

**DEEPCODE DETECTION FRAMEWORK**  
FRAMEWORK PRO DETEKCI DEEPCODES

**MASTER'S THESIS**  
DIPLOMOVÁ PRÁCE

**AUTHOR**  
AUTOR PRÁCE

**Bc. JAN BERNARD**

**SUPERVISOR**  
VEDOUCÍ PRÁCE

**Mgr. KAMIL MALINKA, Ph.D.**

**BRNO 2022**

# Master's Thesis Assignment



140642

Institut: Department of Intelligent Systems (UIT)  
Student: **Bernard Jan, Bc.**  
Programme: Information Technology and Artificial Intelligence  
Specialization: Cybersecurity  
Title: **Deepfake Detection Framework**  
Category: Security  
Academic year: 2022/23

Assignment:

1. Learn about deepfakes (voice and video). Explore the current state of deepfakes detection methods (voice and video).
2. Learn about the technologies needed to create web extensions and technologies for creating scalable server applications.
3. Learn about existing deepfake detection solutions (e.g. other commercial web browser plug-ins)
4. Design an extensible framework (server-client or client-only) for deepfakes detection (support for at least 3 detection methods (voice and video)). Design a web extension for deepfakes detection that will use this framework. The solution should support multiple browsers and allow the detection of displayed content and uploaded files.
5. Implement the tool according to the design.
6. Test the functionality and reliability of the resulting implementation. Perform testing on at least two independent publicly available deepfakes datasets.
7. Discuss usability, detection efficiency and possible extensions.

Literature:

Puspita Majumdar, Akshay Agarwal, Mayank Vatsa, and Richa Singh, "Facial retouching and alteration detection," in Handbook of Digital Face Manipulation and Detection, pp. 367–387. Springer, 2022  
FIRC Anton a MALINKA Kamil. The dawn of a text-dependent society: deepfakes as a threat to speech verification systems. In: Brno: Association for Computing Machinery, 2022

Requirements for the semestral defence:

Items 1 to 4.

Detailed formal requirements can be found at <https://www.fit.vut.cz/study/theses/>

Supervisor: **Malinka Kamil, Mgr., Ph.D.**  
Consultant: Ing. Anton Firc  
Head of Department: Hanáček Petr, doc. Dr. Ing.  
Beginning of work: 1.11.2022  
Submission deadline: 17.5.2023  
Approval date: 3.11.2022

## **Abstract**

Deepfake creation has improved a lot in recent times and hence is a dreaded menace to society. Deepfake detection methods have also responded with development, but there are still not enough good tools available to the general public. This work focuses on creating a deepfake detection framework that will be easily extended by other detection methods in the future, yet simple and accessible to the general public.

## **Abstrakt**

Tvorba deepfake se za poslední dobu velmi zlepšila a tudíž je obávanou hroznou pro společnost. Detekční metody odhalující deepfake také reagovali rozvojem, ale stále není široké veřejnosti dostupné dostatečné množství kvalitních nástrojů. Tato práce se zaměřuje na vytvoření frameworku na detekci deepfake, který bude jednoduše rozšířitlený dalšími detekčními metodami v budoucnu a přitom jednoduchý a dostupný široké veřejnosti.

## **Keywords**

deepfake, framework, deepfake detection, containerazation, web browser extension

## **Klíčová slova**

deepfake, framework, detekce deepfake, kontejnerizace, doplněk webového prohlížeče

## **Reference**

BERNARD, Jan. *Deepfake Detection Framework*. Brno, 2022. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Mgr. Kamil Ma-linka, Ph.D.

# Rozšířený abstrakt

Deepfake je modní slovo, které nemá ustálenou žádnou technickou definici. Jedná se o kombinaci slov, deep a fake. Deep referuje na techniku hlubokého strojového učení a fake zase na tvorbu falešných digitálních médií.

Tvorba deepfake se za poslední dobu velmi zlepšila, její využití je velké například ve filmové produkci. Dříve bylo nutné mít pro tvorbu falešných nebo jinak modifikovaných digitálních medií určité znalosti nebo schopnosti. Byly využívané nástroje jako GIMP, Adobe Photoshop nebo Adobe After Effects. Dnes existují nástroje poháněné umělou inteligencí, které umožňují tvorbu deepfake téměř komukoliv. Vytváření deepfake ovšem nemusí být použito pouze ke správným účelům. Tudíž může být považováno za obávanou hroznou pro společnost.

Jedná se o rychle rozvíjející se obor. Detekční metody odhalující deepfakes také reagovali rozvojem, ale stále není široké veřejnosti dostupné dostatečné množství kvalitních nástrojů. Problémem je velké množství různých technik pro tvorbu jako jsou face swap, morphing a mnohé další. Detekční metody jsou většinou schopné efektivně detektovat pouze jeden typ deepfake. Většina detekčních dnes využívá různé typy features extraktorů s SVM (support vector machines) klasifikátorem. Existují i více „exotické“ přístupy jakožto odhad srdeční frekvence z videa.

Proto vznikl záměr vytvořit framework, který umožňuje integraci různých detekčních metod. Tím by byl odbouraný problém detekce pouze jednoho typu a také by se tento framework mohl použít pro porovnávání s již existujícími detekčními metodami. Cílem této práce je tvorba právě takového frameworku.

V první části této práce naleznete obecné informace o deepfakes jako jsou potenciální hrozby a bezpečnostní rizika s popisem již úspěšných útoků, různé typy deepfakes pro hlas, obrázky a video. Byl také provedena analýza trhu s dostupnými detekčními nástroji.

V druhé části je popsán celý proces vývoje frameworku a klientské aplikace, která má umožnit práci s frameworkem i běžným uživatelům. Proveden byl i detailní návrh zabývající se hlavní funkcionalitou i podpornými systémy. Následně byl framework podle tohoto návrhu implementován a také byly integrovány některé open source detekční metody. Stejný proces byl aplikován i u tvorby klientské aplikace.

Framework je navržen jakožto mikroservice aplikace, která využívá kontejnerový orchestrátor Kubernetes. Vstupním bodem je ASP.NET Core REST API, které vytvoří žádost na zpracování a pošle ji skrze message broker Reuqest Processing jednotkám. Jednotky obsahují jednotlivé detekční metody a po vyhodnocení žádosti vrací výsledky zpět ASP.NET Core aplikaci. Framework umožňuje automatické škálování na základě počtu zaslaných žádostí, což zajistí vysokou dostupnost při relativně nízkých nákladech na provoz.

Pro otestování funkčnosti jsou vytvořeny tři testovací scénáře, které testují spolehlivost, spotřebu zdrojů a schopnost práce v zatížení. Byly také provedeny testy přesnost integrovaných detekčních metod. Framework byl nasazen do Azure AKS a na této instanci byly provedeny všechny testy. Celkově prošlo frameworkem při testování přes 2200 souborů z pěti různých deepfake datasetů jako jsou ASV2021, LJ Speech, WafeFake, CelebDF a FaceForensics++.

Výsledky byly vyhodnoceny a popsány na konci této práce. Přesnost jednotlivých metod pochybná a určitě by bylo přínosné vyzkoušet natrénovat metody na robustnějších datasech nebo zkusit integrovat jiné. Celkově se výsledky testů dají prohlásit za úspěšné, protože framework operoval spolehlivě několik desítek hodin a byl schopen zpracovat téměř veškeré zaslanné požadavky. Testy také ověřili schopnosti škálovatelnosti frameworku.

# **Deepfake Detection Framework**

## **Declaration**

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of Mgr. Kamil Malinka Ph.D. The supplementary information was provided by Ing. Anton Firc. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....  
Jan Bernard

May 15, 2023

## **Acknowledgements**

I would like to sincerely thank my supervisor Mgr. Kamil Malinka Ph.D. for all the advice and insightful comments. The same thanks go to consultant Ing. Anton Firc.

I also owe a debt of gratitude to my family, especially my parents, friends for their wonderful support throughout my studies.

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Security impacts of deepfakes</b>	<b>4</b>
2.1	Human capabilities of deepfake detection . . . . .	4
2.2	Potential risks and impacts . . . . .	5
2.3	Detection tools requirements . . . . .	7
<b>3</b>	<b>Types of deepfakes</b>	<b>8</b>
3.0.1	Neural networks . . . . .	8
3.0.2	Voice deepfakes . . . . .	9
3.0.3	Image or video deepfakes . . . . .	10
<b>4</b>	<b>Deepfake detection</b>	<b>13</b>
4.1	Image/Video detection methods . . . . .	14
4.2	Voice detection methods . . . . .	14
4.3	Analysis of existing tools for detecting deepfakes . . . . .	14
4.3.1	Deepware . . . . .	15
4.3.2	Sensity . . . . .	16
4.3.3	Other . . . . .	17
<b>5</b>	<b>Architecture and technologies analysis</b>	<b>19</b>
5.1	Requirements . . . . .	19
5.2	Containerazation . . . . .	20
5.3	Framework . . . . .	21
5.4	Web browser extension . . . . .	22
<b>6</b>	<b>Framework architecture</b>	<b>23</b>
6.1	Microservice architecture . . . . .	23
6.2	High-level architecture . . . . .	24
6.3	API Endpoint . . . . .	25
6.4	Database . . . . .	26
6.5	Request processing and processing unit . . . . .	27
6.6	Monitoring . . . . .	27
6.7	Containerization and scaling . . . . .	28
<b>7</b>	<b>Client architecture</b>	<b>29</b>
7.1	Web browser plugin . . . . .	29
<b>8</b>	<b>Framework implementation and deployment</b>	<b>31</b>

8.1	API endpoint . . . . .	32
8.2	Message broker . . . . .	33
8.3	Processing unit . . . . .	33
8.4	Monitoring . . . . .	34
8.5	Scalability . . . . .	35
8.6	Deployment . . . . .	36
<b>9</b>	<b>Detection methods integration</b>	<b>38</b>
9.1	AudioDeepFakeDetection . . . . .	39
9.2	fakeVideoForensics . . . . .	41
<b>10</b>	<b>Client application implementation</b>	<b>42</b>
10.1	Functionalities . . . . .	43
10.2	Communication with framework . . . . .	43
10.3	Supported browsers . . . . .	43
<b>11</b>	<b>Test experiment and results</b>	<b>44</b>
11.1	Datasets . . . . .	44
11.2	Test case 1 - accuracy . . . . .	45
11.3	Test case 2 - small bursts . . . . .	50
11.4	Test case 3 - congestion . . . . .	51
11.5	Tests evaluation . . . . .	52
11.6	Cost analysis . . . . .	53
<b>12</b>	<b>Conclusion</b>	<b>54</b>
<b>Bibliography</b>		<b>55</b>
<b>A</b>	<b>Contents of the attached media</b>	<b>60</b>

# Chapter 1

## Introduction

Deefake is the buzzword that has no agreed-upon technical definition. It consists of two words, deep and fake. Deep is referring to deep machine learning, which is used for creating fake voices, images, or even videos. It is a fast growing technical field of study and could be a major threat to society. We live in information era and creation of unrecognizable fake media affects us all. Deepfakes could be found on social media, news portals, etc.

This paper shows some security risks and impacts on many different fields of human life. There are already many popular videos containing deepfakes on social networks with millions of views, but also several successful attacks on biometrics systems. People are not able to consistently recognize deepfakes. That is why we need to create detection tools to help anyone with problem that is already there.

There are many different deepfake categories from face swap to morphing and more. Most of the detection methods focus on a single domain, which means that they are capable of detecting only one deepfake category. There are few tools in the market utilizing those detection methods targeting undemanding/inexperienced users, and some of them are not free to use.

The first two chapters cover general knowledge about deepfakes such as potential risks and security impacts with mention of successful attacks, different types of voice, image, and video deepfakes. You can read about experimental detection methods and available tools in the market.

Second and more important part describes process of development deepfake detection framework and client application, which is the goal of this work. It starts by specifying requirements followed by detailed architecture. Architecture was designed with focus to easily integrates new detection methods into framework.

Last but not least, the implemented framework containing several open source detection methods was deployed into Azure AKS cluster and properly tested. The tests were mostly focused on the testing framework itself, mostly measuring used resources. Integrated detection methods were also tested, and all the results could be found at the end of this paper. Whole project is available in Github repository<sup>1</sup>.

---

<sup>1</sup><https://github.com/PlayerBerny12/VUT-DIP-Code>

## Chapter 2

# Security impacts of deepfakes

The creation of fake media and their detection have been a problem since photography was invented. Digital photography or video with tools such as GIMP, Adobe Photoshop or Adobe After Effects allows more people to create fakes than before, still some experience in this area is needed. Media that have been modified or otherwise manipulated are called synthetic media, and they do not depend on whether it is an analogue or digital medium. Deepfakes also fall under this category [22]. Deep learning-powered tools allow even more unexperienced users to easily create trustworthy fakes.

The quality of deepfakes reached a level when a trained person or even an experienced researcher in this field has a problem of spotting them. This fast development allows creating realistically looking assets to art photography or movie production; unfortunately, it can be used for malicious purposes like creating fake porn videos to blackmail people or manipulate public via fake news. There are many use cases where deepfakes can be applied.

This is not a discussion about potential technology; deepfakes are already there and their usage will probably have an increasing trend. Over the last couple months, we could see huge development and popularity explosion around large language models such as ChatGPT [4]. Deepfakes do not have such publicity, but they have the same potential to change whole human society as previously mentioned language models. We are not far from using a pre-trained voice deepfake model powered by some large language model to automatically perform attack itself.

It is putting huge pressure on researchers to develop new forensics tools or any technology which will prevent malicious usage of deepfakes. As mentioned before, creating fakes is not new, and a whole field of study engaged in spotting fakes and developing techniques over 15 years. Despite continuous research efforts in the past, the advent of deep learning changed the rules of the game. [53]

### 2.1 Human capabilities of deepfake detection

The human ability to recognize fake media from the originals is in contradiction to their quality. Korshunov and Marcel [27] confirmed this in their research. They created a questionnaire containing several videos, and the subject (interviewee) had to answer after watching the video whether the person in the video was genuine, fake, or uncertain. The videos were manually divided into six categories (very easy, easy, moderate, difficult, very difficult, and original).

The videos were split into several categories manually by the researchers, probably without using any metrics but based on their experience and feelings. Afterwards, ANOVA test shows there is an overlap in several categories, so several videos could be moved to a different category. However, the categories are still significantly different.

The results of test in fig. 2.1 certainly demonstrate that people's recognition ability decreases significantly as the quality of deepfakes increases. Also, the audience of this test knows they are looking for fakes, otherwise we can expect worse results if there will be unsuspected audience (e.g., deepfakes on social media). It is quite alarming that the correct answers in the category of "easy" reach only 71,1 %. The quality of deepfake increases over time, thus it can be expected that human recognition ability will continue to decrease. [27]

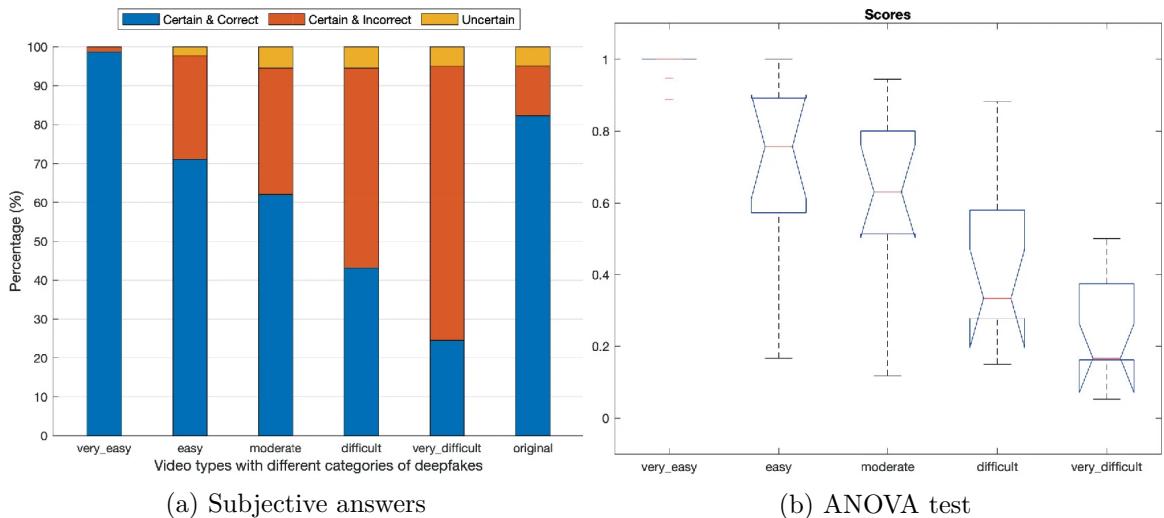


Figure 2.1: Subjective answers and median values with error bars from ANOVA test for different deepfake categories. Retrieved from [27].

Another research tested only recognition of audio tracks and they were comparing humans versus computer program. Attendees had a correct classification between fakes and origins 67 % after the first several rounds. Their accuracy increased while listening and answering to more tracks, but the value stabilizes at 80 %. On average, trained AI performs about 10 % better than human, but this result highly depends on the difference in learning and test dataset. Still, it shows that the computer can outperform humans in spotting deepfakes. [35]

## 2.2 Potentional risks and impacts

Humans are not good at recognizing deepfakes, but "why should we be worried?". Almost every technology humankind created could be used for good or bad – deepfake is no exception. There are plenty different deepfake categories, and each has its own attack vector or use case. This section is covering potential risks of those categories and their closer description will be covered in chapter 3.

Deepfakes are about gaining someone's trust or influencing him. For the last couple of years, there has been an increasing trend of scamming people, mostly via phone or computer [7]. Targeting only one person/victim, for example, to gain their money or information. Those attacks are getting better and more credible and using deepfake to

impersonate close friend of victim could be next step how to improve it, if it is not already happening.

Creating “fake news” to influence a large audience is the most common use case of deepfakes because we live in an information era. There are many targets of “fake news” such as rigging elections, demoralizing military units, or manipulating the stock market. In this case, politicians, celebrities, and significant personalities will be used in deepfakes to influence the audience. We can only imagine what one person or a high quality deepfake can change with enough media reach. For example, after one tweet from Elon Musk about Tesla’s stock, sends shares down more than 10 % almost immediately [47]. [22]

A real example of deepfake is the famous video with Barak Obama insulting Donald Trump, which should spread awareness regarding the fast developing category of new thread<sup>1</sup>. Several years later another video stating Volodimir Zelenskyj talking about surrendering, it was proved that it is a manipulated video, and its purpose was to demoralize Ukraine army and make them capitulate<sup>2</sup>.

There are other videos similar to the video of Barack Obama with the same purpose, spreading awareness about deepfake. For example, speech of Czech Republic president Miloš Zeman created by HBO for propagating their new series<sup>3</sup>, but also to inform general public about this technology. We can consider those videos and their goal to be mostly success. Most people have heard about deepfakes and they are connecting them with those types of videos. However, many people share deepfakes inadvertently on their social media [2].

Not everyone considered them as a big threat. This is probably because there is no proof that there was a successful deepfake attack or fake news that influence them directly. This is the reason why deepfakes could be so dangerous. The video of Volodimir Zelenskyj might have had a good chance to be successful, but it was quite fast debunked because of its terrible quality. People still should be informed about capabilities of deepfakes, but the target of the message should be different. Not saying only be careful we can make politicians say what we want, but show and provide some tools on how to spot the deepfakes.

Another field where deepfakes could be used is to trick biometrics systems to mark attacker as a different person. This technique is used to gain access to secured equipment, to building or even to application (internet banking), etc. Biometrics systems have been proven not ready to deal with deepfakes [14] [12]. The work describing one of attacks is called “Magic passport” and demonstrates how biometric systems could be vulnerable. Most systems will require some changes such as adding a new module to the authentication pipeline, which will be detecting deepfakes [23]. Face or voice biometrics recognition systems are in greatest danger. Falsification of documents is related to this topic and there was a case of smuggling people across borders with an official passport containing morphed photos of two individuals [45].

These cases are only the tip of the iceberg, and in the future, everyone should ask if video on social media with film celebrities is real or even worse, if the evidence in courts is trustworthy or not. The solution to this is to use tools capable of detecting deepfakes. Those tools must be created with caution for unskilled users.

---

<sup>1</sup><https://www.buzzfeed.com/craigsilverman/obama-jordan-peele-deepfake-video-debunk-buzzfeed>

<sup>2</sup><https://www.youtube.com/watch?v=X17yrEV5s14>

<sup>3</sup><https://www.youtube.com/watch?v=FzMnDwpKJrI>

## 2.3 Detection tools requirements

Now we know that deepfakes could be security threats and there is a need for reliable detection tools. There is not many detection tool available for users. Most of them are basically command line scripts which are not for general purpose. More about individual methods could be found in chapter 4.

The user interface should be simple, yet provide all relevant information. Generalizing anything to one number will be great, but not every time it is possible. Speaking of detection methods it should be able to deliver only one number. On the other hand, it will be nice to show the user where on the image is the manipulated area.

Nowadays people work with many different types of files. Only for audio we can have WAV, OGG, MP3 and many more. This means that the detection method supporting only one file type will not have huge success. There are two types of solution to this problem; first is to support as many file types as possible natively. The second approach will be to utilize existing tools to convert input files to one specific format.

In the end how do we compare different detection methods or even the same methods trained on different datasets. We would have to test each method individually and compare the results. There are some problems that need to be sorted out before we will have a reliable detection tools in the market.

# Chapter 3

## Types of deepfakes

There are plenty of methods on how to create deepfakes, and as its name suggests some of them are based on deep neural networks but not exclusively. This section describes most common types of neural networks used for creating voice, image, or video deepfakes. One of the most popular types for face deepfakes is Generative adversarial network (GAN), and it is used to create completely new faces or face manipulations.

Each method leaves traces in the medium that can then be detected. This is one way to recognize deepfakes so understanding process of creation is an advantage. Detection is described in more detail in chapter 4.

### 3.0.1 Neural networks

Neural networks are composed from neurons arranged in layers, and each layer is connected sequentially via synapses. Synapses are weighed, and the process of finding the proper value of all weights is called a learning neural network. To obtain results from the input of  $n$ -dimensional  $x$  process **forward-propagation** is used to propagate  $x$  through each layer. [34]

Input to layer is vector  $a$  of values calculated by previous layer or in case of first layer  $x$  itself. That means result of each layer is also vector calculated by activation function  $f(a * W + b)$ , where  $f$  is activation function (Sigmoid, ReLU, etc.),  $a$  is input vector,  $W$  is matrix of weights between layers  $i$  and  $i + 1$  and  $b$  is dimensional bias. Dimensional bias is a constant offset that helps the network shift the activation function toward the positive or negative side [1]. [34]

Now let's consider the neural network  $M$  as a black box and denote its execution as  $M(x) = y$ . Supervise learning to train  $M$  is using paired samples with from  $(x_i, y_i)$  and loss function  $L$  is defined. Loss function is to generate a signal at the output of  $M$  and propagate him back to find error of each weight in synapses. [34]

Optimization algorithms such as gradient descent are then used to calculate new weights of  $M$  for the number of epochs. As a result of this process, the network learns the function  $M(x_i) \approx y_i$  and is capable of making prediction on unseen data. More detailed descriptions of this could be found in the work of Y. Mirsky and W. Lee [34]. [34]

Next list shows types of neural networks used for generating deepfakes [34]:

- Generative Adversarial Networks (GAN) – Consist of two neural networks working against each other. One layer is generator and second is discriminator. Generator producing fake features trying to fool discriminator, on the other hand, discriminator is learning to distinguish between real sample and fake one.

- Encoder-Decoder networks (ED) – Contains at least two networks, encoder and decoder. It has narrowed layers towards its center. If encoder  $En$  and decoder  $De$  are symmetric and they are trained as  $De(En(x)) = x$ , then the network is called autoencoder. Generating deepfakes using ED trained with function  $De(En(x)) = x_g$ , where  $x_g$  is fake generated features. There is possibility to use multiple different ED chained after each other or using specific variant of ED called variational autoencoder.
- Convolutional Neural Network (CNN) – CNN is learning pattern hierarchies in the data. For deepfakes purposes, it learns filters applied over the input and forming an abstract feature map as the output.
- Recurrent Neural Networks (RNN) – RNN can handle variable length data and during processing it remembers state that can be used in next iteration. RNN are mostly used in audio.

Each architecture has its own subcategories that have small modifications or using some techniques from different architecture. All above mentioned neural networks types are shown in 3.1 and 3.2

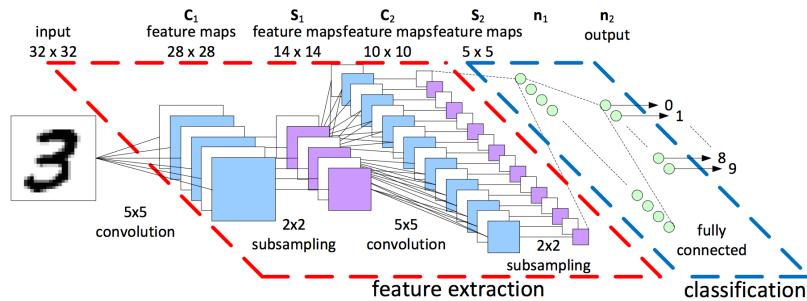


Figure 3.1: Architecture of convolutional neural network. Retrieved from [10].

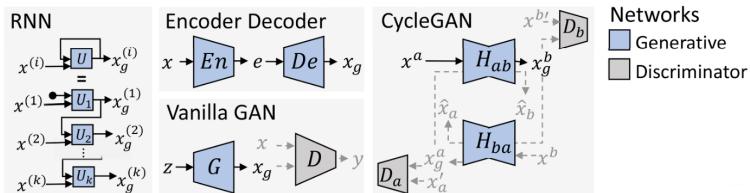


Figure 3.2: Basic neural network architectures (RNN, ED, GAN). Retrieved from [34].

### 3.0.2 Voice deepfakes

There are three different modalities for speech synthesis: text-to-speech (TTS), voice cloning (VC), and replay attack (RA) [57]. The last one is based on capturing victim voice and the replay of it. It is quite easy and cheap to perform because it only requires capture and replay device (today we can use for example smartphone) and current methods for voice recognition still have accuracy issues [33]. First two are using AI-synthesis with content regeneration which makes them more indistinguishable for naked ears. [57]

Text to speech (TTS) converts written text to artificial speech, on the other hand voice cloning consumes source voice. Both methods produce synthesis voice saying desired phrases

specified by the input, high-level diagram how those methods works could be seen on fig. 3.3. Voice deepfakes are used independently or with deepfake video (e.g., full puppet). Creating synthesis voice is computationally challenging and one of the goals is making real-time voice cloning. There are several projects that are trying to accomplish this<sup>1</sup> <sup>2</sup>.

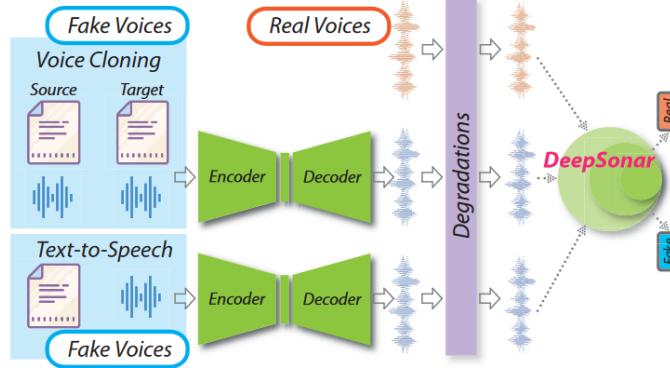


Figure 3.3: Text-to-speech and voice cloning data flow diagram. Retrieved from [57].

### 3.0.3 Image or video deepfakes

The list of the following deepfakes is based on the work R. Tolosana, et al. [24]:

- Identity swap – Replacing the face of subject with the face of target as shown in fig. 3.4. There are two different approaches, classical computer graphics-based technique and deep learning technique. Generally, the process of swap could be described as face detection, cropping, extraction of intermediate representations, synthesis of new face, and blending the generated face.



Figure 3.4: Examples of real and fake identity swap images. Retrieved from [24].

<sup>1</sup><https://github.com/SolomidHero/real-time-voice-conversion>

<sup>2</sup><https://www.resemble.ai/speech-to-speech/>

- Full puppet – Method related to identity swap allows creation of so-called puppet. One person (master) is source of facial expression and body movements that are mapped onto target person as shown in fig. 3.5. [22]



Figure 3.5: Full puppet technique visualisation. Retrieved from [27].

- Morphing – It is a type of manipulation that is used to create artificial biometric face samples. Final face contains resemble biometric information of two or more individuals. It should be possible to be successfully verified by biometrics systems for all individuals who were source for given deepfake. Fig. 3.6 shows an example of a morphed image.

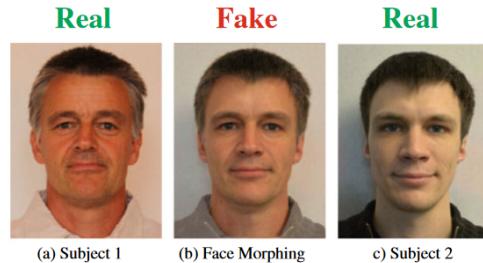


Figure 3.6: Examples of fake morphed identity from Subject 1 and Subject 2. Retrieved from [24].

- Attribute manipulation – Face editing or face retouching technique involves modifying some attributes such as length or color of hair, color of skin, sex, age, adding glasses or other artefacts, and more. Fig. 3.7 shows an example of this technique.



Figure 3.7: Examples of real and fake attribute manipulation category. Retrieved from [24].

- Expression swap – Modifying facial expression of the subject as shown in fig. 3.8. This technique could be used as part of full puppet.



Figure 3.8: Examples of real and fake expression swap category. Retrieved from [24].

- Audio/text to video – This method relates to expression swap synthesising facial expression from audio or text. It is also known as lip-sync deepfakes. Diagram in fig. 3.9 shows how this method works.

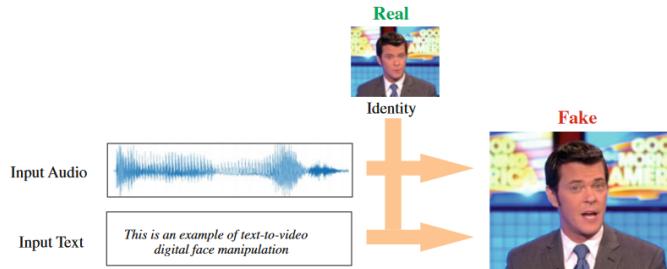


Figure 3.9: Examples of real and fake audio/text to video fake category. Retrieved from [24].

Creating deepfakes nowadays is complex task and many deepfakes is using more techniques so that they could be included into more than one category. Attackers can possibly create identity swap deepfake and after that apply attribute manipulation to this deepfake to tune final results.

## Chapter 4

# Deepfake detection

As we stated, humans are not good at recognizing deepfakes. Creating deepfakes could leave visible defects (e.g., blurring or misalignment on edges in the image). A. Firc [13] summarizes the list of features to focus on when spotting fakes for humans.

- Facial features – eyes and their movement, eyebrows, glasses, facial expression, hair and facial hair, skin, lips, teeth
- Body features – body position and posture, body movements
- Voice features – unusual tempo, end of words, fricatives, conversation
- General indices – blurring or misalignment on edges, inconsistency noise or audio

This list points to critical parts where defects, which were created during the creation of deepfake, could be spotted. Generation of deepfakes is getting better, plus other masking techniques are being used. The big problem for detection is lossless compression, several chained resizes of an image, application of noise, etc. Basically, all methods that lead to some degree of data loss, but without noticeable change for the naked eye or ear. When generation of deepfakes is not good enough manual post-processing could be used for polishing results (images – Photoshop, GIMP, etc.).

Machine detection could be divided into two categories: standard algorithms looking for physical inconsistency and digital integrity, using the same features as Firc described. Other methods are based on machine learning. The same “masking” techniques listed before (compression, chained resizes, etc.) have the same effect on machine-based detection. Any data loss prevents the usage of reliable methods such as frequency analysis. Still, computers perform better than humans because they can also use different features (e.g., pixel-level features), especially neural networks trained for deepfake detection. [53]

The problem of neural networks is that we do not know what it is learnt to detect. It depends highly on training dataset. Probably the worst case scenario would be only recognizing suspicious images containing traces after “masking” techniques such as double compression, noise patterns, etc.

Another problem of detection algorithms is bad generalization. Most of methods are trained on single domain deepfake (e.g., identity swap), which means they are not able to recognize deepfake from different category. When methods are trained on multi-domain datasets, accuracy is going down. [29]

## 4.1 Image/Video detection methods

As stated, before most of methods for deepfake detection are targeting only single domain. This section will describe examples of proposed detection methods for image/video deepfakes. There are conventional approaches and also more “exotic”.

P. Majumdar, et al. referring to multiple detection methods for image retouching (makeup, filters) and alternation (fully synthetize faces, morphing) [29]. Most of them use the same pattern which could be described as specific feature extraction followed by support vector machines (SVM) for classification. One of the methods proposed detection of images using face patches as input in the deep Boltzmann machine for feature extraction and SVM for binary classification. Another method uses softmax probabilities as features for the SVM. Other methods, for example, using convolutional neural networks. [29]

L. Spreewers, et al. made research on using local binary pattern with SVM for morphing detection [48]. A single LBP histogram contains 59 feature values, which means that for a  $3 \times 3$  layout, the feature space has 531 dimensions. The SVM classifiers are trained on between 650 and 1,000 samples. They also stated that EER increases to above 20 % while adding Gaussian noise to the deepfakes images.

Non-conventional detection method is heart rate estimation (remote photoplethysmography) by J. Fierret [21]. They are trying to estimate heart rate from video by evaluating frame-by-frame. There are other human physiological processes that could be used instead heart rate such as blood oxygen or breath rate. The score oscillates during the video and final decision is based on the mean/median/QCD score.

There are many other methods, and each will have its pros and cons, but as we can observe, SVM classification with large range of different feature extractors. Another rising group of detectors is using CNNs. There are not many researches using CNN as SVM, but results seems to be promising as we can see in researches [44] [36].

## 4.2 Voice detection methods

Voice detection complicates different languages; it is similar story to image/video deepfakes. There are face swaps, morphing, etc., and for voice there are different languages and dialects. Voice detection methods also copying trend from image/video detection methods. Support vector machines (SVM) with different feature extractors or CNNs.

Z. Almutairi and H. Elgibreen refer to multiple methods [5]. One of them uses the SVM model with Random Forest to predict synthetic voices based on a feature called Short-Term Long-Term. In this research, they compared SVM with many other classifiers such as Linear Discriminant, Quadratic Discriminant, Linear SVM, weighted K-Nearest Neighbors, and SVM outperforms all of them. Other referred work uses combination of two CNN, 1-D CNN and Siamese CNN. The Siamese CNN contained two identical CNNs that were the same as the 1-D CNN, but concatenated them using a fully connected layer with a softmax output layer. [5]

## 4.3 Analysis of existing tools for detecting deepfakes

Tools for deepfake detection are slowly getting from command line tools for experts to online tools with user-friendly interface. There are not many tools of this kind and some of them are not free to use. The following lines describe two available tools in a market.

### 4.3.1 Deepware

The Bosnia and Hercegovina recognize danger of deepfakes, while their parent company researched methods to develop an AI-based antivirus engine. Deepware company was founded to develop scanner for deepfake recognition.

Deepware provides REST API with web UI<sup>1</sup>, and mobile android application. The backend of this project with pre-trained models is accessible on their GitHub as Python command line tool<sup>2</sup>.

Scan & Detect Deepfake Videos  
Place a video link or upload a video

https://www.example.com/

By submitting data, you are agreeing to [Terms of Services](#) and [Privacy Policy](#)

BETA

Figure 4.1: Deepware scanner input form

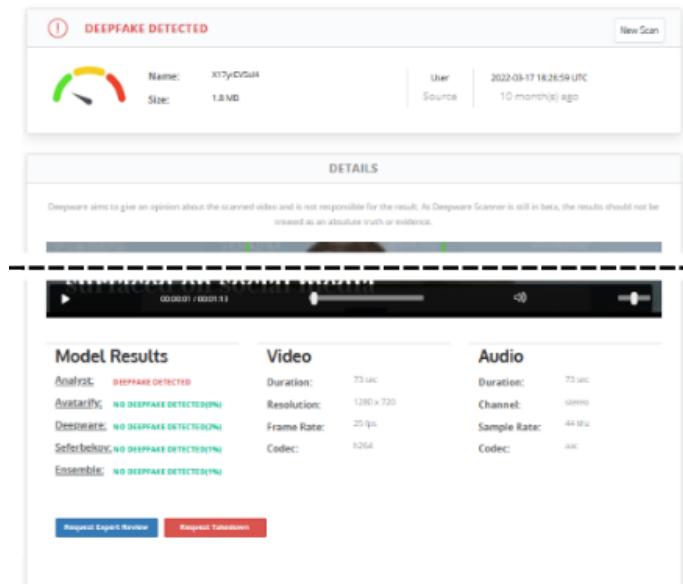


Figure 4.2: Results of Deepware scanner containing probability of deepfakes from multiple detection methods

The web application is stating that the project is in Beta, but unfortunately the last commit to the GitHub repository was made on 7th of June 2022 reported at the time of writing this thesis.

The tool provides a simple user interface to scan only videos. The user can input the link to the video (e.g., YouTube) or upload the video file directly as shown in fig. 4.1 . There are many supported video formats. The only limitation is the length of the video, which has to be shorter than 10 minutes.

<sup>1</sup><https://scanner.deepware.ai/>

<sup>2</sup><https://github.com/deepware/deepfake-scanner>

Processing of approximately 1 minute long video takes several seconds (3-10 seconds). Results contains video and audio metadata, results of multiple detection methods/models, and decides whether it is a deepfake or not with gauge chart of confidence as we can observe in fig 4.2. To use their REST API, you need to request authentication token. API provides the same functionality as web UI via three methods.

- POST /video/scan
- GET /url/scan
- GET /video/report

The first two methods execute scan on a video file or link and return report ID. Results report could be retrieved by last method and results are returned as JSON. Documentation also provides code samples for multiple programming languages on how to use API properly. The provided API can be integrated to other processes like mail communication scans or file upload filters.

	Correct (deepfake/original)	Incorrect (deepfake/original)	Scan failed
Celeb DF	10/7	0/0	3
FaceForensics++	4/5	1/0	10

Table 4.1: Deepware manual testing results

Deepware did not provide requested API key, so in table 4.1 you can find results of manual testing. The test contains 40 videos from two different deepfakes datasets (20 from CelebDF and 20 from FaceForensics++). The distribution between original and deepfakes videos was 1:1. The accuracy of detection is decent, but from 40 tested files 13 files did not go through scan, even repeatedly. It is a very small sample, so we leave it to the reader to draw conclusions.

### 4.3.2 Sensity

Sensity is very similar from the user perspective to Deepware. Based on the post on Sensity blog [46] from 2021 we can explore web UI of their application. The application is not publicly accessible and to obtain access, you need to request it. Sensity provides more tools related to cybersecurity, person identification, and verification.

Sensity allows for detection of images and videos by inserting files or referencing them via a URL link as shown in fig. 4.3. Sensity allows processing of quite small group of file types (png, jpeg, jifif, tiff, mp4, mov). Another limitations are for videos regarding their size (up to 30MB), length (up to 10 minutes), and quality of videos (up to 1440p). [46]

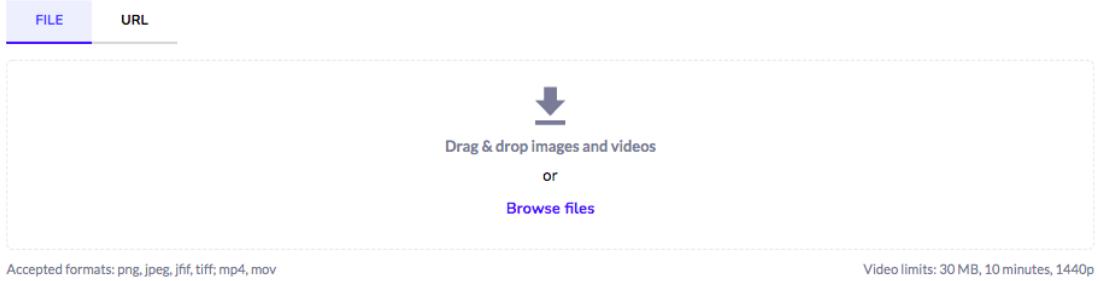


Figure 4.3: Sensity deepfake detection tool input form. Retrieved from [46].

The tool is capable of recognizing only face swap and fully synthesized faces by GAN. It is not able to recognize morphed images or other deepfakes. For GAN-generated faces, it is sometimes able to classify model generator. If deepfake is recognized, tool shows how confident he is, all shown in fig. 4.4. Compared to Deepware, it provides image scans; on the other hand, it does not have mobile application.

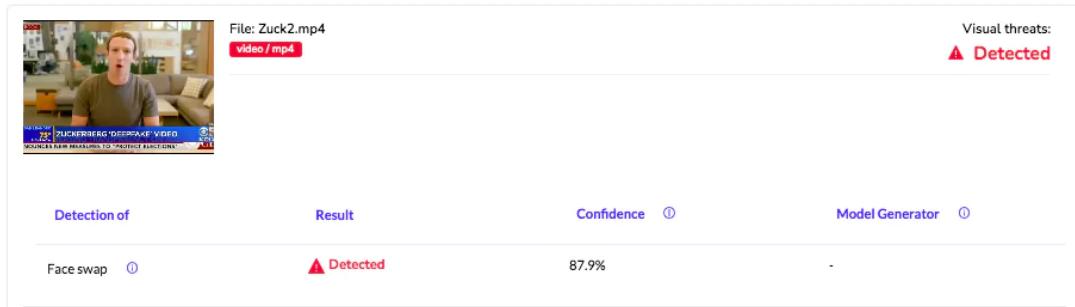


Figure 4.4: Sensity deepfake detection tool results. Retrieved from [46].

### 4.3.3 Other

Below you can find a list of other open source detection methods. These methods usually require a good knowledge of information technologies and programming to make them work on your computer. There is no guarantee that these methods really work, but two projects were selected for integration into the framework arising in this work.

- Video
  - dfdc\_deepfake\_challenge  
([https://github.com/selimsef/dfdc\\_deepfake\\_challenge](https://github.com/selimsef/dfdc_deepfake_challenge))
  - DeepFake Detection  
(<https://github.com/dessa-oss/DeepFake-Detection>)
  - fakeVideoForensics  
(<https://github.com/bbvanexttechnologies/fakeVideoForensics>)
  - Deepstar  
(<https://www.zerofox.com/deepstar-open-source-toolkit/>)

- Audio
  - AudioDeepFakeDetection  
(<https://github.com/MarkHershey/AudioDeepFakeDetection>)
  - specrnet  
(<https://github.com/piotrkwak/specrnet>)

# Chapter 5

## Architecture and technologies analysis

The goal of this work is to develop a deepfake detection framework capable of serving various client applications and an exemplary web browser extension that communicates with framework and clearly displays the results to the user. We already analyze the market and test existing tools. Because there is no external contracting party defining their need, we will define requirements by ourselves. Requirements will be divided into two categories, functional and non-functional requirements. It is important to define not only functional logic, but also some limits on processing time, number of users, etc.

### 5.1 Requirements

As we already said, the framework has to support multiple different types of clients (e.g., mobile, web), so there needs to be a properly defined communication interface between the client applications and the framework itself, which will act as server in this case. Framework allows dynamic changing detection methods, which means it is able to add, remove, or edit detection method. This implicates multiple detection methods at one time for voice, image, and video. Editing detection method means updating the model or parameters.

The platform should support as many file types as possible and adding a new method should not affect methods already presented in the framework, within supported file types. Different detection methods have different models depending on their design, so model management will be wrapped completely into the detection method.

The framework should collect statistics and hardware metrics, but the collection of personal information should be omitted. All collected metrics will be used for future improvements of detection methods and optimizing operating costs and performance. We would like to have a short response time with as low operating costs as possible. Those two parameters are in contradiction, so there has to be some balance between them.

Different cloud services offer their own tools for collecting metrics, but our system should be operating platform independent. That means framework needs to be consist of application, which collects metrics and provides them to other parts of framework, or person who operates the framework.

From a number of users perspective, the platform will handle up to hundreds of users at one time, and with enough resources the framework should handle even more. Scalability of computationally intensive parts of the framework will help meet these requirements.

The client application has to be simple so that anyone can use it. Users should be able to upload files or put a link to a file. The browser extension allows access to DOM of the webpage so an optional extension for client application will be selection of web element for detection. Application then retrieve metadata from selected element automatically.

Working with provided links or elements on specific pages brings several complications such as authentication. The framework does not have user context, so he is not able to retrieve all data that user has access to. We will not deal with these issues and if the file cannot be accessed, the processing request will not be created. It will be one of possible future improvements to the framework.

Because it should cover a large group of possible users, results have to be easy to understand and also provides information for experienced users. Last but not least, the client application should enable users to send feedback.

There are several web browsers in market that cover almost all audience, so the developed browser extension should be portable among multiple web browsers. Today it is standard, but it is necessary to mention that the application should be secured. It will affect code of framework, client application, and environment itself (hardware, operating systems, ...). All work is open-source and accessible on the public GitHub repository<sup>1</sup>.

Summary of functional requirements:

- Adding, removing, and editing (changing parameters and models) detection methods
- Multiple detection methods at once
- Collecting statistics and feedback
- Detection of image, video, and voice
- Detection of file, URL link, or selected HTML element (optional)
- Understandable presentation of results
- Security

Summary of non-functional requirements:

- Scalability
- Small response time
- Low operating costs
- Portability of the browser extension among multiple web browsers (optional)
- Open source

## 5.2 Containerization

The framework requires dynamic management of the detection methods. It means that the framework can contain one or twenty different detection methods. Each method could be developed using different programming languages or technologies. Model management will

---

<sup>1</sup><https://github.com/PlayerBerny12/VUT-DIP-Code>

be encapsulated and handled by each detection method individually. This leads to some isolation of methods with a well defined communication protocol.

Scalability of framework, isolation and resource management, all this together will be reached via containerization. There are many technologies dealing with containers, such as Docker, Podman, LXC. When we start counting orchestration with automatic scalability, the number of technologies drops down.

Azure and AWS offer their own container orchestrators: Azure Container Instances [30], Amazon Elastic Container Service [6]. As already stated, the framework should be operating platform independent, so looking only for orchestrator supported by all cloud providers, we discover Kubernetes. Google Cloud, AWS and Azure support Kuberentes, and no other alternative supported by all of them exist. [30] [6] [17]

Docker is probably the most widely used technology for containerization. Podman is another containerization technology having similar API to Docker. Podman is newer and has different architecture than Docker which results in a more secure operation and fewer resources required for running [3]. But still, we will stick with duo Kubernetes and Docker because of the huge community and good support of Docker in Kubernetes and also less concerns about possible issues during development.

### 5.3 Framework

The framework is a collection of detection methods, an interface for the client application (receiving requests, sending report response), and orchestrates/arranges all communication. It will be built into several containers and one of the containers will be providing client application interface. It could be REST API [42], GraphQL [49] or custom protocol. It is not a good idea to develop a custom communication protocol, and because of simplicity of interface we will use the most used one, REST API. For this purpose, we can use a huge number of different technologies such as ASP.NET Core, Flusk, Django, Ruby on Rails, etc. Basically all named technologies meet defined requirements (response time, number of users, ect.) because most of them are covered by microservice design and they are able to run in containers. Our choice is ASP.NET Core because it has a huge community, Linux support, many external libraries, open-source development, it is suitable for bigger projects, and it has good speed of program development [31].

Communication among containers will be through message broker. There are several alternatives such as Memphis, RabbitMQ, or Apache Kafka. All of them are open source and possibly suitable for our use case. The final choice is RabbitMQ because of small resource consumption, a quite large community, and this project is developed since 2007. That means we can count on future development and updates. [20]

The framework also needs an application for collecting metrics. Here, we chose Prometheus [39] and Grafana [19]. Both tools are free to use and open source. Prometheus is collecting metrics, but it is not a good visualization tool; for this purpose, there will be Grafana. Grafana provides dashboards where all metrics could be presented as graphs, etc. Reasons why we take Prometheus and Grafana is because they have good cooperation and RabbitMQ has metrics compatible with Prometheus [55]. Capturing resources is performed by Node Exporter. It is another open source project from Prometheus, which ensures good compatibility with the previously mentioned systems [38].

All previously selected technologies also have good support of Kubernetes and in some cases provide custom Kuberentes operators for easier deployment. More details about it are provided in chapter 8.

## 5.4 Web browser extension

Web extension are based on web technologies such as HTML, CSS, JavaScript, TypeScript, etc. Because of portability and good integration, we will choose HTML, CSS, and TypeScript. TypeScript is a strict syntactical superset of JavaScript and adds optional static typing to the language. It is also compiled into JavaScript [32].

There are several TypeScript frameworks that speed up development of application and make portability among different browsers easier. Angular seems to be as good viable option and it is frontend framework, with which we have most experience [16].

Browser extensions are integrated through a standard called “Web app manifest”. It is the definition of manifest file used for integration extensions to browser, now in version 3. Standard defines the structure of the manifest that allows the configuration of different types of capabilities. All browsers have partial or full support. Using this manifest should help to integrate the extension with browsers such as Google Chrome, Microsoft Edge, or Mozilla Firefox. [37]

# Chapter 6

## Framework architecture

The architecture must reflect that each detection method could be developed by different technology stack. We can isolate the detection methods from each other and wrap them into independent services. We do not need communication among all methods because they do not cooperate or share data. In this case, the microservice architecture meets all defined requirements.

### 6.1 Microservice architecture

Microservice architecture is the style of developing applications. Example of design can be observe in fig. 6.1. The microservice allows the application to be separated into smaller logical independent parts. The service then has its own realm of responsibility and they can communicate with each other. The use of containers for the deployment of microservice application is a well-suited option. [18]

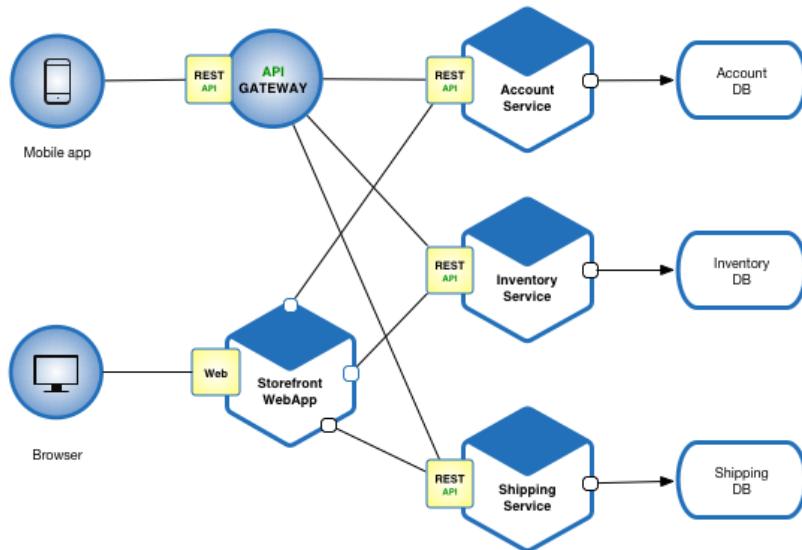


Figure 6.1: Microservice architecture of fictitious e-commerce application

Microservices improve fault isolation that non-functional service should not affect others, in best case scenario. For example, if one service contains a memory leak, it is not propagated into other ones. In real cases microservices depend on each other and this might

lead to a problem. Another benefit was already mentioned, and it eliminates commitment to one technology stack. One service has better maintainability because it should be small (better understandability, faster tools, etc.), which leads to more productive development.

This architecture also has several drawbacks. It increases the complexity of the architectural design and the additional implementation of cross-service communication. Here comes the usage of message brokers. When saving data to the database, there are several approaches on how this could be implemented. You do not want your database to be bottleneck of your system, so developers in most cases have to deal with distributed database systems. [43]

## 6.2 High-level architecture

The client application communicates directly with the REST API endpoint. It is responsible for handling requests, preparing and validating data, and selecting which type of processing should be used. After processing is done, it collects all results, stores them in database, and distributes them back to the client application. Fig. 6.2 contains high-level design of a framework with data flow. Numbers represent steps in which order data flow through the framework.

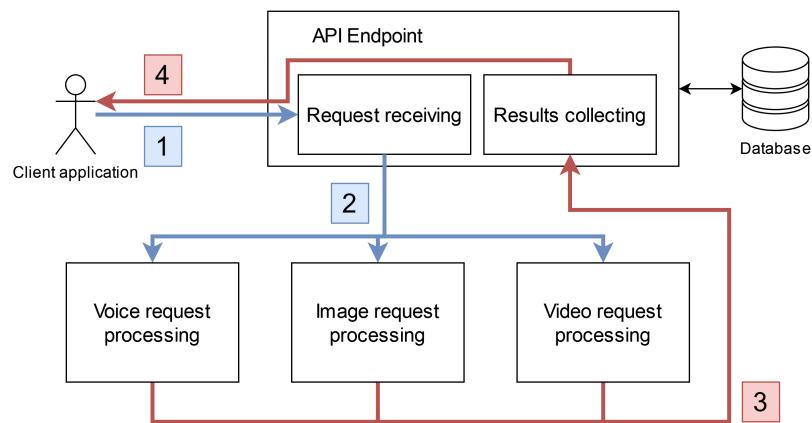


Figure 6.2: High-level design of whole framework

The architecture separates voice, image, and video detection into independent units. Each unit contains a processing queue where the request will be assigned by the API endpoint. The queue is serviced by one or multiple processing units that contain all related detection methods as shown in fig. 6.3.

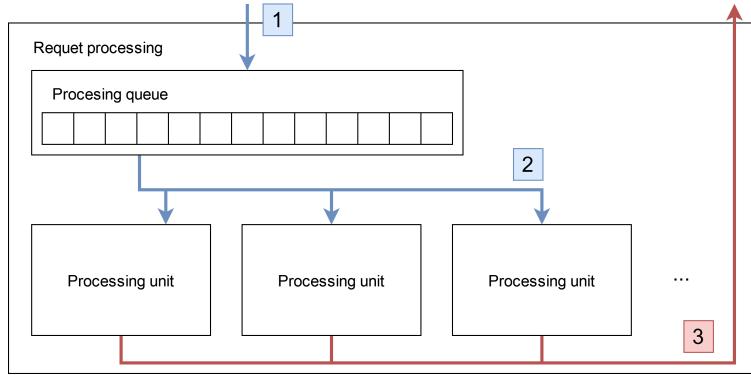


Figure 6.3: Request processing detail

The processing unit works as a parallel pipeline. Closer look at design is in fig. 6.4. Some detection methods require input data in specific format, others not, so the first step is optional data preparation. The next step is the detection method that decides whether the input data contains deepfake. Detection methods are different, so are their results. As last step it is need to properly generalize and also normalize resulting values to interval from 0 (deepfake) to 1 (original).

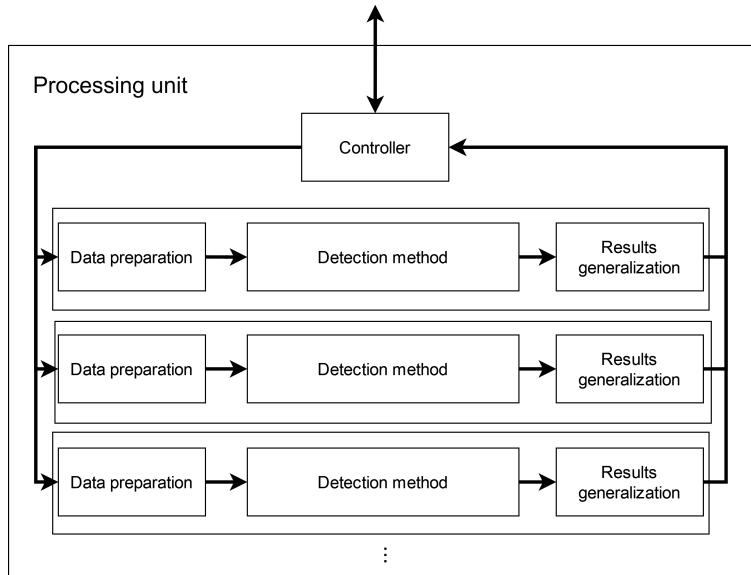


Figure 6.4: Processing unit pipeline

### 6.3 API Endpoint

As mentioned before, API endpoints communicate directly with client application and with backend processing mechanism. Processing takes some time, so client application creates request and wait asynchronously to be processed by the framework. The API for this purpose contains **detect** and **request** methods.

**Detect** methods calculates checksum of file and lookup in database if there is request with same file checksum. If any request is found, then new request is not created and found

request is returned. In case there is no request with same checksum new request is created and stored in database. Then this request is move into next step of processing.

There are other support methods for health check or providing feedback. All methods are described in following list:

- /ping
  - GET /ping – healtcheck endpoint
- /detect
  - POST /detect/file – creates request with file in HTTP request body
  - POST /detect/link – creates request with link to file in HTTP request query parameters
- /request
  - GET /request/results/ – returns results of processed request or empty results if request is in process
- /feedback
  - POST /feedback/<request\_id> - collects user feedback with optional parameter request\_id

Each detection method has different processing times. There are two options to return results to user: partial results from the methods already completed or returns only when everything is completed. Because there is no need to show partial results, the second option will be used. The framework also calculates the overall score and this requires all results from all methods. However should not be much code changes if in future there will be need of partial results. Calculating the overall score requires considering method differences as well as their domain focus. A single detection method is usually developed to detect only single domain (e.g., face swap). We leave the exact definition of calculation to the implementation.

The method **request/results** could be possibly replaced by websockets which will increase complexity, but on the other hand a client application could report status of the running process more precisely. It is another possible extensions to the framework.

## 6.4 Database

API Endpoint needs to store some data in a database to efficiently manage multiple requests. It is also not welcome to process the same file repeatedly. By keeping the results in the database, it will save computational resources and the application will have minimal processing time for already processed files. It is also necessary to store the sent feedbacks. In fig. 6.5, you can see the ER model of the database for this framework.

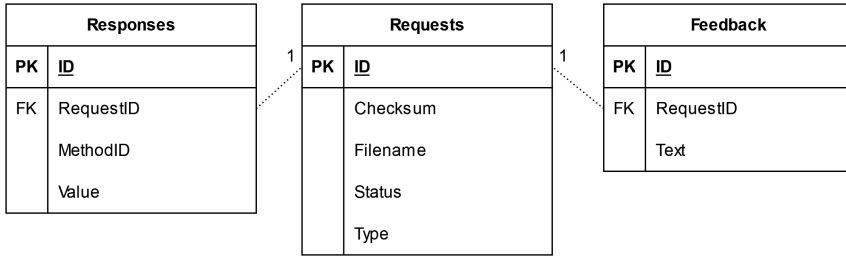


Figure 6.5: Entity-relationship model

When any method is added or anyhow changed (e.g., different model) related or all results in database should be deleted. If we do not delete stored results users will be getting old results, that is not what we want.

## 6.5 Request processing and processing unit

API Endpoint placing all new requests in the request queue. One of the available processing units takes the first request from queue. This request contains all the information such request ID, data for detection. Message queue in RabbitMQ will be used for this purpose.

Free Controller takes the message from the queue and distributes it to all detection methods under his administration. Because each detection method is encapsulated into its own container, it needs communication channel with Controller. To make integration of new detection method easy as possible we use REST API once again. The detection method should expose one method called `detect` on any port that will be placed in Controller configuration. Controller can call this method to pass parameters and start processing.

The first step in the processing pipeline is optional data preparation. The detection method could be trained only on a specific resolution of the video/image or on a specific length of the voice sample. This means that a data processing unit must be designed for a given detection method. Prepared data then are picked up by detection method and evaluated.

After detection, there is optional data generalization/normalization. Because the results must be correctly presented to user and the framework must calculate overall score, all results need to be normalized to same interval from 0 to 1. Where zero means detected file contains deepfake and one means opposite. The results are then pushed to the output queue.

## 6.6 Monitoring

One of the requirements was to collect metrics on resource consumption and also metrics on which to scale the number of processing units. Node exporter was selected as the tool to collect resource consumption metrics to be run on every node in the cluster. RabbitMQ handles the collection of metrics itself, no additional configuration is required.

Prometheus collects metrics from both of these sources and stores them in its database. Node exporter collects metrics about CPU utilization, RAM consumption, disk utilization, network card utilization, etc. RabbitMQ collects metrics about the number of messages in each queue, number of connections and channels, etc. These metrics will be displayed by Grafana in their dashboards and Prometheus will be used as a source.

## 6.7 Containerization and scaling

Detection methods have to be containerized, and supported methods relevant to detection are encapsulated together with detection method. Processing unit is consists of controller and multiple detection methods and all togeather it is creating multicontainer pod.

Scaling can be done vertically or horizontally. Vertical scaling changes the number of resources, such as CPU cores or the size of RAM. Horizontal scaling changes the number of nodes/units/containers that process a given task. Fig. 6.6 shows the difference. Kubernetes allows automatic scaling based on the consumed resources. We use horizontal automatic scaling based on consumed resources for the number of nodes in the cluster. Second scaling uses the number of messages in the queue, and if the number of requests in the queue changes up or down, then horizontal scaling adds or reduces the number of processing units for given queue. [8]

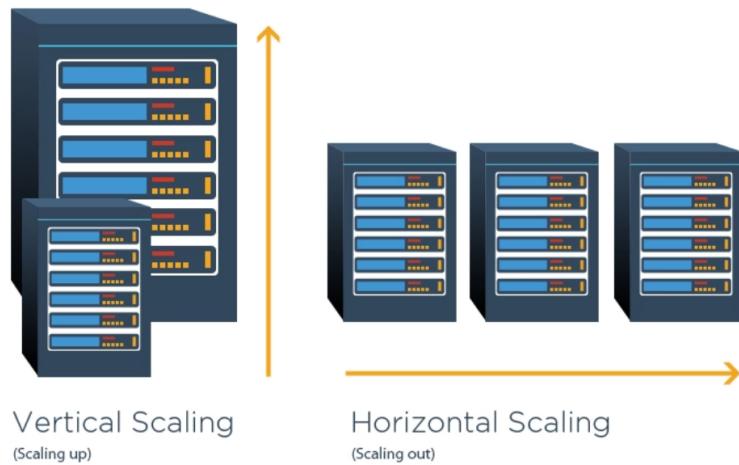


Figure 6.6: Vertical and horizontal scaling comparison. Retrieved from [8].

The number of messages in queue can be found in Prometheus, but we need to make it available to the Kubernetes API. To do this we will use the Prometheus adapter; this is a project that maps defined metrics from Prometheus to messages understood by the Kubernetes API [28]. Then we just need to configure the Horizontal Pod Autoscaler to turn scaling on with usage of these values. Horizontal Pod Autoscaler will scale up when there are a lot of requests waiting to be processed in the queue. On the other hand, to save costs, it will automatically scale down when the peek of requests drops down.

# Chapter 7

# Client architecture

The use case of the client application is straightforward, so there is no need for use case or data flow diagrams. The application will try to prepare the file for inspection and send it to the framework. The next section contains several wireframes of how the client application should look like with the description.

## 7.1 Web browser plugin

It should allow the user to insert a file via file upload, link, or HTML element containing a targeted file. In fig. 7.1 we can see selection of insertion types, for file upload the user will be prompted by system file upload picker. When user pick link as input, floating window will pop up, and user then can insert URL to file. Those two selections are the same as other tools in market also provide. Optional improvement will be element selection which switches the application to interactive mode where user can point to HTML element and application will try to retrieve metadata of audio, image, or video directly from HTML.

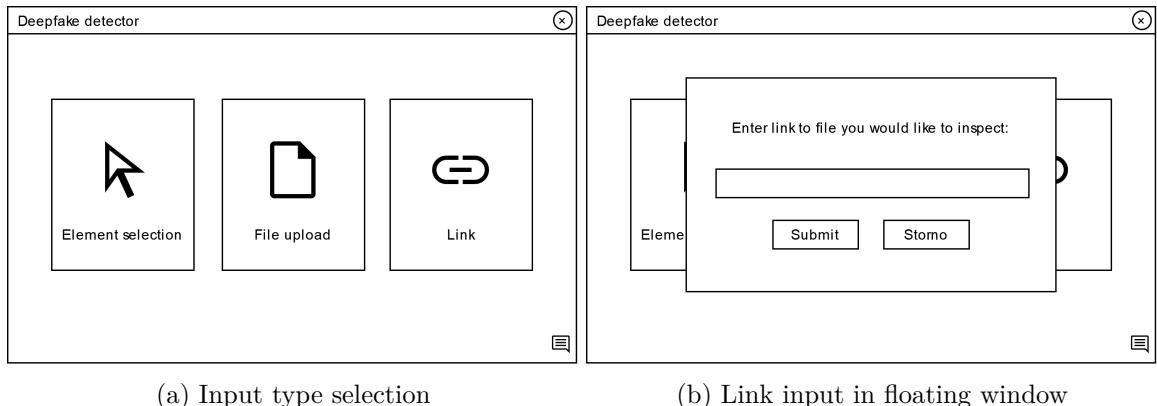


Figure 7.1: Input type selection screens

Because detection framework will contain several detections method we need to show the result of each method independently. It could be a little confusing for user so there will also be overall score which interprets/generalizes all the results. The overall score will indicate the results as a percentage and as emoticon. The palette will contain five emoticons shown in fig 7.3. For better understanding there are question marks in the UI that after mouse hover shows tooltip with description. Whole results screen can be view in fig. 7.2.



(a) Results view

(b) Tooltip with description of detection method

Figure 7.2: Results view screens

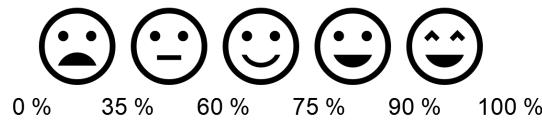


Figure 7.3: Palette of emoticons indicating if inspected file contains deepfake or not

At the bottom of each screen there is an icon that enables the user to send feedback from application. After clicking on this icon floating window will appear, shown in fig 7.4. When user will be sending feedback on the result page, request ID will be added as an additional parameter to the feedback message.

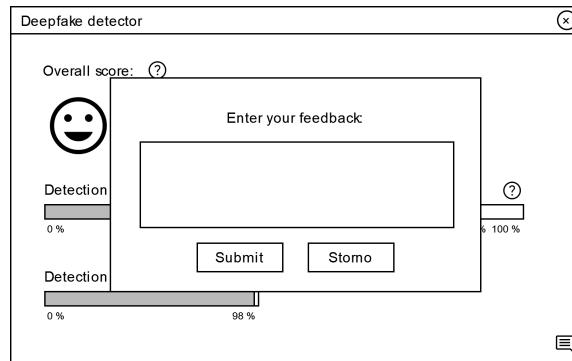


Figure 7.4: Feedback form in floating window

# Chapter 8

## Framework implementation and deployment

The implementation is available in the public repository<sup>1</sup>. The project is divided into three folders: `clients`, `server`, and `templates`. The `clients` folder is intended for various implementations of client applications such as web client, web browser extension, etc. Currently, only one client is implemented, but other types can be added in the future. The `server` folder contains the framework implementation and the Kubernetes deployment manifest files. The last folder contains templates for easy integration of detection methods.

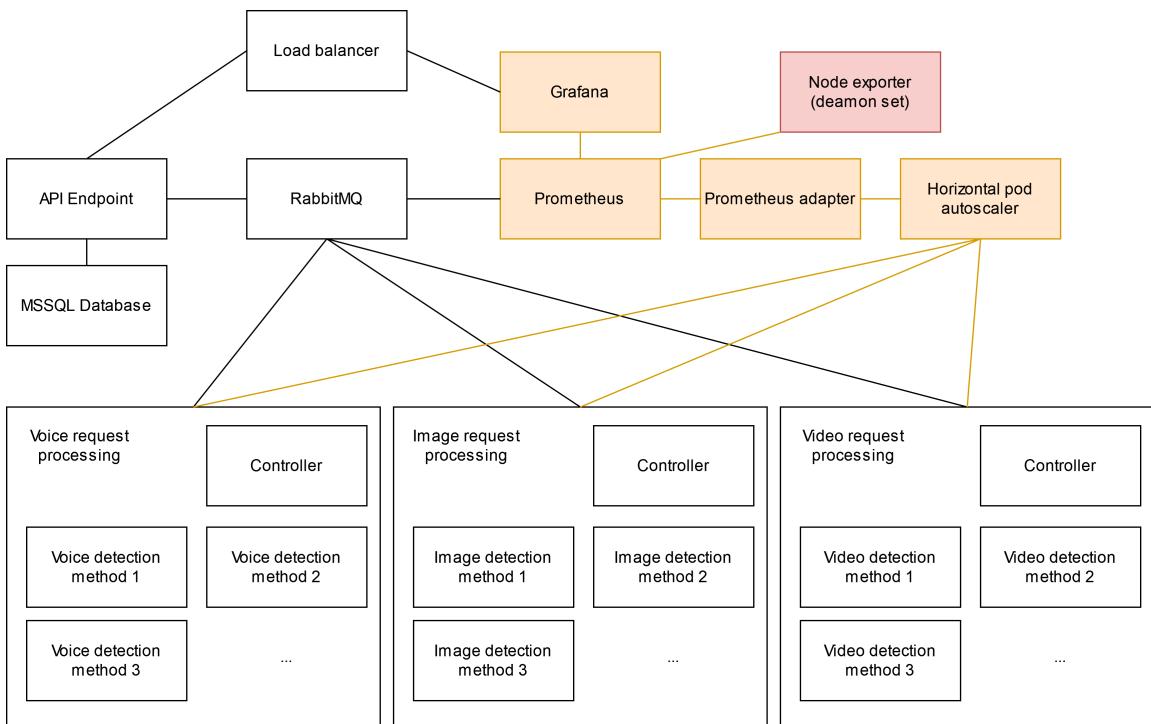


Figure 8.1: Diagram of implemented framework

The framework was implemented on the previously defined architecture. The architecture primarily dealt with the main functionality of the framework, but it also covered

<sup>1</sup><https://github.com/PlayerBerny12/VUT-DIP-Code>

supporting systems, for example, systems for collecting metrics used for scaling, etc. The overall implemented framework can be seen in diagram in fig. 8.1. The system could be divided into two parts: processing - API Endpoint, message broker, processing units (white rectangels), and monitoring - metric collector, observability platform, etc. (orange/red rectangels).

The individual parts are described in more detail in the following sections. The whole system is designed to work in Kuberentes, the container orchestration tool. The overall implementation involved the development of an API Endpoint and Request Processing Controller. Subsequently, it was necessary to configure other systems, such as message broker, database, or metrics collector. Finally, it was necessary to integrate the different selected detection methods; for this purpose a templates were created to make the integration as simple as possible.

In the following sections, you will see terms related to Kubernetes; you will get an more detailed overview of them in section 8.6. These are terms such as Kuberentetes object, operator, or API.

## 8.1 API endpoint

The interface for the client applications is a REST API developed in ASP.NET Core 7 with EF Core for relational object mapping over the MSSQL database. The RabbitMQ.Client nuget package is used to communicate with the RabbitMQ message broker. The application uses a software architecture similar to MVC [26], but without a view implementation. Views are completely replaced by the client applications, but those are handled separately. API is split into two Visual Studio projects: first contains bussinies logic (controllers, services, etc.) and second one defines all data models. Controllers receive HTTP requests containing individual input parameters that are mapped to objects called viewmodels. When mapping to a.viewmodel or directly after it, validations are performed. These include type checking, input length, or when uploading a file, checking for pairs of supported extensions and MIME Types.

If all input data is correct, the query processing can continue. In case of detection, the file is saved to a shared storage where the API endpoint is the only one with write permissions. Other parts of the framework can only read from this storage. The SHA256 checksum of that file is then calculated and compared against the database records. If this checksum is already in the database, the request ID containing this checksum is returned and the file is deleted. When the client queries the results, it gets the answer immediately because it is already in the database and the system is not unnecessarily overloaded with recomputation. However, if the checksum is not found in the database, a new request is created with the status **processing** and stored in the database. Subsequently, a message is sent via the message broker forcing the detection of this file. The message contains the same information as the database request (id, checksum, filename, status, type).

The OutputService class implements a function that handles the background consumption of messages containing results. When the server starts, it registers an event handler that is called when a message is received in the outuput queue. All the results are stored in the database and the status of the processed request is also changed to **done**. It is also necessary to delete the detected file from the shared storage. The incoming message contains the request ID and a list of responses, where each individual response contains the ID of the detection method and the resulting value. The list may also be empty, indicating an error or that a single method was unable to process the file. Detection methods can only

support processing certain file types, so if no method is able to process the file, the list will be empty.

The results are issued to the client upon request via the `/requests/results` endpoint. In case the request processing is not complete, an empty response (HTTP 204) is returned. After (un)successful processing, an object containing the request ID, the overall score, and a list of results consisting of the detection method ID, the detection method name, the description, the name of the training dataset, and the result value is returned to the client. Information about the detection method is read from the configuration file `appsettings.json` based on detection method ID. The client must perform pooling until it receives the first non-empty response [41]. The client should timeout in case of a complete system failure.

Configuration file `appsettings.json` contains connection strings to database and RabbitMQ, definition of supported file types and detection methods details.

## 8.2 Message broker

RabbitMQ was selected as the message broker, which is designed for asynchronous messaging. Individual messages are stored in a message queue where producers insert them. The messages are then read by the consumers. The AMQP protocol is used for communication, which defines the structure of messages, their acknowledgement, etc [54].

RabbitMQ is designed to send a large number of small messages. It would not be appropriate to send, for example, a whole file in BASE64 encoding through it. Therefore, the file to be detected is stored on shared storage and a message containing the remaining necessary parameters is sent through the message broker. Consumers are scaled based on the number of messages in each queue to handle requests as quickly as possible.

Four different message queues are used for communication: `queue_audio`, `queue_image`, `queue_video`, and `queue_output`. On the basis of the file type, the API Endpoint decides which queue the request should be placed in. The individual queues are then consumed by the individual Controllers in Request Processing units. The results are sent back to the API Endpoint through the output queue.

The RabbitMQ Cluster Operator provides a simple deployment to Kuberentes by defining new Kuberentes objects. The complete deployment manifest for RabbitMQ includes configuring the number of container replicas, default credentials, and resource usage limits. More details about deployment is in section 8.6. [56]

## 8.3 Processing unit

Controller is implemented as a single file Python script. The same functionality could be implemented in C# as well, but for this use case, using a scripting language was more straightforward. It is a script that receives a request and distributes it among the different detection methods. `Pika` package is used to communicate with the message broker.

It first establishes a connection and then connects to a given channel for which the QOS is set to 1. This indicates that it is able to consume only one message at a time. When calling the `start_consuming` function on opened channel, the script starts listening on that channel for new messages. When a message is being consumed, a new thread is created that does not prevent the library from functioning. `Pika` needs to be able to send regular heartbeat messages. [52]

The new thread will only start processing the request itself. Since the message sent is in JSON format, the request is mapped to a dictionary. Using `asyncio` and `aiohttp` packages, the HTTP endpoints of each detection method are called asynchronously. Since everything is stored in a single pod, the localhost interface is used for this communication. The ports of each detection method are read by Controller from an environmental variable that is defined in the Kubernetes manifest. The HTTP request contains all the defined parameters, and as a result an object containing the detection method id, and value (decimal number). When detection is complete, the individual results are stored in a list, which is converted again to JSON and sent to the output queue when all methods are finished.

## 8.4 Monitoring

In order to scale, it is necessary to have metrics on which to scale. Every cloud service provides some form of monitoring and metrics collection, but since we don't want to be dependent on the platform on which the framework will be hosted, we need to configure the framework to collect metrics on its own. For this case, the Prometheus platform was chosen.

Prometheus collects the defined metrics and stores them in a time series database. It can store metrics for several years. In our case, our interest was to collect RabbitMQ metrics that can be used for scaling and resource consumption of each cluster node. On the basis of the resource consumption, we are able to better allocate resources to different parts of the framework.

Prometheus provides what is called the Prometheus Operator, which, as with RabbitMQ, simplifies the deployment of this application. After deploying the operator to a Kuberentes cluster, it is possible to create objects of type Prometheus. For a description of the deployment, see 8.6. [40]

The RabbitMQ Cluster Operator automatically collects metrics, and we only need to configure API for providing compatible metrics to Prometheus. Then only all you have to do is to create new Kuberentes ServiceMonitor object which actually enables these metrics consumption by Prometheus. The metrics contain data about the number of connections, the number of open channels, the number of messages in each queue, and much more. Number of unconsumed messages in queue was chosen for scaling. This value needs to be exposed to the Kubernetes API, which is not done by default.

To do this, it was necessary to incorporate a Prometheus Adapter that reads the selected metrics and makes them available to the Kubernetes API. More about it is in section 8.5.

Node Exporter allows collecting metrics of resource usage of individual cluster nodes. If we configure Node Exporter container as a DeamonSet in a cluster, this container is then deployed to all running nodes in the cluster. The container must be configured with access to `/proc` and `/sys` virtual Linux directories from which it can retrieve individual metrics. Another ServiceMonitor object add these metrics to Prometheus the same way as for RabbitMQ. [11]

Prometheus is a good tool for collecting metrics, but its representation is not very good. In its web interface, it allows query calls that can be represented, for example, as a graph or just the current value. Grafana was chosen to represent the data from Prometheus.

Grafana is the only one that requires manual configuration. First, it is necessary to access the web interface and enter the default login credentials. After that, a request to change password will pop up. After the change is made, a new data source must be

configured in settings. Select prometheus and enter its domain name in the kuberentes cluster, which is “`http://prometheus-service.default.svc.cluster.local`”.

The user community shared the dashboards created on the official Grafana website, where it was possible to find the official dashboard for RabbitMQ<sup>2</sup> and also the community dashboard for Node Exporter<sup>3</sup>. Both dashboards provide a representation of a large number of metrics using charts, etc., which were sufficient for our needs. Fig. 8.2 shows Grafana web interface with Node Exporter dashboard. It would be possible to create a custom dashboard containing selected metrics if needed.



Figure 8.2: Grafana - Noede Exporter dashboard

## 8.5 Scalability

The Prometheus Adapter is designed deployment using the Helm tool, which extends the capabilities of the kubernetes manifest files. To keep the deployment as simple as possible and to avoid having to use many different tools, all deployment files have been copied from the Prometheus Adapter repository to the framework repository. With some modifications, it was possible to get Prometheus adapter working using only Kuberentes manifest files.

ConfigMap object for deploying Prometheus Adapter contains configuration such as name, which metrics and how they should be published, etc. Publication to Kubernetes API was achieved by creating a Kubernetes object of type APIService. Testing whole integration is possible by using the command [28]:

```
kubectl get --raw /apis/custom.metrics.k8s.io/v1beta2
```

Example results is shown in list. 8.1.

Horiznotal Pod Autoscaler needs several attributes to work. First, it is the identification of the object to be scaled and on what metric [50]. We have defined the name of the metrics in the Prometheus Adapter ConfigMap. Then we need to specify the minimum and maximum number of replicas and the average value. These values are then used in

<sup>2</sup><https://grafana.com/grafana/dashboards/10991-rabbitmq-overview/>

<sup>3</sup><https://grafana.com/grafana/dashboards/1860-node-exporter-full/>

calculation of number of replicas. In our case, minimum number of replicas is set to 1 and the maximum to 3. Average value of the messages in the queue is set to 2. These values can be modified according to the needs and environment in which the framework will operate. [50]

```
{  
    "kind": "APIResourceList",  
    "apiVersion": "v1",  
    "groupVersion": "custom.metrics.k8s.io/v1beta2",  
    "resources": [  
        {  
            "name": "jobs.batch/rabbitmq_metrics_audio_queue_depth",  
            "singularName": "",  
            "namespaced": true,  
            "kind": "MetricValueList",  
            "verbs": [  
                "get"  
            ]  
        },  
        ...  
    ]  
}
```

Listing 8.1: Result of calling the command to get metrics

## 8.6 Deployment

The deployment of the entire framework is divided into several Kubernetes manifest files, where individual Kubernetes objects are defined. These are persistent entities that represent parts of the system, and each object is of a specific type. The basic types are, for example, pod, service, deployment, etc. If we want to work with these objects, we can only do so through the Kubernetes API. It is possible to define a new API in Kuberentes and other new object types in it. This functionality is used by so-called operators, which are designed to automate repetitive procedures or to facilitate the deployment of a more complex system into another. When deploying this solution, it is important to consider that the RabbitMQ Cluster Operator and Prometheus Operator were used. Both operators have prepared manifest file which enables deployed via one command. [51]

Two bash scripts were created to make deployment and deletion easier. In deployment both operators are applied to the cluster and then the whole framework is deployed from all manifest files. Pulling all images, allocating resources and spinning up containers takes a while. It can take several minutes to start everything up and because some containers are started earlier then other ones, it can cause their failuer. Failed containers will be automatically restarted and the whole system then should be fully operational.

To communicate with the Kuberentes API, you can use `kubectl` tool, which provides options for creating or deleting objects, reporting cluster status, or more detailed descriptions of individual objects. Some parts of the framework require persistent storage, such as shared storage for files or databases. For this purpose, Kuberentes provides objects of type `PersistanceVolume` and `PersistanceVolumeClaim`. Kubernetes allows the creation of `StorageClasses` when installing a cluster, which automatically create a `PersistanceVolume`

when a request is made to create a PersistanceVolumeClaim. Different cloud providers provide different StorageClasses depending on the usage needs (speed, availability, backup, etc.), and it is possible to modify the configuration of these objects during deployment. Manual creation of PersistanceVolume without Storage Class is also possible, but requires more complex setup of the cluster itself or custom storage outside the cluster. We will not develop this option further.

The deployment of the framework, if we omit the Prometheus Adapter, is divided into ten different manifest files. Among the most important objects created are the deployment type objects that create individual pods. It allows you to set the number of replicas (automatically using scaling or manually) and can be connected to the Kuberentes load balancer, etc. Other essential objects types are Service, RabbitMQ, Prometheus, possibly ServiceMonitor, HorizontalPodAutoscaler and DeamonSet.

For deployment, it is recommended to use the prepared script `deploy.sh`, but if you need to manually interfere with the already deployed framework, you can use the tool “`kubectl apply -f <path>`” to apply a specific manifest file or folder. On the other hand, “`kubectl delete -f <path>`” is used for removal.

The Endpoint API and Processing Request Controller are built into a Docker image using Github Action, which is then published to the GHCR public container registry provided by Github. Any change pushed to the `main` branch of the repository will trigger this pipeline. These images are configured in Kuberentes manifests, which makes deployment of the latest version very easy. If only one component is being updated, such as the API Endpoint, just stop and start the component again with “`kubectl delete`” and “`kubectl apply`”, which will cause the latest version of the container to be downloaded from the container registry.

# Chapter 9

## Detection methods integration

Integrating the detection method only requires a container with an exposed REST API endpoint with one GET method (`detect`) on a specific port. This container image then needs to be correctly configured in the deployment files. Since this is a multicontainer pod, it is necessary that each detection method has its own port. There is a sample Dockerfile and one Python script with FastAPI library in the `template` folder. This script contains only one GET method with the given parameters mapped from the URL query parameters (id, checksum, filename, status, and type). Most of the parameters will probably not be used; the important one is filename, which is the file for detection saved in shared storage.

Template contains designed pipeline such as data preparation, detection and results normalization/generalization. If the method evaluates at any step of pipeline that it is unable to continue processing, either because of the file type or an internal error, it should correctly terminate the work and return `null` as the result. If processing is correct, it will return a detected value between 0 and 1. Result object should also return the ID of the detection method, which allows the Endpoint API to associate this result with a description of the detection method. This template does not need to be used and can be replaced by another technology more suitable for certain cases. It is a functional demonstration of how the integration should work.

```
- name: dfdf-detection-audio-2
  image: path/to/container/registry:main
  imagePullPolicy: IfNotPresent
  resources:
    requests:
      memory: "3.25G"
      cpu: "500m"
    limits:
      memory: "3.25G"
      cpu: "500m"
  volumeMounts:
    - name: processingdata
      mountPath: /mnt/processingdata
      readOnly: true
```

Listing 9.1: Container configuration in Kubernetes manifest

Subsequently, the container must be built and published to a registry accessible from the Kuberentes cluster. Depending on the type of detection method, it must be con-

figured in a given deployment file. The project has three different deployment files according to file type: `processing-unit-voice.yaml`, `processing-unit-image.yaml`, and `processing-unit-video.yaml`. It should be mentioned that the size of the resulting container should not be too large. Otherwise, it causes problems with the speed of the deployment itself. If deployment of Request Processing unit is slow, then scaling will not be so effective. The Docker image must be downloaded from the registry, and this takes some time; the smaller the image, the less this operation will affect the speed of creating a new container.

Adding a new container to the list in the “`spec.template.spec.containers`” section ensures that it will be part of the pod. You need to define the Docker image, the amount of resources the container can consume, and you also need to mount a shared repository where the files for detection will be stored. Storage should be mounted in read-only mode. In list. 9.1 you can see an example of configuration of one detection method. Controller needs to know that a new method exists so that the port where API is exposed has to be added to environmental variable for the Controller container. The variable is called „`ProcessingUnitsPorts`“.

```
{
    "ID": 1,
    "Type": 1,
    "Name": "Detection Method XX",
    "Description": "This is awsome detection method...",
    "TrainingDataset": "Training dataset XYZ"
}
```

Listing 9.2: Detection method description in `appsettings.json`

In the last step of the integration, you need to update the `appsettings.json` file in the Endpoint API by adding the detection method description. The description includes the ID, which must be unique, type, name, description, and the name of the training data set. The type can take values: 0 = audio, 1 = image, 2 = video. An example can be seen in list. 9.2.

## 9.1 AudioDeepFakeDetection

This project<sup>1</sup> implements different types of models and combines them with different feature extractors. The project provides six different neural network models and three different feature extraction methods, providing a considerable number of combinations to create different detection methods. This project was chosen because it offered great potential for trying different detection methods and the code was relatively easy to understand at first glance. The project describes well the steps needed to make it work, which led to success without any problems after replication.

The problem with most open source detection methods is that they are scientific papers and not tools for ordinary users. For optimization, these tools work with batches of files at a time, both in learning and in verifying the results. This leads to the fact that these methods cannot verify only one file as a rule. This is not different in the case of AudioDeepFakeDetection. Therefore, it was necessary to start modifying the code. A fork<sup>2</sup>

---

<sup>1</sup><https://github.com/MarkHershey/AudioDeepFakeDetection>

<sup>2</sup><https://github.com/PlayerBerny12/AudioDeepFakeDetection>

of the repository was created in which this change was made. A function was created to allow only one file to be processed.

The project provided pre-trained models for download, but these were not used. The datasets used for training were LJ Speech and WaveFake. A more detailed description of the datasets can be found in section 11.1. When replicating the experiment, 6 different detection methods were eventually created. Combinations of the most successful models and feature extractors based on output of their experiment were selected. Each method was trained for 15 epochs. All methods had the highest achieved accuracy of over 90 % on the test dataset, some even very close to 100 %. The training results are presented in tab. 9.1. The first part of the name indicates the name of used model, the second part is name of the feature extractor, and the last part can take two values: I = in distribution, O = out distribution.

Experiment	Accuracy	F1	ROC AUC	EER
ShallowCNN_mfcc_O	0.942	0.941	0.9418	0.0743
ShallowCNN_lfcc_O	0.949	0.949	0.9494	0.0647
TSSD_wave_O	0.958	0.958	0.9576	0.0462
TSSD_wave_I	0.997	0.997	0.9969	0.0042
ShallowCNN_mfcc_I	0.997	0.997	0.9971	0.0053
ShallowCNN_lfcc_I	0.999	0.999	0.9994	0.0008

Table 9.1: AudioDeepFakeDetection training results

The training and test dataset are created automatically from the two provided datasets. The split was in the simplest possible way, the first part of the data was labelled as training, and the second as testing. The parameter “in\_distribution” also influenced whether the distribution of deepfake and originals was 1:1 or 7:1 in favor of deepfakes. Training 15 epochs on the Nvidia GeForce GTX 1660Ti graphics card took around 9 hours. For example, the model ShallowCNN\_lfcc\_I was trained from 8:04 AM to 4:58 PM (final time 8:54). All training logs can be found in the repository in the `saved` folder.

For the integration, 6 different Docker images had to be created. Prepared templates were used for this operation. An automatic build of the images in Github Actions when pushing any change to `master` branch was implemented. The images are then published in the GHCR container registry and can be embedded in the framework.

This project is built on the PyTorch library and the official PyTorch Docker image was used as the base for the image creation. After building this image, everything worked correctly and all methods processed the files in the framework in parallel. However, a problem arose, downloading the images to a given node. It turned out that the base PyTorch image was several gigabytes in size and with the model embedded, downloading the necessary pip package had a resulting size of 10.59 GB. Although some layers are shared between images, it took several minutes to initialize all 6 detection methods.

It was necessary to proceed to optimization. The PyTorch Docker image as the base image was replaced with a Python image. This was because it turned out that PyTorch contained all the necessary libraries to run on the graphics card, which were not needed. PyTorch was installed in the Python image to run only on the CPU and other minor optimizations were made, such as clearing the memory cache after installing the package, etc. Due to these changes, the size was reduced to 1.48 GB. This optimization reduced the

deployment time to a maximum of tens of seconds in an environment with good connectivity from the cluster to the container registry.

## 9.2 fakeVideoForensics

The integration procedure for this project<sup>3</sup> was the same as in the previous methods, perhaps a little simpler. Again, prepared templates were used, and an automated build was created using Github Actions with image publishing to GHCR container registry.

This method focusses on video detection, but provides no description of how the model should be trained. The repository probably does not even contain a script that could be used for this purpose. However, it was possible to use an already pre-trained model. This project focused on processing only one file, so no code changes were necessary, still fork<sup>4</sup> of this repository was created.

The lesson learnt from previous integration of detection methods was to optimize the image size as a second step. The image created on the first attempt had 4.81 GB. We tried to use PyTorch without the ability to run on the graphics card, deleting the cache, etc. The success was not that great in this case, but we still managed to reduce the image size to 3.63 GB.

---

<sup>3</sup><https://github.com/afuentesf/fakeVideoForensics>

<sup>4</sup><https://github.com/PlayerBerny12/fakeVideoForensics>

# Chapter 10

## Client application implementation

A browser extension is actually a web application that has minor specifics. A must have is a „Web app manifest“ that specifies what attributes and permissions the extension has. The basic attributes are name, version, default popup file (application’s entrypoint file), etc. One of the permissions is the ability to access the DOM on the currently opened/hosted page. This allows it to read and search for certain elements on the page or even modify them.

As the extension does not require any specific permissions in the base, it is a normal web application. If interactive selection of elements on the page is implemented, then the application would require the use of special browser APIs.

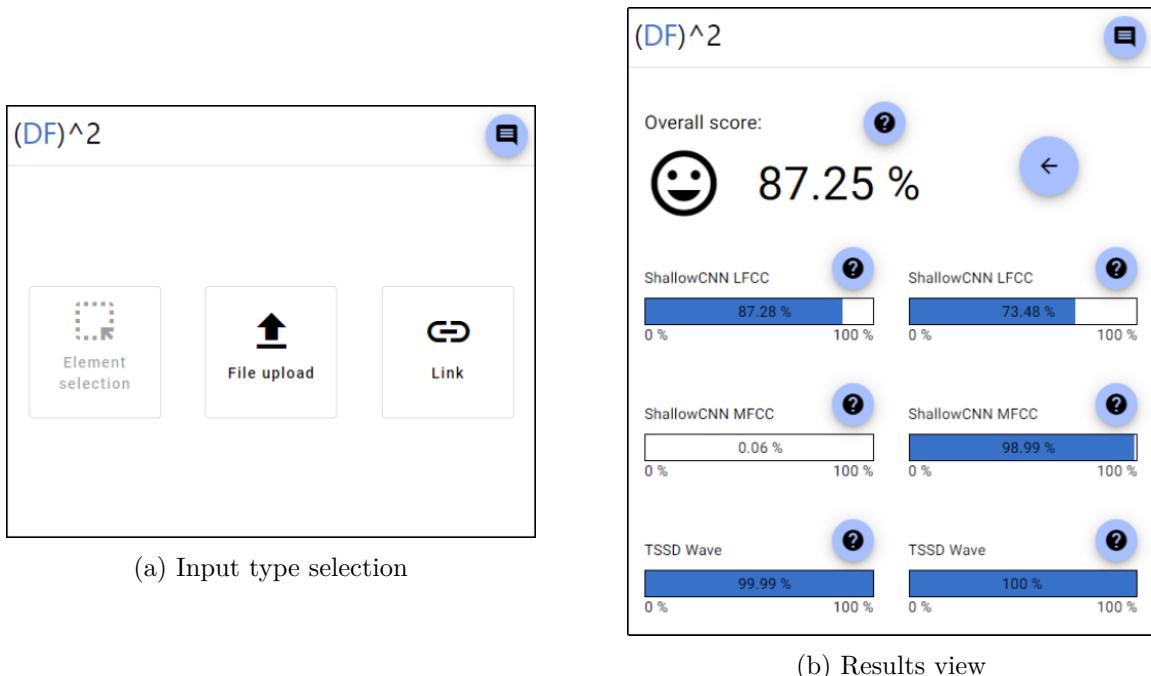


Figure 10.1: Web extension implementation

The Angular Material framework with the Angular Material component library was used for implementation [15]. This made it possible to quickly create a simple application that is visually appealing, but its main purpose was focused on simplicity and understandability. You can see how application looks like in fig. 10.1.

## 10.1 Functionalities

The app provides all the proposed functionality, but not all potential designed features were covered. Thus, it is not possible to interactively select an element on the page to detect. Otherwise, it is possible to upload your own files or links. Once the detection is started the application displays a loading screen. When the request is processed, the results are displayed. The implemented application tried to keep the layout and functionality designed in wireframes, but it was not always technically possible.

One difference from the design is the use of a dialog box instead of a tooltip. Angular Material does not allow tooltip creation, and the use of another 3rd party library was deemed inappropriate for subsequent application maintenance. This is a minor change that is more technical and does not affect the functionality in any way.

The second difference was in moving the button to send feedback from the bottom of the application to the top. During development, this location was identified as inappropriate because it obscured the results bars.

## 10.2 Communication with framework

The most specific part of the implementation of the application was communicating with the framework and waiting for the detection to complete. The application uses the so-called pooling, it is a technique of repeatedly sending messages to the server to retrieve certain information [41]. The application resends the message to the REST API using the `/requests/results` method and waits until it receives a non-empty response. Since processing is expected to take some time, the pause between calls was chosen to be two seconds.

In the extreme case, a framework crash could occur, so a timeout should be implemented to notify user something is wrong. The timeout value was set to 20 min. The timeout should not really occur and it is a last resort.

This communication could be replaced by websockets in the future, where resource wastage would be reduced in some way and the server could notify completion as soon as it receives processed results. Managing websockets is not free, but after implementation, it would be possible to compare which technique is more efficient.

## 10.3 Supported browsers

The extension does not use a specific API to communicate with the browser, making the application more easily portable between different browsers. The functionality was tested in Google Chrome (v113), Microsoft Edge (v113) and Mozilla Firefox (v113). The extension worked without errors in all browsers with one exception. It is not possible to upload a file in Mozilla Firefox because when the file picker is closed, the extension itself is also automatically closed and the user is unable to view the results page in this case. This is bug<sup>1</sup>, which after a long time might be closed in one of the upcoming versions. Using the link to detect it works without any problems.

Using the `npm run build` command, the project is built and in the `dist` folder, and it is possible to load the extension in the browser when using browser in developer mode.

---

<sup>1</sup>[https://bugzilla.mozilla.org/show\\_bug.cgi?id=1292701](https://bugzilla.mozilla.org/show_bug.cgi?id=1292701)

# Chapter 11

## Test experiment and results

Three different test scenarios were created to test the reliability of the framework, the resources consumed, the workload, and the accuracy of the detection methods. Five different datasets were used to create data for each individual test scenario. In total, more than 2200 files were run through the framework during the tests.

The following sections show the CPU load and RAM usage graphs from Grafana, unfortunately, the labels are hard to read in the document. The graphs should have a sufficient textual description to compensate for this deficiency. Alternatively, high-resolution images are available on the included media, where the labels should be readable. In the case of the CPU load graph, the colors represent the following properties/values: yellow - busy system, blue - busy user, red - busy lowait, and green - idle. On the other hand, for RAM usage, the colors represent: yellow - RAM used, blue - RAM cache + buffer, green - RAM free. CPU load is in percentages from 0 to 100 % and RAM usage from 0 to 32 GB.

### 11.1 Datasets

Five different datasets were used for testing; three audio datasets: LJ Speech<sup>1</sup>, WaveFake<sup>2</sup>, and ASV2021<sup>3</sup>, and for video: FaceForensics++<sup>4</sup> and CelebDF<sup>5</sup>. Some datasets were used for training of detection methods; these data were not used for testing where even possible. For fakeVideoForensics, it was not possible to verify which dataset data was used for training, hence the accuracy of the method may be slightly higher in our tests. To make the tests more provable, data from a dataset on which no learning was performed were always used.

For the first test scenario, 500 recordings/videos were selected from a given dataset, where 250 were deepfakes and 250 were originals. Thus, four sets of 500 files were created. For stress testing, 75 audio recordings and 25 from the WaveFake and CelebDF datasets were pseudo-randomly selected. In this case, it was not the detection result that mattered, but the processing time and consumed resources.

The ASV2021 dataset contained FLAC recordings, but the detection methods included in the framework were unable to process this format, so they were converted to WAV format using the `ffmpeg` tool.

---

<sup>1</sup><https://keithito.com/LJ-Speech-Dataset/>

<sup>2</sup><https://github.com/RUB-SysSec/WaveFake>

<sup>3</sup><https://www.asvspoof.org/index2021.html>

<sup>4</sup><https://github.com/ondyari/FaceForensics>

<sup>5</sup><https://github.com/yuezunli/celeb-deepfakeforensics>

The LJ Speech and WaveFake datasets do not achieve high quality from today's perspective, because LJ Speech contains recordings of only one person [25]. WaveFake is then a dataset generated based on LJ Speech applying different generation techniques. Perhaps this is why the detection methods in AudioDeepFakeDetection achieved such high values during training. Therefore, it will be interesting to see how these detection methods perform against ASV2021 recordings.

## 11.2 Test case 1 - accuracy

The first test scenario tests the reliability of the framework and the accuracy of each method. Four iterations of this scenario were performed, where each input data was from one dataset. One iteration took several hours to perform. Thus, one iteration contained 500 files (250 originals, 250 originals). The files were sent sequentially for detection to the framework, that means always waiting for one file to finish processing before starting the next one. For each item, the processing time, the results of each detection method, and the file size were measured. ROC curves with AUC values were generated for each iteration and each detection method.

Four different overall score calculations were also tested, with the best one to be selected and included in the framework. The first calculation used a weighted average, in which it prioritized the minimum value (deepfake) of all the resulting ones. The second calculation, on the other hand, prioritized the maximum value (original) using the weighted average. The third calculation took the average value of all methods. The last calculation combined all the previous three variants, and the calculation depended on how many methods were above 50 % value and how many methods were below it.

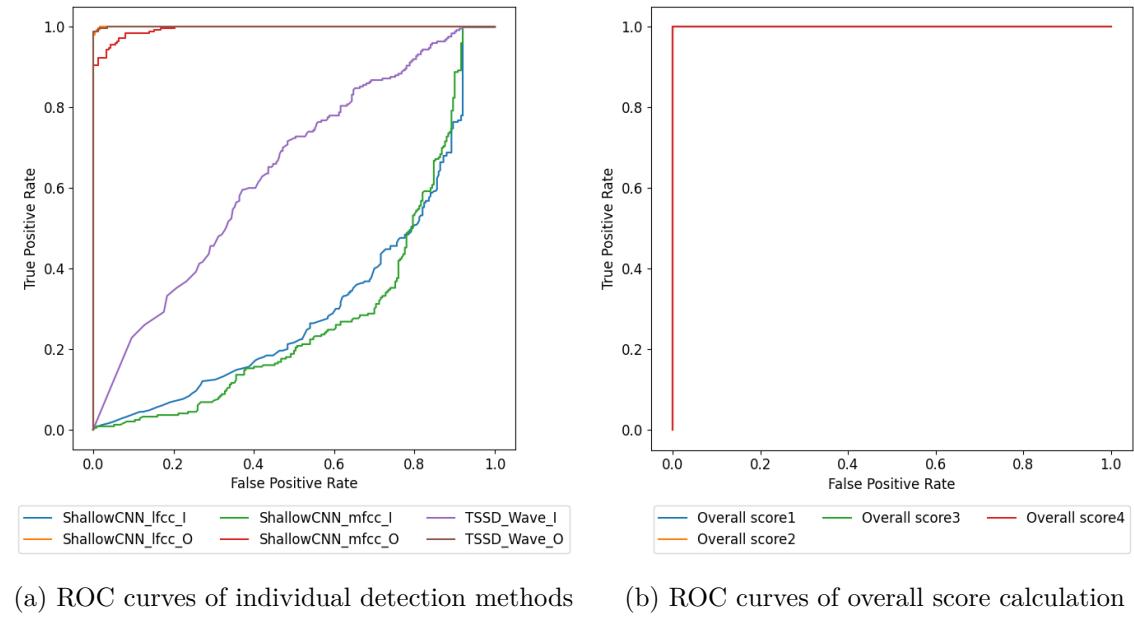


Figure 11.1: Test results for LJ Speech and WaveFake data

The first iteration was performed with data from the LJ Speech (250 originals) and WaveFake (250 deepfakes) datasets. AudioDeepFakeDetection automatically split data in a very primitive way into training and testing sets. In the case of the „in distribution“ type,

only one type of deepfake was used to be detected. Thus, we pseudo-randomly selected data from the testing part of the dataset.

Some methods achieved very decent results; these are the methods that were „out of distribution“. The „in distribution“ methods were trained on only one type of deepfake, which was minimally represented in the test data. The ShallowCNN model trained on only one type of deepfake was unable to generalize, and its results were very poor. All calculations of the overall score achieved a perfect classifier. Results are shown in fig. 11.1.

In table 11.2 you can see statistics on the test data and the processing time.

Detection method	AUC
ShallowCNN_lfcc_I	0.313040
ShallowCNN_lfcc_O	0.999808
ShallowCNN_mfcc_I	0.289784
ShallowCNN_mfcc_O	0.994016
TSSD_Wave_I	0.641496
TSSD_Wave_O	0.999760
Overall score1	1.0
Overall score2	1.0
Overall score3	1.0
Overall score4	1.0

Size average	288.3 kB
Size median	299.5 kB
Size minimum	67.9 kB
Size maximum	445.0 kB
Time average	72.9 s
Time median	77.9 s
Time minimum	16.6 s
Time maximum	85.4 s

Table 11.2: Test data and processing time statistics for LJ Speech and WaveFake data

Table 11.1: AUC values for LJ Speech and WaveFake data

The total CPU load exceeded 60 % almost all the time, of which about 20 % was taken by system load and the remaining 40 % by applications from user space (see fig. 11.2). System load is likely to include resource management, throttling, context switching, etc. For RAM usage, we can see spikes reaching up to 30 GB if you include RAM Used and RAM Cache + Buffer (see fig. 11.3). Memory is always freed up between detection runs, and we can see that PyTorch runs are relatively RAM intensive.

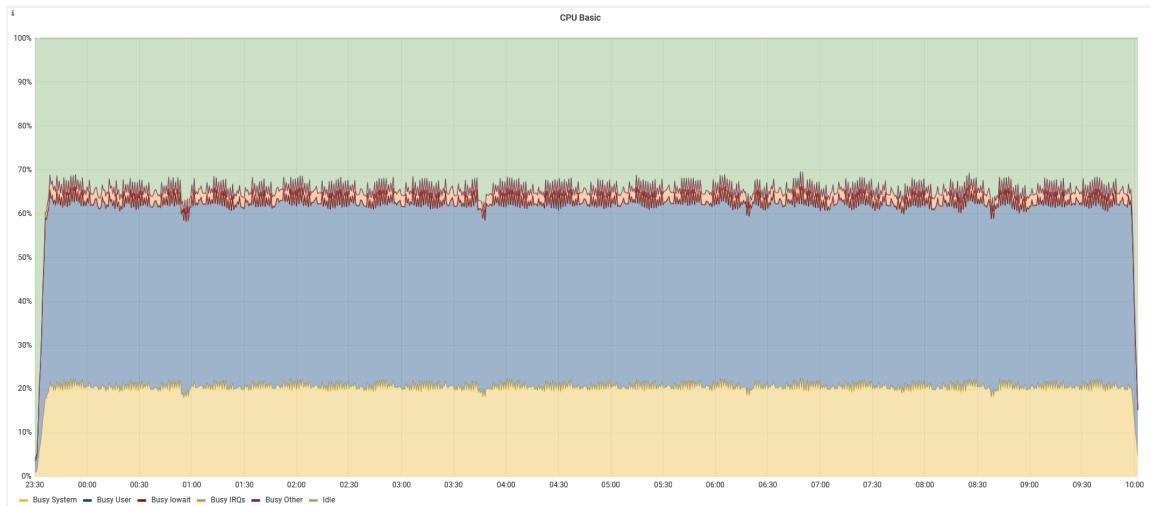


Figure 11.2: CPU load for LJ Speech and WaveFake data

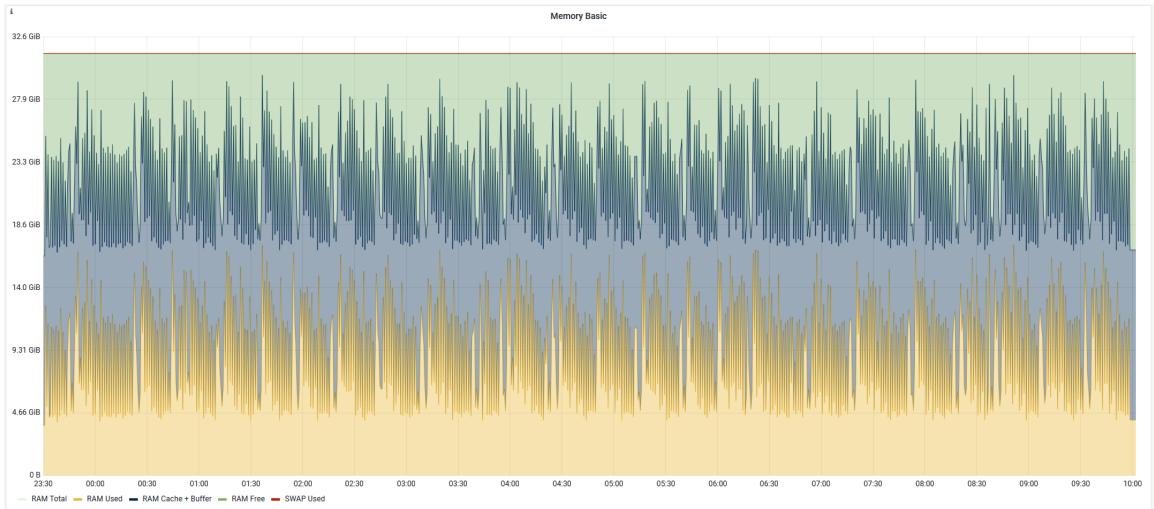
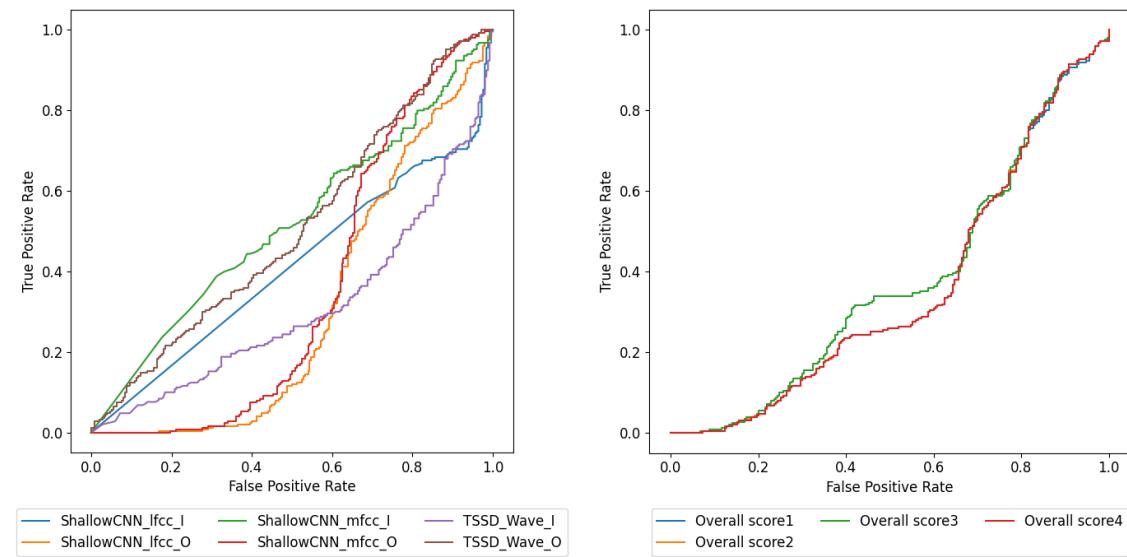


Figure 11.3: RAM usage for LJ Speech and WaveFake data

The second iteration used the ASV2021 dataset, which is divided into three parts [9]. One part is the deepfakes, the second part is the original recordings, and the third part is the original recordings played back and resampled by the recording device. All three parts were used in the tests and where the originals and the re-recorded originals were in a 1:1 ratio.



(a) ROC curves of individual detection methods

(b) ROC curves of overall score calculation

Figure 11.4: Test results for ASV2021 data

The results are no longer as perfect as in the first iteration (see fig. 11.4a). It could be said that all the detection methods in this case are at the level of a random classifier. If we compare the two iterations, the TSSD\_Wave\_0 detection method performs the best. First iteration confirmed results of AudioDeepFakeDetection experiment results but second iteration does not show such good results. We can say that those methods are not par-

ticularly beneficial for real-word application. Had these methods been trained on a more robust dataset, they might have had better results.

Detection method	AUC
ShallowCNN_lfcc_I	0.410944
ShallowCNN_lfcc_O	0.305408
ShallowCNN_mfcc_I	0.517672
ShallowCNN_mfcc_O	0.349888
TSSD_Wave_I	0.309056
TSSD_Wave_O	0.507360
Overall score1	0.353648
Overall score2	0.354896
Overall score3	0.376416
Overall score4	0.354496

Size average	87.5 kB
Size median	78.5 kB
Size minimum	19.1 kB
Size maximum	265 kB
Time average	49.5 s
Time median	76.1 s
Time minimum	16.3 s
Time maximum	96.1 s

Table 11.4: Test data and processing time statistics for ASV2021 data

Table 11.3: AUC values for ASV2021 data

Interestingly, the calculations of the overall score are not particularly different, and the best performance overall score is based on a average (see fig. 11.4b). Perhaps if individual detection methods had performed a little better, then comparison of the total score calculation would be more interesting. In this case, we cannot objectively declare one calculation to be the best. We think the fourth method might have potential and will therefore be implemented in the framework.

The video detection method is only one integrated method, so we have no comparison. We could try to compare the results with Deepware, but unfortunately we did not get access to their API. We did a small manual test, but its comparison would not have much informational value.

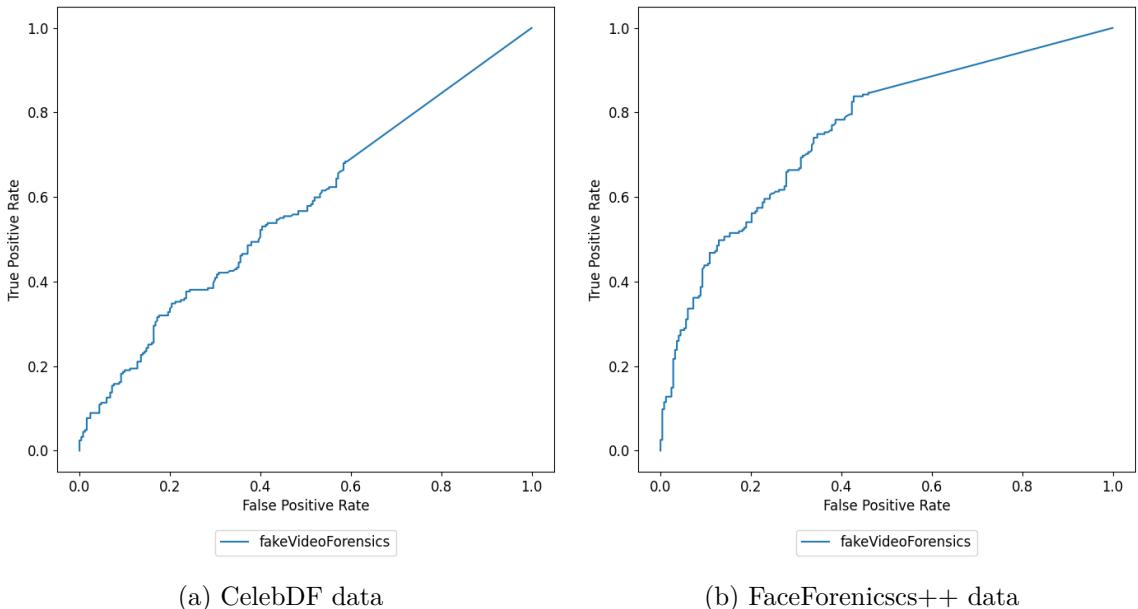


Figure 11.5: ROC curves for video detection

The third iteration was done with selected data from the CelebDF dataset, and the fourth iteration from FaceForensics++. This detection method used FaceForensics++ dataset for learning, but unfortunately nowhere is it described what data were used for training. Therefore, the data selection was again done pseudo-randomly.

The AUC values for the CelebDF data were 0.5738 and for the FaceForensics++ data 0.7584. As mentioned, the result for FaceForensics++ may be a bit biased because we do not know how much our test data overlap with the real training data. However, a result of around 60 % could be considered acceptable.

Size average	1893 kB
Size median	1674 kB
Size minimum	467 kB
Size maximum	5992 kB
Time average	153.8 s
Time median	145.7 s
Time minimum	34.6 s
Time maximum	496.8 s

Table 11.5: Test data and processing time statistics for CelebDF data

Size average	2031 kB
Size median	1559 kB
Size minimum	449 kB
Size maximum	10694 kB
Time average	389.5 s
Time median	305.3 s
Time minimum	126.4 s
Time maximum	881.7 s

Table 11.6: Test data and processing time statistics for FaceForensics++ data

Video processing took much longer and is at the limit of what is bearable for the average user. Averaging around three 3 minutes for the CelebDF data (see table 11.5) and 6.5 min for FaceForensics++ data (see table 11.6). Yet the videos are no means larger, but very likely a more efficient codec or more compression is used, and the videos are longer.

On the CPU and RAM usage graphs (see fig. 11.6, 11.7), we can see that this method is slightly more efficient in RAM consumption, and hence we do not see the various spikes in the graph. Since there is no need to solve multiple detection methods in parallel in different containers, the CPU usage by the system is minimal and 20 % is taken by the detection method itself.

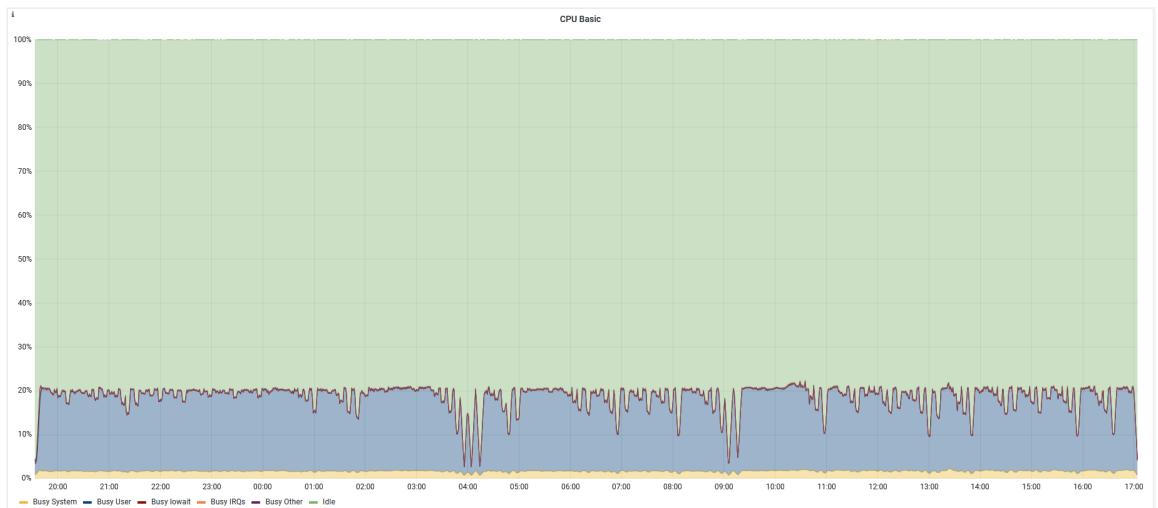


Figure 11.6: CPU load for CelebDF data

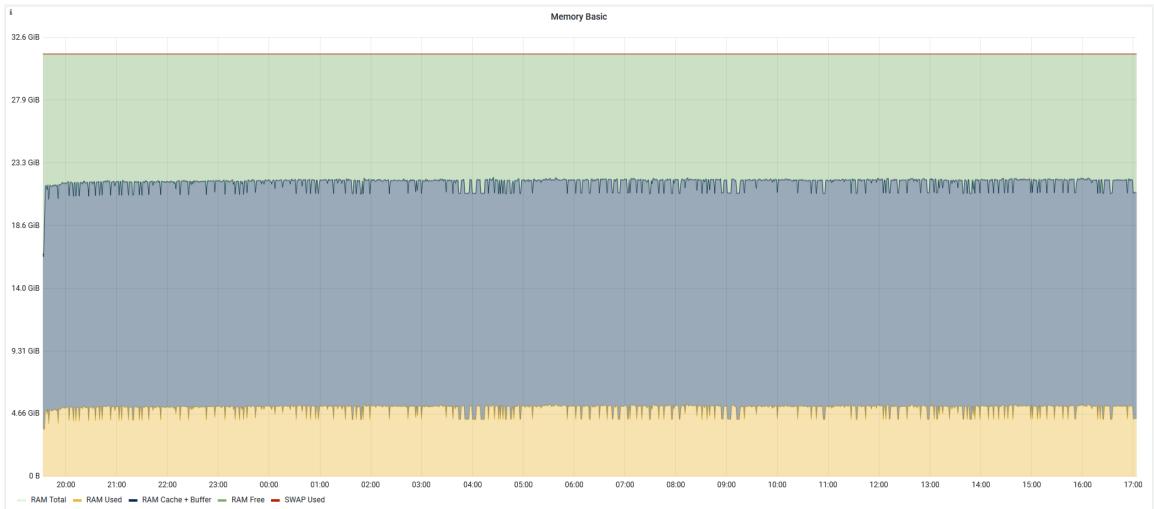


Figure 11.7: RAM usage for CelebDF data

All files were processed by the framework, with a few exceptions where the file was not processed due to timeout. These were just a couple of videos. Compared to Deepware which did not process a large portion of the files, from a reliability perspective, the framework comes out better. The accuracy of the detection is questionable.

### 11.3 Test case 2 - small bursts

The second test no longer looked at the accuracy of the detection methods, but only at the resource usage in the workload. In this test, 100 files (25 videos and 75 audio recordings) were used. These files were shuffled and sent for detection in batches containing 5 files. Subsequently, all files were waited for evaluation before sending another batch.

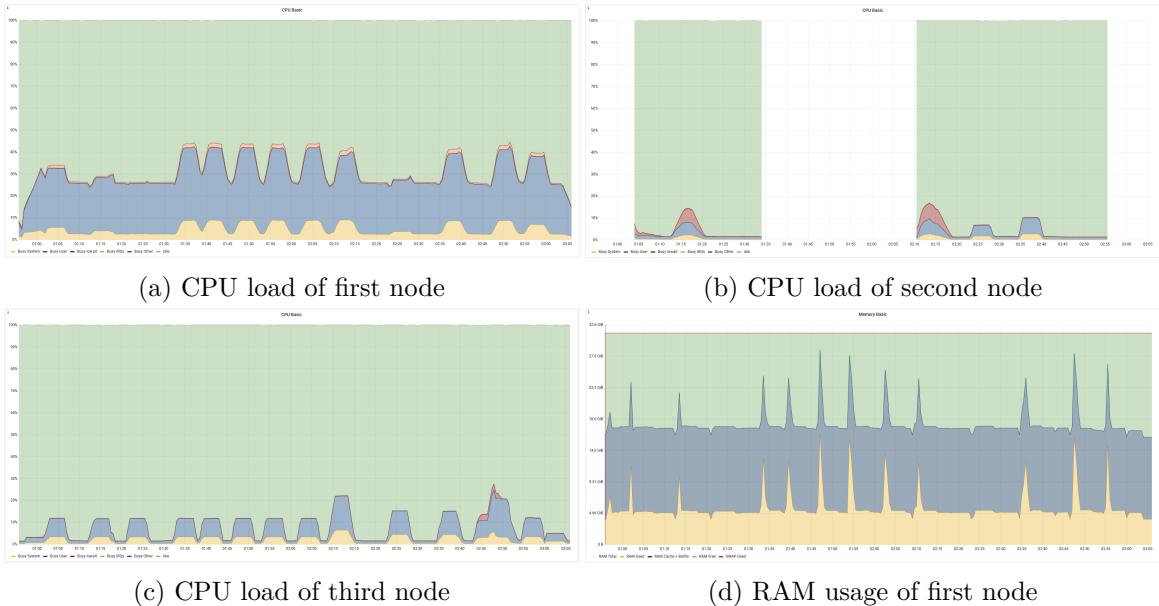


Figure 11.8: Resource consumption during test scenario 2

It took 2 hours 8 minutes to process all files. In graphs in fig. 11.8, we can see that three nodes were used during the test. Not all of them worked all the time, but this fact confirms that the scaling works, the load was spread over multiple nodes, and in some cases the processing was done in parallel. Five files are borderline to effectively activate scaling, and these are more likely to be cases where 4 audio recordings and one video were sent for detection.

This test was repeated one more time to test if storing the results in the database would relieve the load on the system. This time the test lasted 2 minutes 41 seconds and according to the graphs (see fig. 11.9) the system hardly recognized that 100 files were detected.

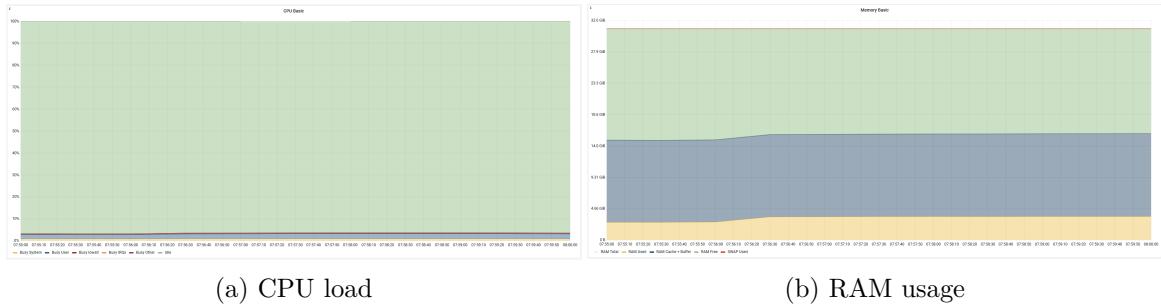


Figure 11.9: Resource consumption during test scenario 2 with cached results

#### 11.4 Test case 3 - congestion

The last test used the same data as the previous scenario; the database was deleted so process was unable to use the already calculated results. In this scenario, all 100 files were sent at once to verify how the framework performs under load.

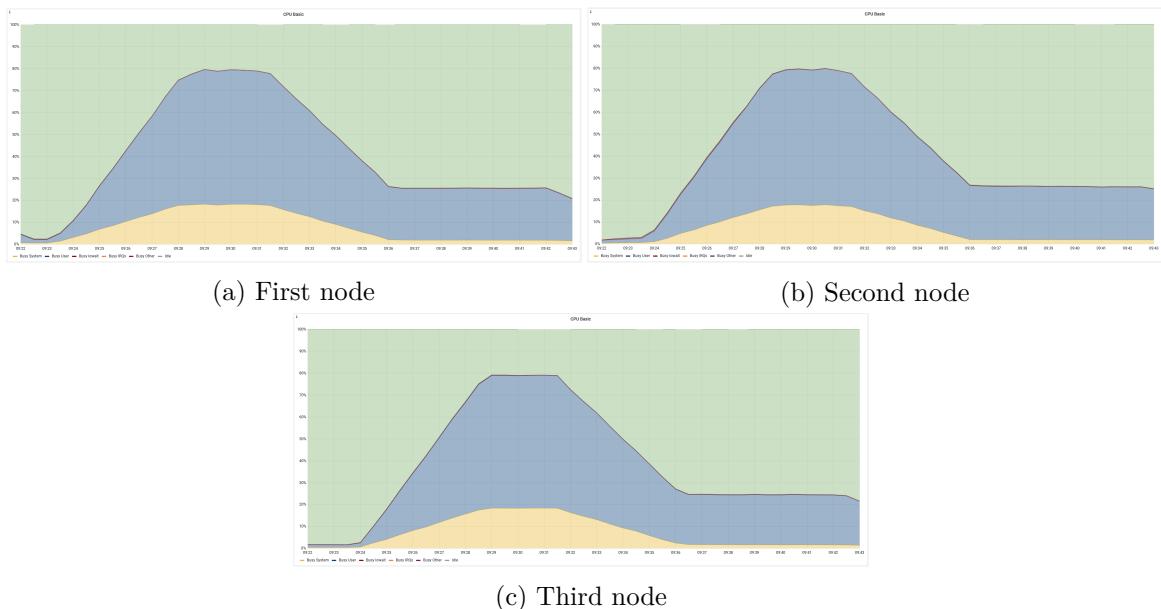


Figure 11.10: CPU load during test scenario 3

We can see that three nodes were activated and that the CPU load of all three was around 80 % for the time that the audio and video files were processed. Even though there were more audio files, the video processing took much longer and we can observe that the load dropped dramatically in the middle, which lasted another few minutes (see fig. 11.10). A similar trend can be seen for the RAM load (see fig. 11.11).

The framework was able to process all 100 files in a time of 19 minutes, 31 seconds. This is a rapid acceleration over the previous scenario. We can observe that the scaling of the system is fast and helps the processing speed under load to be very substantially.

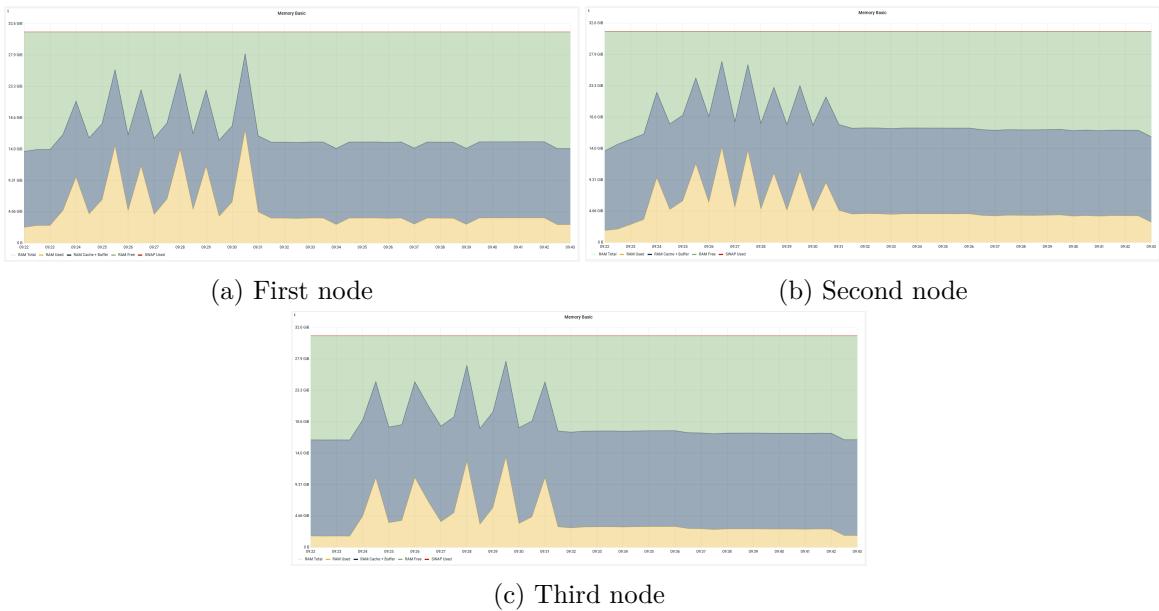


Figure 11.11: RAM usage during test scenario 3

## 11.5 Tests evaluation

In the first scenario, we can see that the system is not efficiently loaded and resources are being wasted. At first glance it may look like this, but each method has access to a limited amount of resources, so when the system is under load it will work reliably. Therefore, sequential utilization is not appropriate.

Kubernetes can be set up to have individual containers contend for resources, but this greatly increased the management overhead of the system and did not produce a performance increase. The resource allocation can be easily changed, and certainly the system could be set up little bit better based on these tests. It will also depend greatly on the particular HW that will be used. Still, the framework works reliably, and especially its good features will only show up in a real workload.

The system had to run with 32 GB of RAM; otherwise, the audio detection methods failed because of lack of memory. Each method must have at least 3 GB of memory guaranteed, better a bit more. All methods run in parallel at the same time, which puts a lot of demands on the host platform. This problem could be fixed by changing the Request Processing Controller so that all six detection methods do not run in parallel but in smaller batches. This would increase the processing time, but it would be possible to run on a platform with only 16 GB of RAM.

The video detection method was not very fast in processing, so recommendation for use in the production version would not be granted. Some optimizations would need to be made. Audio detection methods on the other hand did not provide great detection accuracy, this could be changed by using a more robust dataset for training.

Although the individual methods did not achieve high quality, the implementation and tests can be declared successful. The framework can perform its work reliably for several hours and the scaling efficiency becomes apparent under load. As a result, the framework is able to handle a large number of files in a relatively short time. The only problem is the use of more powerful HW when integrating a large number of methods, this could be partially reduced by changing Request Processing Controller or possibly optimizing individual methods. The deployment of the framework strongly depends on the integrated methods and, used hardware.

## 11.6 Cost analysis

The framework with describe detection methods require HW with 32 GB of RAM for its functionality, and the use of neural networks is a computationally demanding operation. So minimal recommendation is to use at least 4 CPUs. In the end, the B8ms virtual machine was chosen for deployment in Azure AKS. This machine provides 8 CPUs, 32 GB RAM and costs \$0.38 per hour to operate.

A system with no load needs one such machine, but if there is a load and scaling is applied, there may be more of these machines running at the time. In our case, we have set the maximum number of nodes to 3. Therefore, if the framework were at 100 % load, an hour of three of these machines would cost \$1.14. The framework also requires external IP, load balancer, and storage. These items in our case cost a tenth of what virtual machines cost.

The framework was tested for several days and scaling was applied at certain stages. One day cost about \$18.8 on average to run this framework. That would work out to a monthly operation of over \$500. In our case, we were using a free cluster that is designed for development and testing. The standard cluster is chargeable but provides a more stable environment that is focused on speed, availability, and reliability. The cost of the standard cluster is \$70<sup>6</sup>, so the total cost would be around \$600 per month.

If a machine with only 16 GB of RAM could be used and more scaling could be utilized, savings could be realized while maintaining similar properties. Alternatively, utilize a custom HW, where the purchase price of the HW and its potential lifetime, maintenance, and cost of operation would have to be factored into the price.

---

<sup>6</sup><https://azure.microsoft.com/en-us/pricing/calculator/?service=kubernetes-service>

# Chapter 12

## Conclusion

The aim of the work was to study the deepfakes and create a framework that allows the integration of different detection methods. A client application (web extension) which present the functionality of the framework was also part of the goals.

The topic of deepfake is very broad, and even the assignment of this thesis contained many different tasks that were successfully completed. In the first part, techniques for creating and detecting different types of deepfakes were covered. One of the tools was tested, but unfortunately a comparison with the created tool would not provide much informational value.

The framework was successfully implemented according to the proposed architecture, from which it did not deviate in any way. The development of such a tool was not a simple, as not all things could be tested on a local station. The microservice architecture allowed for development and testing of individual parts, but the overall framework had to be tested and debugged in the cloud.

Most of the work was spent developing the framework, which is not visible to the average user. On the one hand this is probably correct. The whole system is very complex and consists of a large number of components, but the user interacts only through the client application. On the basis of the tests, the user should not be able to tell much difference when the framework is under load or not.

Personally, I had little experience with containerization, but this was the first time I came in contact with an orchestrator like Kubernetes. In doing this thesis, I gained a lot of valuable experience in developing and deploying microservice architectures and containerization itself. Many different technologies have been used, the use of which may have been questionable at first glance. I hope I have managed to dispel any doubts and explain why the framework was designed and implemented in this way.

I would like to end this paper with a quote from Alan Turing that I have remembered since I first read it:

“Sometimes it is the people no one can imagine anything of who do the things no one can imagine.”

# Bibliography

- [1] *What Is the Necessity of Bias in Neural Networks?* [<https://www.turing.com/kb/necessity-of-bias-in-neural-networks>]. [cit. 2022-12-29].
- [2] AHMED, S. Who inadvertently shares deepfakes? Analyzing the role of political interest,cognitive ability, and social network size. *Telematics and Informatics*. 2021, vol. 57, p. 101508. DOI: <https://doi.org/10.1016/j.tele.2020.101508>. ISSN 0736-5853. Available at: <https://www.sciencedirect.com/science/article/pii/S0736585320301672>.
- [3] ALEX GAMELA, R. F. *Podman vs Docker: What are the differences?* [<https://www.imaginarycloud.com/blog/podman-vs-docker/>]. [cit. 2023-05-06].
- [4] ALEXANDRA BARUFFATI. *Chat GPT Statistics 2023: Trends And The Future Perspectives* [<https://blog.gitnux.com/chat-gpt-statistics/>]. [cit. 2023-05-04].
- [5] ALMUTAIRI, Z. and ELGIBREEN, H. A Review of Modern Audio Deepfake Detection Methods: Challenges and Future Directions. *Algorithms*. 2022, vol. 15, no. 5. DOI: 10.3390/a15050155. ISSN 1999-4893. Available at: <https://www.mdpi.com/1999-4893/15/5/155>.
- [6] AMAZON. *Containers at AWS* [<https://aws.amazon.com/containers/>]. [cit. 2023-05-06].
- [7] AMOS,ZACHARY. *Hybrid Vishing Attacks Skyrocketing: What to Know* [<https://itsupplychain.com/hybrid-vishing-attacks-skyrocketing-what-to-know/>]. 2022 [cit. 2022-12-28].
- [8] CLOUDZERO TEAM. *Horizontal Vs. Vertical Scaling: How Do They Compare?* [<https://www.cloudzero.com/blog/horizontal-vs-vertical-scaling>]. [cit. 2023-05-9].
- [9] DELGADO, H., EVANS, N., KINNUNEN, T., ET AL.. *MVC Framework Introduction* [<https://www.asvspoof.org/index2021.html>]. [cit. 2023-05-15].
- [10] DEOTTE, C. *How to choose CNN Architecture MNIST* [<https://www.kaggle.com/code/cdeotte/how-to-choose-cnn-architecture-mnist>]. [cit. 2023-01-04].
- [11] DEVOPSCUBE. *How to Setup Prometheus Node Exporter on Kubernetes* [<https://devopscube.com/node-exporter-kubernetes/>]. [cit. 2023-05-10].

- [12] FERRARA, M., FRANCO, A. and MALTONI, D. The magic passport. In: *IEEE International Joint Conference on Biometrics*. 2014, p. 1–7. DOI: 10.1109/BTAS.2014.6996240.
- [13] FIRC, A. *Applicability of Deepfakes in the Field of Cyber Security*. Brno,CZ, 2021. Diplomová práce. Vysoké učení technické v Brně,Fakulta informačních technologií. Available at: <https://www.fit.vut.cz/study/thesis/23761/>.
- [14] FIRC, A. and MALINKA, K. The dawn of a text-dependent society: deepfakes as a threat to speech verification systems. In: *SAC '22: Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*. Association for Computing Machinery, 2022, p. 1646–1655. DOI: 10.1145/3477314.3507013. Available at: <https://www.fit.vut.cz/research/publication/12595>.
- [15] GOOGLE. *Components* [<https://material.angular.io/components/categories>]. [cit. 2023-05-08].
- [16] GOOGLE. *Introduction to the Angular docs* [<https://angular.io/docs>]. [cit. 2023-05-08].
- [17] GOOGLE. *What is container orchestration?* [<https://cloud.google.com/discover/what-is-container-orchestration>]. [cit. 2023-05-06].
- [18] GOOGLE. *What is Microservices Architecture?* [<https://cloud.google.com/learn/what-is-microservices-architecture>]. [cit. 2023-01-09].
- [19] GRAFANA LABS. *What is Grafana?* [<https://grafana.com/oss/grafana/>]. [cit. 2023-05-07].
- [20] HASAN, T. *6 Top Message Brokers for Modern Applications* [<https://geekflare.com/top-message-brokers/>]. [cit. 2023-05-07].
- [21] HERNANDEZ ORTEGA, J., TOLOSANA, R., FIERREZ, J. and MORALES, A. DeepFakes detection based on heart rate estimation: single-and multi-frame. In: *Handbook of Digital Face Manipulation and Detection: From DeepFakes to Morphing Attacks*. Springer International Publishing, 2022, p. 255–273. DOI: 10.1007/978-3-030-87664-7\_5. ISBN 978-3-030-87664-7. Available at: [https://doi.org/10.1007/978-3-030-87664-7\\_5](https://doi.org/10.1007/978-3-030-87664-7_5).
- [22] HOMELAND SECURITY. *Increasing Threat of Deepfake Identities* [[https://www.dhs.gov/sites/default/files/publications/increasing\\_threats\\_of\\_deepfake\\_identities\\_0.pdf](https://www.dhs.gov/sites/default/files/publications/increasing_threats_of_deepfake_identities_0.pdf)]. 2021 [cit. 2022-12-16].
- [23] IBSEN, M., RATHGEB, C., FISCHER, D., DROZDOWSKI, P. and BUSCH, C. Digital Face Manipulation in Biometric Systems. In: *Handbook of Digital Face Manipulation and Detection: From DeepFakes to Morphing Attacks*. Springer International Publishing, 2022, p. 27–43. DOI: 10.1007/978-3-030-87664-7\_5. ISBN 978-3-030-87664-7. Available at: [https://doi.org/10.1007/978-3-030-87664-7\\_5](https://doi.org/10.1007/978-3-030-87664-7_5).

- [24] IBSEN, M., RATHGEB, C., FISCHER, D., DROZDOWSKI, P. and BUSCH, C. An Introduction to Digital Face Manipulation. In: *Handbook of Digital Face Manipulation and Detection: From DeepFakes to Morphing Attacks*. Springer International Publishing, 2022, p. 3–26. DOI: 10.1007/978-3-030-87664-7\_5. ISBN 978-3-030-87664-7. Available at: [https://doi.org/10.1007/978-3-030-87664-7\\_5](https://doi.org/10.1007/978-3-030-87664-7_5).
- [25] ITO, K. and JOHNSON, L. *The LJ Speech Dataset* [<https://keithito.com/LJ-Speech-Dataset/>]. 2017.
- [26] KAALEL. *MVC Framework Introduction* [<https://www.geeksforgeeks.org/mvc-framework-introduction/>]. [cit. 2023-05-15].
- [27] KORSHUNOV, P. and MARCEL, S. The Threat of Deepfakes to Computer and Human Visions. In: *Handbook of Digital Face Manipulation and Detection: From DeepFakes to Morphing Attacks*. Springer International Publishing, 2022, p. 97–115. DOI: 10.1007/978-3-030-87664-7\_5. ISBN 978-3-030-87664-7. Available at: [https://doi.org/10.1007/978-3-030-87664-7\\_5](https://doi.org/10.1007/978-3-030-87664-7_5).
- [28] KUBERENTES SIGS. *Walkthrough* [<https://github.com/kubernetes-sigs/prometheus-adapter/blob/master/docs/walkthrough.md>]. [cit. 2023-05-12].
- [29] MAJUMDAR, P., AGARWAL, A., VATSA, M. and SINGH, R. Facial retouching and alteration detection. In: *Handbook of Digital Face Manipulation and Detection: From DeepFakes to Morphing Attacks*. Springer International Publishing, 2022, p. 367–387. DOI: 10.1007/978-3-030-87664-7\_5. ISBN 978-3-030-87664-7. Available at: [https://doi.org/10.1007/978-3-030-87664-7\\_5](https://doi.org/10.1007/978-3-030-87664-7_5).
- [30] MICROSOFT. *Azure Container Instances and container orchestrators* [<https://learn.microsoft.com/en-us/azure/container-instances/container-instances-orchestrator-relationship>]. [cit. 2023-05-06].
- [31] MICROSOFT. *Overview of ASP.NET Core* [<https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-7.0>]. [cit. 2023-05-07].
- [32] MICROSOFT. *TypeScript is JavaScript with syntax for types*. [<https://www.typescriptlang.org/>]. [cit. 2023-05-08].
- [33] MILLS, A. G., KAEWCHARUAY, P., SATHIRASATTAYANON, P., DUANGPUMMET, S., GALAJIT, K. et al. Replay Attack Detection Based on Voice and Non-voice Sections for Speaker Verification. In: IEEE. *2022 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*. 2022, p. 221–226.
- [34] MIRSKY, Y. and LEE, W. The Creation and Detection of Deepfakes: A Survey. *ACM Comput. Surv.* New York, NY, USA: Association for Computing Machinery. 2021, vol. 54, no. 1. DOI: 10.1145/3425780. ISSN 0360-0300. Available at: <https://doi.org/10.1145/3425780>.
- [35] MÜLLER, N. M., PIZZI, K. and WILLIAMS, J. Human perception of audio deepfakes. In: *Proceedings of the 1st International Workshop on Deepfake Detection for Audio Multimedia*. Association for Computing Machinery, 2022, p. 85–91. DOI:

10.1145/3552466.3556531. ISBN 9781450394963. Available at:  
<https://doi.org/10.1145/3552466.3556531>.

- [36] NGUYEN, H. H., YAMAGISHI, J. and ECHIZEN, I. Capsule-Forensics Networks for Deepfake Detection. In: *Handbook of Digital Face Manipulation and Detection: From DeepFakes to Morphing Attacks*. Springer International Publishing, 2022, p. 275–301. DOI: 10.1007/978-3-030-87664-7\_5. ISBN 978-3-030-87664-7. Available at: [https://doi.org/10.1007/978-3-030-87664-7\\_5](https://doi.org/10.1007/978-3-030-87664-7_5).
- [37] PETE LEPAGE, T. S. *Add a web app manifest* [<https://web.dev/add-manifest/>]. [cit. 2023-05-08].
- [38] PROMETHEUS AUTHORS. *Monitoring Linux host metrics with the Node Exporter* [<https://prometheus.io/docs/guides/node-exporter/#monitoring-linux-host-metrics-with-the-node-exporter>]. [cit. 2023-05-9].
- [39] PROMETHEUS AUTHORS. *What is Prometheus?* [<https://prometheus.io/docs/introduction/overview/>]. [cit. 2023-05-07].
- [40] PROMETHEUS OPERATOR. *Getting Started* [<https://prometheus-operator.dev/docs/user-guides/getting-started/>]. [cit. 2023-05-10].
- [41] PUBNUB. *What is HTTP Long Polling?* [<https://www.pubnub.com/blog/http-long-polling/>]. [cit. 2023-05-14].
- [42] RED HAT. *What is a REST API?* [<https://www.redhat.com/en/topics/api/what-is-a-rest-api>]. [cit. 2023-05-06].
- [43] RICHARDSON, C. *Pattern: Microservice Architecture* [<https://microservices.io/patterns/microservices.html>]. [cit. 2023-01-09].
- [44] ROY, R., JOSHI, I., DAS, A. and DANTCHEVA, A. 3D CNN Architectures and Attention Mechanisms for Deepfake Detection. In: *Handbook of Digital Face Manipulation and Detection: From DeepFakes to Morphing Attacks*. Springer International Publishing, 2022, p. 213–234. DOI: 10.1007/978-3-030-87664-7\_5. ISBN 978-3-030-87664-7. Available at: [https://doi.org/10.1007/978-3-030-87664-7\\_5](https://doi.org/10.1007/978-3-030-87664-7_5).
- [45] SCHERHAG, U., RATHGEB, C. and BUSCH, C. Face Morphing Attack Detection Methods. In: *Handbook of Digital Face Manipulation and Detection: From DeepFakes to Morphing Attacks*. Springer International Publishing, 2022, p. 331–349. DOI: 10.1007/978-3-030-87664-7\_5. ISBN 978-3-030-87664-7. Available at: [https://doi.org/10.1007/978-3-030-87664-7\\_5](https://doi.org/10.1007/978-3-030-87664-7_5).
- [46] SENSY TEAM. *How to Detect a Deepfake Online: Image Forensics and Analysis of Deepfake Videos* [<https://sensity.ai/blog/deepfake-detection/how-to-detect-a-deepfake>]. [cit. 2023-01-08].
- [47] SHEAD, S. *Elon Musk's tweets are moving markets — and some investors are worried* [<https://www.cnbc.com/2021/01/29/elon-musks-tweets-are-moving-markets.html>]. 2021 [cit. 2022-12-28].

- [48] SPREEUWERS, L., SCHILS, M., VELDHUIS, R. and KELLY, U. Practical Evaluation of Face Morphing Attack Detection Methods. In: *Handbook of Digital Face Manipulation and Detection: From DeepFakes to Morphing Attacks*. Springer International Publishing, 2022, p. 351–365. DOI: 10.1007/978-3-030-87664-7\_5. ISBN 978-3-030-87664-7. Available at: [https://doi.org/10.1007/978-3-030-87664-7\\_5](https://doi.org/10.1007/978-3-030-87664-7_5).
- [49] THE GRAPHQL FOUNDATION. *Introduction to GraphQL* [<https://graphql.org/learn/>]. [cit. 2023-05-06].
- [50] THE KUBERENTES AUTHORS. *Horizontal Pod Autoscaling* [<https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/>]. [cit. 2023-05-11].
- [51] THE KUBERENTES AUTHORS. *Objects In Kubernetes* [<https://kubernetes.io/docs/concepts/overview/working-with-objects/>]. [cit. 2023-05-12].
- [52] TONY GARNOCK JONES, G. M. R. *Introduction to Pika* [<https://pika.readthedocs.io/en/stable/>]. [cit. 2023-05-10].
- [53] VERDOLIVA, L. Media Forensics and DeepFakes: An Overview. *IEEE Journal of Selected Topics in Signal Processing*. 2020, vol. 14, no. 5, p. 910–932. DOI: 10.1109/JSTSP.2020.3002101.
- [54] VMWARE. *AMQP 0-9-1 Model Explained* [<https://www.rabbitmq.com/tutorials/amqp-concepts.html>]. [cit. 2023-05-10].
- [55] VMWARE. *Monitoring with Prometheus & Grafana* [<https://www.rabbitmq.com/prometheus.html>]. [cit. 2023-05-07].
- [56] VMWARE. *RabbitMQ Cluster Operator for Kubernetes* [<https://www.rabbitmq.com/kubernetes/operator/operator-overview.html>]. [cit. 2023-05-10].
- [57] WANG, R., JUEFEI XU, F., HUANG, Y., GUO, Q., XIE, X. et al. Deepsonar: Towards effective and robust detection of ai-synthesized fake voices. In: *Proceedings of the 28th ACM international conference on multimedia*. 2020, p. 1207–1216.

## Appendix A

# Contents of the attached media

**training** – Logs from training, and trained models for AudioDeepFakeDetection project

**results** – Test results CSVs, resources consumption graphs

**src** - Sources codes

**clients** – Browser extension implementation

**server** – Framework components and deployment files

**api** – API Endpoint implementation

**k8s** – Kuberentes deployment manifests files

**processing\_units** – Request Processing Controller implementation

**tests** - Test and graphs plotting scripts

**templates** - Templates for detection methods integration

**README.md** - Description of how to work with framework

**xberna18.pdf** - This document