Model Architectures and Algorithms for Frugal Deep Learning Applications

By

Zhongwei Teng

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

December 17, 2022

Nashville, Tennessee

To my beloved parents, who support my decisions in each stage of my life.

ACKNOWLEDGMENTS

First and foremost, I would like to express my most genuine respect and gratitude to my advisor, Prof. Jules White, for his patient instructions and generous financial support throughout my Ph.D. journey. Dr. White guided me with comprehensive research directions on numerous research projects and opportunities, providing much help in my publications.

I would like to thank my committee members, Prof. Douglas C. Schmidt, Prof. Peng Zhang, Prof. Aniruddha Gokhale, and Prof. Maria Powell, for serving on the committee of my dissertation. They have helped me with many projects and courses and offered constructive insights into my research work in the past few years. It is my great honor to have such a supportive and responsible committee. I also wish to thank Dr. Cui for his supportive mentorship during my summer internship at Meta.

I would like to express my special thanks to my parents, Mr. Maoping Teng and Mrs. Wenzhu Xu, for their unconditional love and enormous support in every stage in my life. I want to also thank my dearest friends my friends and colleagues, Quchen Fu, Carlos Olea, Michael Sandborn, Pengfei Wang, Zhuangwei Kang, Ziran Min, Shuang Zhou, Xiaosi Zhang, Ziteng Liu, Yubo Fan, Baiting Luo, and many others who have always encouraged me and backed me up during in my incredible journey.

Chapter 1

Introduction

Research works in deep learning have shown promising results in solving real-world problems in various domains with distinct human inputs, such as images[1], natural languages [2], speeches [3], and motions [4]. With novel ideas on model structure, such as ResNet [5] and Transformer [6], evaluation results on problem benchmarks keep reaching better performance. On the other side, industry attendance at international AI conferences indicates that the trending works of on-the-edge deep learning are also applied in real-world applications [7], extending from pure research to in-the-wild context. As a result, deep learning theory is usually combined with engineering experience to build the in-the-wild best practice. For example, researchers and developers often publish guidance books to help engineers improve working efficiency and better understand strategy in deep learning applications [8, 9].

As the connections between research and applications become stronger, besides significant challenges in research fields in the deep learning community, such as model performance and model explainability [10, 11], challenges related to in-the-wild tasks appear and become more and more prevalent. For example, when extending from pure research environments to applications, compared to static evaluation datasets, there are higher expectations of models' robustness towards unexpected, noisy human inputs and lower model inference latency [12]. Thus, when generalizing from research ideas to user-orientated applications, practical case-based challenges usually need to be solved to make applications work. The gaps often happen outside the research environments, such as scalability[13], reliability, efficiency[12], robustness[14], and usability[15].

Among those research-to-application gap challenges, a notable but relatively under-discussed question is the costs of designing well-performed machine learning applications [16]. In many scenarios, it is impractical to directly apply state-of-the-art models in business systems due to limitations in computational cost. For example, due to healthcare privacy policy, it is better to perform computations of machine learning models on wearable devices, such as smartwatches, and thus raises additional demands on model complexity.

On the other hand, the cost limitation also exists when applying deep learning models with novel ideas to solve specific problems, such as exploring human-computer interaction with novel inputs. In this scenario, a commonly available research dataset usually can not be directly applied to target applications, which requires developers to prepare their dataset. However, human work on labeling can be expensive in generating large-scale datasets, which are required for most deep learning models, especially when domain knowledge is necessary for labeling works. For example, to detect common language vulnerabilities with a deep learning model, the labeler must have a solid cyber-security background to build a trustful dataset. As a result, collecting a large amount of data in the initial stage of the problem is almost infeasible. Furthermore, the developer needs more data to validate model performance effectively.

Thus, the frugality of designing machine learning applications remains a challenge in both data and model complexity.

Research questions: What are frugality concerns in deep learning applications, and how do we build model architectures and algorithms to improve the frugality of deep learning applications? Frugality in deep learning is a relatively underrepresented problem in research publications and lacks comprehensive reviews. It requires a balance between better model performance and increasing cost demands. As a broad question, interpretations of frugality are case-based and thus highly depend on inputs and the context of specific deep learning applications.

In this thesis research, we will focus on different stages of optimizing deep learning applications with clear use cases and discuss their frugality challenges. As shown in figure1.1, we will discuss the frugality problem in different stages during developing deep learning

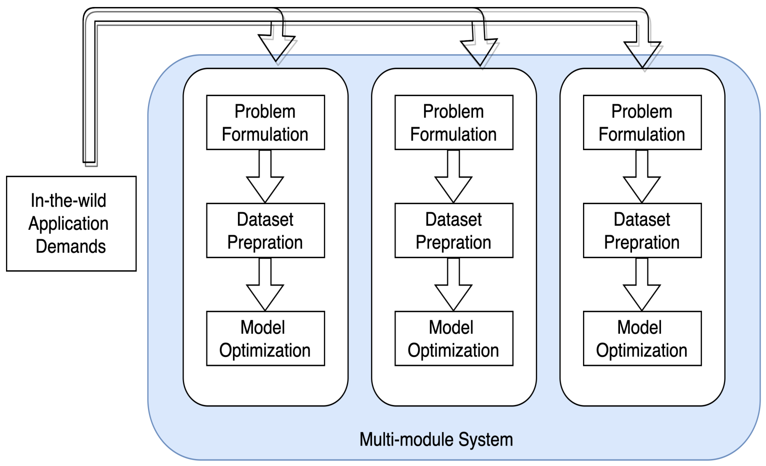


Figure 1.1: Stages of developing deep learning applications.

applications.

1. Problem Formulation and Dataset Preparation. At the beginning of development, we must formulate a real-world problem into a machine-learning problem. We often have challenges that the data needs to be more/ or we need an eligible dataset. It is straightforward to collect high-quality data which are human-labeled. At the same time, this high-cost solution also prevents validating research ideas and further improving model performance due to the scalability of human-labeled methods.

2. Better performance with minor computation cost. For the given research problem, we need to improve the model performance on the target dataset. Models with more parameters usually reach better results. Especially for in-the-wild problems, models need to be robust to handle much more noisy data. To apply more complex models, the computational costs can be a challenge in real applications. How to solve a machine learning problem with less computation cost has become more and more critical.

3. Multiple models in complex machine learning problems. A machine learning system may contain multiple independent models to solve complex problems as a pipeline structure. The final results are decided by an ensemble or cascaded ways via predic-

tions from submodels. Improving overall performance and inference efficiency with the results of all submodels is the major problem we need to focus on.

To better investigate challenges across different stages, we will discuss frugality problems based on three specific use cases, including automatic programming and speaker verification tasks. Discussion in each use case will focus on frugality concerns in a specific aspect, from data preparation to model complexity. The three use cases include:

1. Automatic programming problem.

   Drawing is a natural way for users to express their programming goals without specifying domain terminologies since hand-drawn sketches are rich in information. For instance, describing a workflow by drawing a diagram is more intuitive than explaining it verbally. By leveraging sketches as a type of input for automatic programming, we could potentially make it easier for domain experts and untrained professionals to realize systems without manually writing code.

2. The ASVSpoof problem.

   Automatic speaker verification (ASV) models have been adopted widely in acoustic applications. However, potential spoofing attacks (e.g., synthetic speech, speech reply) propose challenges to the models' robustness. Therefore, CounterMeasture(CM) systems are developed as independent models to protect ASV systems. The ASVSpoof problem focus on developing a classifier to distinguish bonafide human speech with various unseen spoof attacks.

3. The SASV problem.

   The development of ASV systems and CM systems is usually independent, and the evaluation of CM systems is based on results from fixed ASV systems. Therefore, the Spoofing Aware Speaker Verification (SASV) problem is proposed to achieve better results by jointly optimizing ASV and CM systems. The problem focuses on reaching a single score based on the outputs of ASV and CM systems to determine if a speech is from target speakers and avoid the risks of spoofing attacks simultaneously.

1.1 Understand Research Challenges

This section discusses the critical frugality challenges in developing deep learning applications, which arise in different stages of development. The rapid development of deep learning research benefits from the volume of datasets and computational resources. As discussed in the previous section, the frugality challenge is a trade-off between achieved performance and costs.

1.1.1 Challenge 1: Expensive dataset preparation in domain-specific and non-trivial tasks

Progress in popular deep learning questions, such as image recognition, inspires researchers and developers to explore new ideas to apply the state-of-the-art model in domain-specific use cases. With well-performed model architectures and algorithms, deep learning is ubiquitous. It can be integrated into almost any user-oriented application to improve efficiency, security, and reliability in real-world problems For example, general image recognition models can be used in applications determining the type of mushrooms.

However, the domain-specific questions are usually case-based, so most of the time, available resources, such as the public dataset, can not be used as the training dataset. In this case, good models exist, but the solutions do not. The application development needs dataset preparation and benchmarks for performance evaluation, where great human efforts are necessary. Moreover, these challenges can be costly if the labeling work is a task that requires domain experts to create datasets. For example, in the NLC2CMD challenge, English descriptions need to be translated into Bash commands with machine learning models. IBM contributed an initial dataset by hiring bash experts clean data collected from StackOverflow. Even though the current dataset in NLC2CMD contains 9,305 NL/Bash pairs, it is still a relatively small number for a deep-learning problem. The method used to produce the NLC2CMD dataset is too expensive to scale up because it requires manual editing by Bash experts. As a result, the performance of the NLC2CMD application is primarily affected by the data volume.

The difficulty in data preparation indicates a gap between theoretical models and domain-specific applications. Even though it is believed that an algorithm is a solution to solve a real-world problem, data limitations often prevent researchers from further validating their ideas. Thus, improving frugality in data preparation becomes a critical challenge in many domain-specific deep learning problems.

1.1.2 Challenge 2: Better model performance requires higher computational costs

. A significant metric to evaluate deep learning models is the quality of prediction results, such as accuracy or equal error rate. The promising results are usually derived from research challenges, without consideration of noisy in-the-wild environments, where model performance is expected to be more robust and stable for unseen data or even malicious attacks. Thus, we have higher demands on deep learning models towards unexpected inputs in given tasks. To achieve better model performance, researchers typically design a more complex model architecture, such as deeper networks, with higher computational costs. For example, in image recognition tasks, by adapting the ResNet architecture, layers of the convolution network are boosted to 50, even 101, and reach better performance on various tasks, improving the results of smaller networks.

Conversely to developing a single deep network, another prevalent solution is to improve model performance with hybrid models and inputs. Multiple models towards the same tasks are trained independently and final results can be calculated. This method is preferable when input data has more than one representation, such as speech-related tasks. Each independent model benefits from its input signal and thus can complement each other, hugely improving models' stability towards unseen inputs. For example, speech signals with different frequency filters focus on different parts of the raw waveform, and energy loss is inevitable during signal processing. As a result, the ensemble model can take advantage of both inputs and submodel structure characteristics.

Single deeper, or hybrid models are preferable in algorithm selections for better prediction quality. However, both solutions tend to have more training parameters, proposing potential challenges in model complexity. When the computational resources or data transmission are limited, the model complexity may become a bottleneck for better performance. For example, wearable application applications may only be allowed to use local computational resources due to healthcare privacy concerns.

1.1.3 Challenge 3: Increased complexity of systems with multiple models

Systems of deep learning applications tend to become more complex to adapt to new in-the-wild challenges. Each submodel can contribute to a specific part of the overall problem and then make final predictions. For example, at the beginning of the face recognition problem, it has less concern about synthetic face images. However, with the development of deep fake technology, face recognition systems need to add a model to detect spoofing images to solve risks caused by malicious deep fake attacks.

Robust deep learning applications rely on various submodels designed under different research contexts. In many scenarios, different components in a machine learning system take homogeneous inputs (e.g., images, speech, or natural languages) with different data distributions, so that submodels can be optimized on specific evaluation metrics. When applications need to solve a new problem, the system usually needs to supplement new models into existing pipelines. Empirical results indicate that more submodels and appropriate fusion strategies, such as embedding-based methods, can lead to better accuracy performance.

In the meantime, potential challenges in efficiency and computational costs arise due to the increased number of system components. Restraining a system to be simple can be a more challenging problem compared to building a complicated system.

1.2 Understand Use Cases

1.2.1 Overview: Human Sketch to Visualization(Sketch2Vis)

Data visualization is a vital tool that enables people to better understand the driving forces behind real-world phenomena [17]. These visualizations help provide insights by creating graphical representations of data element relationships, trends, and dominant features. Many visualization tools are available, ranging from programming-based visualization libraries (such as Matplotlib [18] and D3 [19]) to user-friendly graphical user interface (GUI) apps (such as Tableau [20]) for building visualizations interactively.

Building good data visualizations from raw data is hard. Programmers need training to use data visualization tools competently, including tools that use GUI-based interfaces [21]. A consequence of this need for training is that domain experts who need visualizations to understand physical systems (e.g., power grids, pedestrian traffic, or healthcare processes) often team up with a visualization drafter (e.g., a software engineer or data scientist) to build a data visualization collaboratively [22].

For example, consider a scenario where a physician unfamiliar with data visualization tools needs several visualizations produced to understand how data (e.g., heart rate, pulse oxygen, blood glucose data) from medical devices attached to a patient correlate with changes in the patient's blood pressure. The physician could state these requirements to the drafter and describe the format they want the medical device data presented in, which could then be used to generate the visualizations. The physician might also draw a sketch to describe roughly what they expect the final visualization to look like.

In this scenario, the sketch would provide an abstract representation or set of constraints that the final visualization of the data is expected to adhere to (e.g., a drawing of a line plot with separate series for each of heart rate, pulse oxygen, blood glucose vs. blood pressure). The drafter would take the high-level sketch of the requirements describing the type of visualization and a description of the data to use (e.g., provided by the labels of the sketch).

They could then instantiate a concrete visualization in source code and connect it to the appropriate data corresponding to labels in the sketch.

Research question: Can deep learning be used to generate visualizations from hand-drawn sketches automatically? Drawing is a natural way for domain experts to express their visualization goals since hand-drawn sketches are rich in information without specifying visualization terminologies. By leveraging sketches as a type of input for data visualization, visualization tools could potentially make it easier for domain experts and untrained professionals to explore data sets. Moreover, as a natural way to interact with mobile devices, sketches can be integrated into mobile data exploration tools [23] to help users operate visualization features easier on mobile devices. At current stage, we focus on the generation of the source code to realize the visualization-only.

Progresses in deep learning has shown great promise in solving a number of unstructured problems, such as identifying the content of images or writing essays as if they were written by a human. Deep learning provides us a potential way to improve current automatic programming approaches. For example, neural networks, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), have been adopted in various domains, such as machine translation and image recognition, due to their ability to understanding images and natural language. Deep learning advances will help make this next step in automatic programming possible by raising the level of abstraction and making the coding process be more declarative using languages that are closer to the flexibility of human languages.

1.2.2 Overview:Automatic Speaker Verification Spoofing and Countermeasures Challenge(ASVSpoof) Challenge

Automatic speaker verification (ASV) is a widely used bimetirc authentication in many acoustic-related applications, such as awaking mobile devices, or IoT applications. It can be regarded as a gate to subsequent speech recognition tasks. ASV systems provide a usable and reliable authentication methods, even for in-the-wild context, for users without extra tokens. However, compared to high performance on bonafide human speech, ASV systems are vulnerable to presentation attacks, where attackers can spoof the system by masquerading target speakers with synthetic speech or replayed speech.

A countermeasure system is developed to protect ASV systems' resistance from malicious spoofing attacks. A CM system is a binary classifier to distinguish bonafide and spoof speech, filtering both zero-effort and spoofed impostors for subsequent works.

Due to the continuous development of speech synthesis technology, the exact nature of spoofing attacks could hardly be interpretable. Generalization becomes a significant demand for countermeasures systems to process unseen spoofing attacks. However, the uncertainty of in-the-wild environments proposes a big challenge for the ASVSpoof problem when using a single classic speech signal as input features. Considering inevitable energy loss during the calculation of speech signals, ensemble models with multiple inputs always show better performance. In addition, a deep complex convolution network shows its ability to process raw waveform. Due to less energy loss, it could be more suitable for the ASVSpoof problem. Both of the solutions would primarily increase the model's computational costs.

As discussed in the previous section, when the ASV system is developed on smaller devices with limitations on computational resources, the model complexity becomes another concern.

1.2.3 Overview: spoofing aware speaker verification (SASV) Challenge

Empirical study shows that the ASV and CM systems show promising performance for in-the-wild acoustic environments. The minimum tandem detection cost function (t-DCF) is used to evaluate CM systems' impact on ASV systems when confronting spoofing attacks. However, training and evaluation of the CM system are based on fixed ASV systems. In other words, the development of the ASV system and CM are independent.

Table 1.1: Research challenges and proposed approaches in the research

Challenge	Approach	Section
Expensive Dataset preparation in domain-specific and non-trivial tasks	Automatic dataset generation with reverse captioning in Sketch2Vis problem	2.5
Better model performance requires higher computational costs	Auxiliary strategy of hybrid models in ASVSpoof problem	3.3
Increased complexity of systems with multiple models	Single SASV model based on generalization of different types of speech data	4.3

The SASV challenge extends prior research and tries to achieve better results by considering ASV and CM models as a single integrated system, bridging the gap between the ASV and CM systems. In the SASV problem, the system can deliver one single score from input speech, recognizing target speakers' information and considering the risks of spoofed utterances at the same time.

1.3 Overview of The Research

Table 1.1 shows the challenges and proposed approaches, as well as section locations in this dissertation research.

Figure 1.2 shows an overview of our research work. As shown in the figure, we will discuss frugality challenges with 3 use cases.

Contribution 1: Proposing an scalable dataset generation solution for code generation problems using reverse captioning and validating the feasibility of the solution in the Sketch2Vis Challenges.

A common challenge in automatic code generation problems is the limitation of the training dataset. Due to the requirements on domain knowledge of programming languages, each code generation application needs an entirely different training dataset. Traditionally, specific programming language experts are hired to manually translate input data (e.g., En-

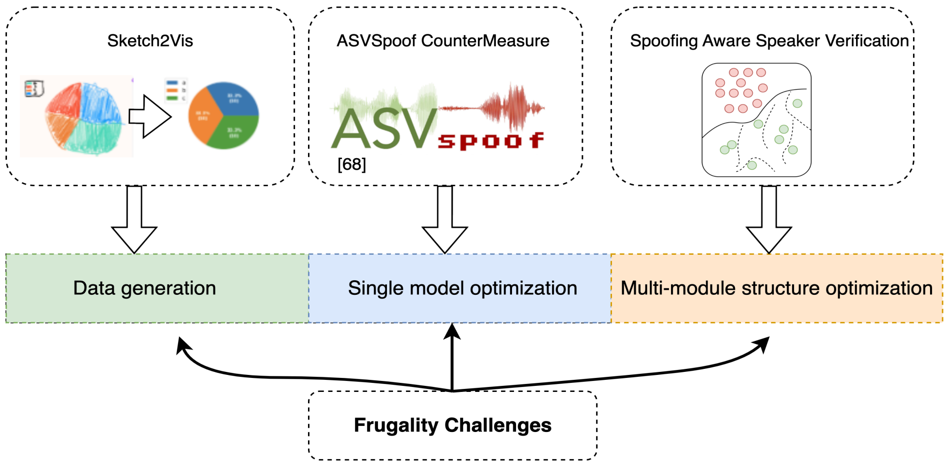

Sketch2Vis
ASVSpoof CounterMeasure
ASVspoof
[68]
Spoofing Aware Speaker Verification
Data generation
Single model optimization
Multi-module structure optimization
Frugality Challenges

Figure 1.2: Overview of the research work.

glish descriptions, image descriptions) into functional code. Thus, high-quality datasets are generated expensively, lacking the scability to enlarge data volume. It prevents researchers from effectively solving the problem in deep learning models.

To build a training dataset frugally, we proposed a reverse captioning solution for the automatic code generation problem. Labels of inputs/code pair can be generated first, and execution results of generated code will be processed with style transfer or image augmentation techniques as training features. A smaller-sized human-labeled data will be used for validating model performance.

In this dissertation research, we implement our reverse captioning solution in the Sketch2Vis problem, where a model needs to translate human sketches into visualization code. We automatically combine Domain-Specific Language (DSL) with style transfer to generate sketch-styled training data with corresponding visualization code. The proposed solution overcomes the challenge of sourcing a data set of paired hand-drawn sketches with data visualization source code. Moreover, the results show that models trained on this synthetically generated data effectively generalize to hand-drawn sketches.

We trained a transformer-based model on our generated dataset and validated model performance on the human-labeled validation set. The results demonstrate the feasibility of using transformer models with the generated dataset to reach the desired accuracy of generated visualization code.

Contribution 2: Proposing a hypothesis towards speech-related problem to leverage multiple speech signals in hybrid models frugally and validating the hypothesis with an implementation model structure, ARawNet, in the ASVSpoof challenge, improving model performance with less model complexity.

An emerging trend in audio processing is capturing low-level speech representations from raw waveforms. These representations have shown promising results on a variety of tasks, such as speech recognition and speech separation. Compared to handcrafted features, learning speech features via backpropagation can potentially provide the model greater flexibility in how it represents data for different tasks. However, results from empirical studies show that, in some tasks, such as spoof speech detection, handcrafted features still currently outperform learned features.

In this dissertation research, instead of evaluating handcrafted features and raw waveforms independently, we proposes an Auxiliary Rawnet model to complement handcrafted features with features learned from raw waveforms for spoof speech detection. A key benefit of the approach is that it can improve accuracy at a relatively low computational cost. The proposed Auxiliary Rawnet model is tested using the ASVspoof 2019 dataset and the results reaches best performance with smaller model complexity.

Results from this dataset indicate that a lightweight waveform encoder can boost the performance of handcrafted-features-based encoders for 10 types of spoof attacks, including 3 challenging attacks, in exchange for a small amount of additional computational work.

Contribution 3: Discussing solutions for the SASV problem and proposing a joint-training structure with multiple loss functions to simplify the system complexity with a single training stage.

Research in the past several years has boosted the performance of automatic speaker verification systems and countermeasure systems to deliver low Equal Error Rates (EERs) on each system. However, research on joint optimization of both systems is still limited. The SASV 2022 challenge was proposed to encourage the development of integrated spoofing aware speaker verification system (SASV) with new metrics to evaluate joint model performance.

To improve system frugality in SASV problem, we propose an ensemble-free end-to-end solution, SA-SASV, to build a SASV system with multi-task classifiers, which are optimized by multiple losses and has more flexible requirements in training set. The proposed system is trained on the ASVSpoof 2019 LA dataset, a spoof verification dataset with small number of bonafide speakers, and improves the performance of baseline systems from 8.76% to 4.86% in SASV-EER. Results of SV-EER and SPF-EER indicates that, the model performance can be further improved by training in complete ASV and CM dataset

Dissertation Outline. The remainder of the dissertation is organized as follows: Chapter 2 proposed a reverse-captioning solution to generate data efficiently in the Sketch2Vis problem. Chapter 3 presents a hypothesis in speech-oriented hybrid deep learning models and proposed a hybrid model called ARawNet in ASVSpoof challenges. Chapter 4 discuss the feasibility of joint training ASV and ASVSpoof model with multiple decoders to simplify model complexity. Concluding remarks and future works are discussed in Chapter 5.

Chapter 2

Case study: Generating Data Visualizations from Hand-drawn Sketches with Deep Learning

This chapter is adapted from ”Sketch2Vis: Generating Data Visualizations from Hand-drawn Sketches with Deep Learning.” published in 2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA) and has been reproduced with the permission of the publisher and my co-authors Quchen Fu, Jules White, and Douglas C. Schmidt.

- Teng, Zhongwei, Quchen Fu, Jules White, and Douglas C. Schmidt. ”Sketch2Vis: Generating Data Visualizations from Hand-drawn Sketches with Deep Learning.” In 2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA), pp. 853-858. IEEE, 2021.

2.1 Problem Overview

Data visualization is a vital tool that enables people to better understand the driving forces behind real-world phenomena [17]. These visualizations help provide insights by creating graphical representations of data element relationships, trends, and dominant features. Many visualization tools are available, ranging from programming-based visualization libraries (such as Matplotlib [18] and D3 [19]) to user-friendly graphical user interface (GUI) apps (such as Tableau [20]) for building visualizations interactively.

Building good data visualizations from raw data is hard. Programmers need training to use data visualization tools competently, including tools that use GUI-based interfaces [21]. A consequence of this need for training is that domain experts who need visualizations to understand physical systems (e.g., power grids, pedestrian traffic, or healthcare processes)

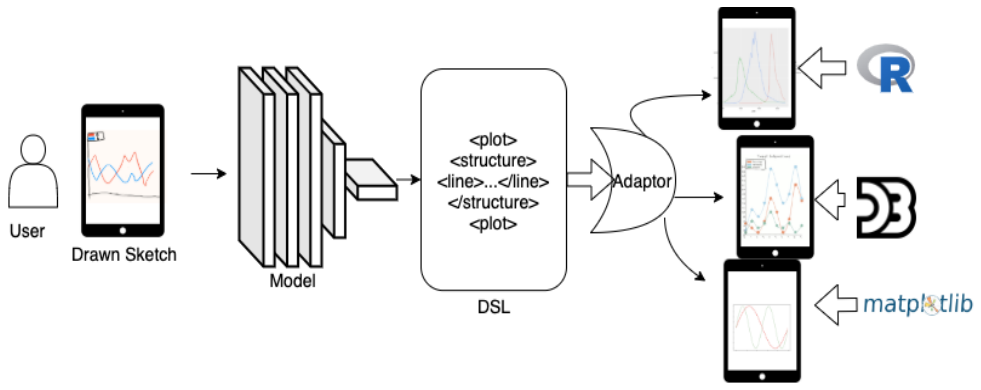


Figure 2.1: A Scenario of the Sketch2Vis Problem in Mobile Devices.

often team up with a visualization drafter (e.g., a software engineer or data scientist) to build a data visualization collaboratively [22].

For example, consider a scenario where a physician unfamiliar with data visualization tools needs several visualizations produced to understand how data (e.g., heart rate, pulse oxygen, blood glucose data) from medical devices attached to a patient correlate with changes in the patient's blood pressure. The physician could state these requirements to the drafter and describe the format they want the medical device data presented in, which could then be used to generate the visualizations. The physician might also draw a sketch to describe roughly what they expect the final visualization to look like.

In this scenario, the sketch would provide an abstract representation or set of constraints that the final visualization of the data is expected to adhere to (e.g., a drawing of a line plot with separate series for each of heart rate, pulse oxygen, blood glucose vs. blood pressure). The drafter would take the high-level sketch of the requirements describing the type of visualization and a description of the data to use (e.g., provided by the labels of the sketch). They could then instantiate a concrete visualization in source code and connect it to the appropriate data corresponding to labels in the sketch.

Research question: Can deep learning be used to generate visualizations from hand-drawn sketches automatically? Drawing is a natural way for domain experts to express their visualization goals since hand-drawn sketches are rich in information without specifying visualization terminologies. By leveraging sketches as a type of input for data visualization, visualization tools could potentially make it easier for domain experts and untrained professionals to explore data sets. Moreover, as a natural way to interact with mobile devices, sketches can be integrated into mobile data exploration tools [23] to help users operate visualization features easier on mobile devices.

This dissertation presents "Sketch2Vis,“ which is the first deep learning solution for translating human sketches into data visualization source code. Figure 2.1 shows a scenario of applying the Sketch2Vis problem on mobile apps. As shown in this figure, users can draw sketches on a mobile device without specifying visualization implementation details and a deep learning model can then generate source code for the desired visualization in multiple target visualization libraries. We focus on the generation of the source code to realize the visualization-only at this point.

The results of our research shows the capability of deep learning networks in generating multi-platform visualization code from hand-drawn sketches via a domain-specific language. The key research contributions of this dissertation include:

- Showing how data visualization code generated from a hand-drawn sketch can be modeled as an image captioning problem and test our method with three baseline models. Our empirical results demonstrate that Transformer models can achieve 95% structural accuracy in correct source code generation for hand-drawn sketches.

- Describing a novel approach for using style transfer and automated code generation to generate visualization sketches that look like they were drawn by a human and have accompanying source code as labels. Empirical results show that models trained on this synthetically generated sketch and source code labeled data generalize to real hand-drawn sketches by humans.

- We propose new evaluation metrics to score generated visualizations that are tailored to the domain of generating data visualizations from sketches. These metrics overcome challenges (such as classification accuracy) encountered using conventional metrics.

- We compare and evaluate the performance of recurrent neural network (RNN)-based and Transformer-based networks on the Sketch2Vis problem. Our empirical results provide insights into what model architectures and domain-specific language (DSL) designs perform best on the Sketch2Vis problem.

This chapter focuses only on the generation of source code to realize the visualization (e.g., render a line plot with the correct number of series, colors, etc.). Generating complex queries from natural language to select data for the plot is a separate and rich domain of research [24, 25, 26, 27]. Although combining these approaches could fully automate the selection and visualization of data, this chapter assumes a manual process for selecting data and automatically generating source code that renders the data visualization with the input data.

The remainder of this dissertation is organized as follows: Section 2.2 discusses key challenges of building a model to translate hand-drawn sketches into multi-platform visualization code; Section 2.3 analyzes the feasibility of the Sketch2Vis problem in terms of dataset construction; Section 2.4 discusses different approaches for evaluating deep learning model performance on the Sketch2Vis problem and proposes new evaluation metrics; Section 2.5 analyzes empirical results from experiments we conducted on two key deep learning architectures: RNN and Transformer models; Section 2.6 compares related work with the techniques we explore in the dissertation; and Section 2.7 presents concluding remarks and lessons learned.

2.2 Research Challenges

Deep learning networks, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), have shown great promise in understanding images and natural language. An interesting question is whether the performance they exhibit in other domains translates to the Sketch2Vis problem. This section discusses challenges specific to this problem that a deep learning model must address.

2.2.1 Challenges 1: Source Code is Platform-specific

A deep learning model should ideally be able to generate sketches using a variety of visualization libraries. However, source code for visualization libraries is typically platform-specific (e.g., Java vs. JavaScript, D3 vs. Tableau). To convert directly from sketch to data visualization, therefore, the machine learning (ML) model must be trained how to produce visualizations using the underlying visualization library to achieve the desired goal(s), i.e., the ML model must learn the syntax, grammar, and semantics of the underlying visualization tool. For example, if D3.js is used as the target visualization platform, the ML model needs to understand how to produce valid JavaScript code that will achieve a visualization corresponding to a sketch.

Although two data visualizations may appear similar to a human, if different programming languages or libraries are used to create them, the actual implementations can vary greatly, depending on the underlying platforms and programming languages. For example, the axis and plot type in a line graph are explicit to a human, regardless of whether or not a person has data visualization knowledge. However, different plotting libraries specify axes via different mechanisms, such as JavaScript arrays vs. Python lists. Users therefore need to produce separate implementations for the same visual representation each time a visualization is instantiated on a different platform and/or with a different programming language.

To train a deep learning model, a dataset is needed that associates (i.e., "pairs") images of sketches with the concrete source code to realize the appropriate visualization for the sketch. If the dataset is platform-specific (e.g., the source code targets a particular library), the model must be retrained for each individual target visualization platform. Moreover, variations in the target languages and frameworks may make this training process easier or harder.

2.2.2 Challenges 2: Sketches and Paired Source Code are Expensive to Obtain

Deep learning models require large volumes of data to increase their robustness and generalize effectively to as-yet unseen problems. Public datasets for most image processing problems typically contain over 100,000 training images and labels. For example, ImageNet [28] has 1,281,167 images with 21,841 labels and the Open Images Dataset [29] has 9,011,219 images with more than 5,000 labels.

To train a deep learning model to turn sketches into source code, a dataset that pairs hand-drawn sketches with the source code to instantiate the appropriate visualization is needed. There is no available dataset containing both hand-drawn visualizations and the corresponding source code representation. It is therefore hard to train and experiment with these models since producing a large dataset of paired sketches and source code requires working with data visualization experts, which is prohibitively expensive relative to other domains, such as labeling what is in an image (e.g., cats, dogs, flowers, etc.).

For example, 10,000 sketches and associated source code would be needed to train models on the low-end of data volume scale. Obtaining these types of datasets is not easily crowd-sourced. Moreover, even if the dataset *is* somehow crowd-sourced, ensuring that the source code labels are correct is a much harder problem than simply determining if a bicycle or street light is in a picture using the mechanisms (such as captchas) applied in related work.

2.2.3 Challenges 3: Correlations Among Human-drawn Visualizations

There is large variation in how a human may sketch a particular visualization. Each type of visualization must therefore be drawn numerous times in the training set for models to learn. For example, if we want a 20,000-image dataset supporting five visualization types, 4,000 images for each type of visualizations should be drawn repeatedly.

In contrast, human drawn sketches are affected by personal style, e.g., sketches drawn by same person are often similar, especially when the target visualizations are simple. In this case, even if the goals of data volume can be achieved, getting variation in drawing style still requires a very large number of human drawers. A quality dataset must, therefore, provide as much randomness as possible in the sample sketches.

2.2.4 Challenges 4: Evaluation of Generated Code

Deep learning models rely on automated scoring of their outputs to learn and improve throughout the training process. It is hard, however, to score data visualization quality automatically. The performance of generated natural language captions, which is the type of model we used to attack the problem, is often evaluated by n-gram matching metrics [30], such as Bleu [31] and METEOR [32]. Unfortunately, evaluating the predicted code snippets for a data visualization (which are the outputs of a deep learning model) focus on low-level details that may map poorly to visualization features.

In particular, the importance of different visualization features must be considered to enhance the usability of visualization implementations. For example, in a visualization code block, legend selections may contribute more than color selections to the validity of the visualization. Therefore, although many prior scoring mechanisms have been applied in prior natural language deep learning work, it is not clear if they are sufficient or effective on the Sketch2Vis problem.

2.3 Generating Data Visualization Source Code from Hand-drawn Sketches

Image captioning is the process of producing a textual representation of an image (e.g., a car parked in front of a copse of trees). In recent work, a number of problems have been phrased as image captioning, such as the generation of HTML for a sketch of a web page layout [33]. This section describes how we investigated the feasibility of phrasing the Sketch2Vis problem as an image captioning problem. It also explores a key challenge of obtaining data and explains how we overcame this challenge by developing a novel approach using a *domain-specific language* (DSL) and *style transfer*, which is a computational process that makes an image appear as if it was produced by a human artist rather than a computer.

2.3.1 Developing a Training Dataset

Preparing a large volume of hand-drawn images paired with source code to realize corresponding visualizations is hard. A dataset that pairs hand-drawn sketches with the source code to instantiate the appropriate visualization is needed to train a deep learning model to convert sketches into source code. There is no available dataset containing both hand-drawn visualizations and the corresponding source code representation. It is therefore hard to train and experiment with these models since producing a large dataset of paired sketches and source code requires working with data visualization experts, which is prohibitively expensive relative to other domains, such as labeling what is in an image (e.g., cats, dogs, flowers, etc.).

Style transfer has emerged as an active area of research. Deep learning style transfer models can take images and make them look like they were drawn in a particular human artistic style. Significant progress has been made to advance style transfer capabilities, particularly in the domain of deep learning [34]. Some visualization libraries, such as Matplotlib, have also added simple style transfer mechanisms to make charts look hand-drawn.

We propose a solution to the training data set problem that applies (1) a domain-specific language (DSL) and its grammar to generate source code for data visualizations randomly and (2) a variety of style transfer approaches to convert these computer-rendered visualizations into images that appear hand-drawn. Our dataset creation process inverts the normal dataset curation process of obtaining raw data and then labeling it with a human. Instead, the approach we employ operates as follows:

1. Generate the source code (label),
2. Execute the source code to render the visualization and export it as an image (semi-raw data), and
3. Apply style transfer and image augmentation techniques to produce sketches of the visualization that look like they were drawn by humans (raw data).

As discussed in Section 2.2.1, a key advantage of using a DSL is that it makes the training process independent of the target visualization library. Likewise, the language design can be tailored to facilitate faster learning. Compared with sketches hand-drawn by human artists, synthesized images that appear hand-drawn have several advantages, including:

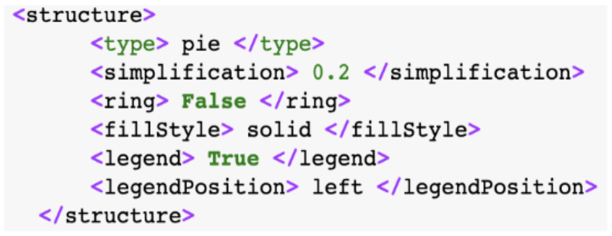
1. **Scalable rendering of highly variable sketches.** A problem with a human-driven approach is that each human tends to produce sketches with a similar style. Using hand-made sketches to train not only requires a large number of hand-drawn images, but also a large number of people to ensure the dataset shows sufficient variation to generalize to the larger population. Our DSL-based approach enables a much wider variation in visualization features/types through randomization.
2. **Balanced training data distribution.** Since the dataset is created in a semi-random way, we can manipulate the distribution of the training data. Our approach avoids potential deep learning challenges caused by imbalanced datasets, which can yield models that generalize poorly.
3. **Dramatically lower cost.** A large volume of source code labels can be prepared through source code generation from the DSL without relying on time-consuming and costly manual creation of source code for each visualization. Hiring data visual-

ization engineers to produce source code for sketches is prohibitively expensive and difficult to crowd-source.

2.3.2 From DSL to Visualization Code

As described in Section 2.2.1, a key problem in generating programming language code from sketches is that the models must be trained on each programming language that they target, which is suboptimal. To overcome this challenge—and to support realization of the sketch using multiple visualization platforms—we employ an intermediate DSL model that represents the abstract goals of the user with a simple syntax that can be learned readily by a deep learning model. The deep learning model produces code using our DSL and then code generators translate that DSL code into the implementation details of a specific visualization library.

Our DSL uses an XML-based syntax as shown in Figure 2.2. A token dictionary (called



```
<structure>
      <type> pie </type>
      <simplification> 0.2 </simplification>
      <ring> False </ring>
      <fillStyle> solid </fillStyle>
      <legend> True </legend>
      <legendPosition> left </legendPosition>
  </structure>
```

Figure 2.2: An Example of Describing a Pie Chart Instance with the DSL Model.

a token pool) is built and updated for the DSL model, as shown in Figure 2.3. Visualization characteristics are described by tokens enclosed by starting and closing tags, such as ”$< type >$” and ”$< /type >$”. These tokens are categorical values tokens, such as visualization type (e.g., scatter plot), indicating visualization feature candidates. A visualization specification is built from a sequence of tokens and the appropriate enclosing tags. The complete DSL grammar used in the experiments is available in the supplementary materials referenced at the end of the dissertation.

Images in a dataset are generated by different visualization tools. Therefore, the corresponding visualization features in our DSL may vary, i.e., features supported by some tool A may not be supported by another tool B. This syntax enables us to assign a visualization type with flexible features in our DSL (instead of a fixed length of features), such that each feature is independent. In this case, only supported features in the target visualization tool must be considered during dataset preparation.

As shown in Figure 2.3, after a DSL specification for a visualization is produced, we can apply a code generation adapter to produce the native visualization code needed to render the visualization on the target platform (e.g., matplot, D3, etc.). Adopting an intermediate DSL (rather than directly relying on native visualization code for the visualization specification) enables us to decouple the deep learning model from the generation of the source code.

Different code generators can be plugged into target different visualization platforms without retraining the underlying deep learning model. However, if a new platform has visualization capabilities that were not captured in the DSL language and trained tokens, additional training data is needed to support these new visualization features that were not trained on previously.

In summary, our DSL-based approach has the following key benefits:

- **Source code generators can produce visualizations for several platforms using a single model inference.** Since model outputs describe sketches independently of the concrete implementation, the deep learning model must only learn one syntax during the training process. A single inference from the model can be run through multiple code generators to target different platforms. The model and the source code generators can be shared and reused more easily.

- **The dataset can be updated and expanded easily.** When a training dataset must be enlarged to train more complex plots, new tags can be added to the token dictionary. After a model is trained that can interpret sketches and produce DSL instances, a code adaptor can be built for different target platforms without retraining the original models.

2.3.3 Dataset Construction and Expansion

To ensure high quality training images and reduce the possibility of overfitting, pairs of images and DSL instances in the dataset must be produced from multiple visualization tools. The goal is to ensure the rendered sketches of visualizations have different styles and wider variation that better match the variations in how humans sketch. Figure 2.3 shows these steps to initialize and extend the training dataset with new visualization tools.

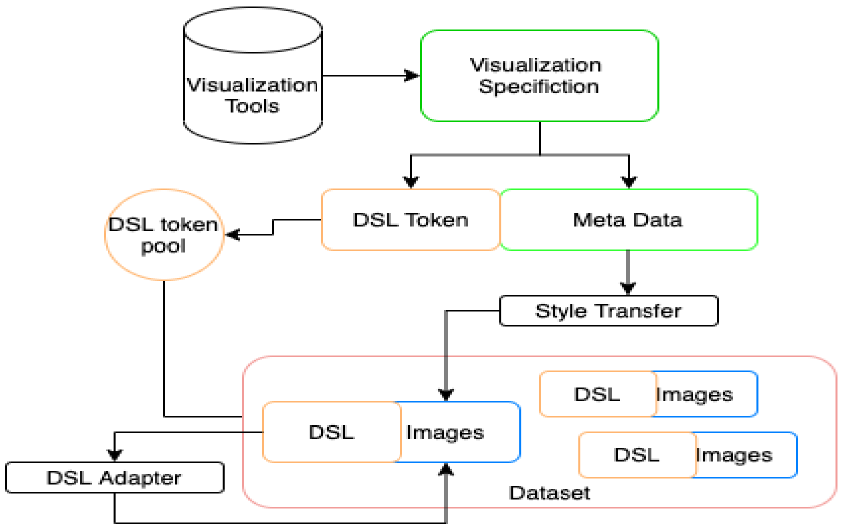


Figure 2.3: Diagram of the Dataset Construction Mechanism.

2.3.4 Adding New Visualization Capabilities to the Training Dataset

Before a new visualization tool is used to generate images for the training dataset, the supported plotting arguments for the tool must be enumerated carefully. All data used to generate the training images must be stored for future use. This data is referred to as *metadata*, which represents the implementation of the native visualization tool and influences the design of an updated DSL by creating structural tokens and allowed values for the tokens, as discussed in Section 2.3.2.

Since the new tool's visualization *metadata* is likely different from the prior visualization tools used to create the training visualizations, the expected data formats and naming convention of visualization features may also be different. Figure 2.3 shows how a DSL token pool playing a role of word vocabulary is introduced to ensure a unified format for the DSL. Instead of generating new DSL tokens directly from the *metadata*, the new features must be compared with existing tokens in the DSL token pool and mapped to equivalent tokens wherever possible to eliminate semantic duplication across tokens.

By separating the *metadata* and DSL syntax, we can ensure data differentiation without introducing excess tokens and reduce the size of the word vocabulary, which is important for the deep learning models. Unused information in the *metadata*, such as the textual name for axes, can be saved for potential future use. For example, we can train the model to recognize and label text in sketches so that the model can communicate with data directly without requiring users to input this information manually.

2.3.5 Image Corpus Differentiation

To enhance the robustness of the synthesized hand-drawn image corpus—thereby overcoming the challenge we discussed in Section 2.2.4—we applied randomization to generate a widely varying corpus of images, including the following:

1. **Visualization type**. The types of visualizations (such as line plot, bar plot, and pie plot) are generated randomly.

2. **Specifications to the visualizations**. The parameters sent to the underlying visualization libraries for the visualization type of visualization are randomized. For example, the parameters of a bar chart may include the height of the bars, color of the bar faces, color of bar edges, and legend position.

3. **Selections of the style transfer**. Style transfer approaches often have range of arguments that impact rendering style. We achieve this capability by generating images

from (1) multiple visualization tools, which support style transfer functions, and (2) sketch style transfer deep learning models, which are trained based on real hand-drawn sketches of various instances.

4. **Text differentiation**. To ensure randomness in the textual elements in images, we randomize the labels applied to visualization elements.

5. **Multiple visualization instances**. A single sketch may contain multiple visualizations, so we randomized how different visualizations were shown in a sketch.

Using our randomization approach, a wide variety of sketches can be generated that appear hand-drawn and cover a large number of visualizations. Samples of our generated sketches are shown in Figure 2.4. Figure 2.4(a) depicts four visualization types generated

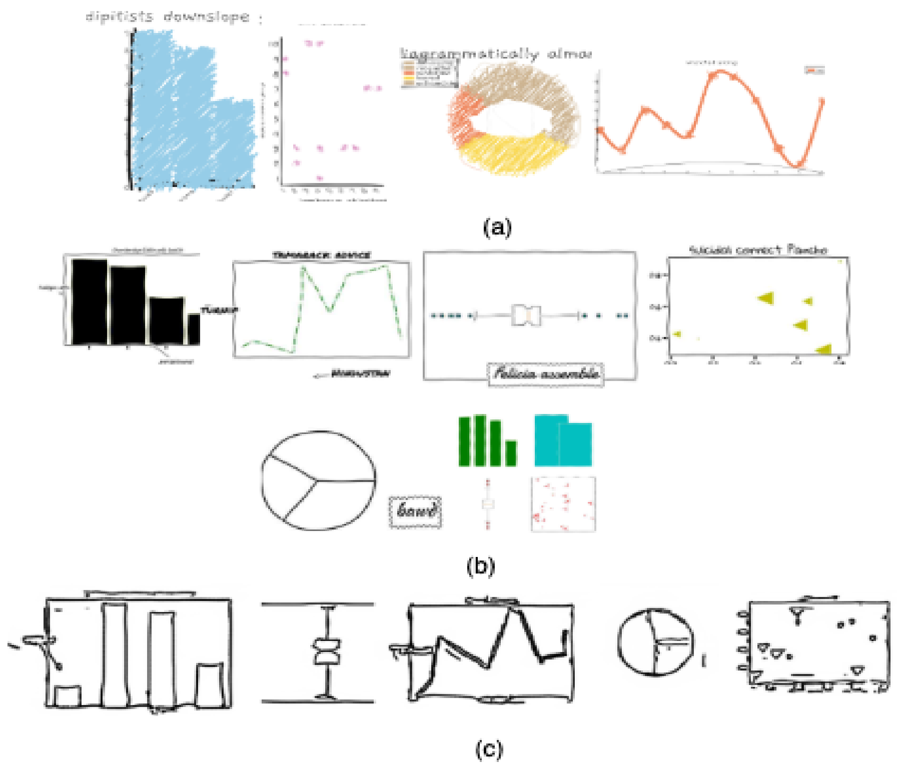


Figure 2.4: Examples of Synthetically Generated Training Images Using Style Transfer

via D3.js/RoughViz [35] on web browsers . Figure 2.4(b) shows five single visualization types and a visualization instance containing multiple visualization type generated via

$XKCD()$ function supported by Matplotlib [36]. Figure 2.4(c) shows sketches transferred by *Photo-Sketching* [34] based on images generated by visualization tools.

2.4 Sketch2Vis Evaluation Metrics Exploration

A key question explored by our research was what deep learning architectures were most promising for the Sketch2Vis problem. RNN [37, 38, 39] and Transformer [40, 41] models have each shown excellent results on image captioning problems. We experimented with a number of RNN and Transformer architectures and present the best performing architectures that we found and their accompanying results on the Sketch2Vis problem.

One challenge of our architectural analysis was determining an appropriate scoring metric to use. We initially looked to establish metrics from the domain of machine translation since we were translating images into text. In translation problems, such as English to German translation, n-gram matching metrics[30] that split a string into n-length substrings have been widely used. For example, the Bleu score [31] is widely used in evaluation of translated sentences.

In contrast to natural language translations, n-gram matching is not well-suited to evaluate the generated DSL code since there are natural differences in contribution to code quality among different tokens. N-gram matching focuses on semantic similarities between the ground truth translation and the generated translation. In contrast, when evaluating the generated DSL code, execution must be considered since the DSL code must execute correctly *and* generate the desired visualization. For example, generated DSL code snippets that execute without run time errors should be considered more accurate than DSL code snippets with higher similarity scores that cannot be executed.

Previous work on natural language to code generation with deep learning considered both the output code blocks and execution output as metrics in the evaluation [27]. For example, prior work on GUI code generation [33] that used GUI screenshots as inputs evaluated models based on classification errors. That prior work evaluated a model's ability to correctly classify the GUI components in images.

The performance of natural language to SQL models has also been evaluated by the accuracy of the executed query results [27]. That approach reflected the model's ability to retrieve the correct data since even the generated SQL queries may be different from the ground truth.

Our approach presented in this dissertation evaluates models in the Sketch2Vis problem with one accuracy metric (Acc_{cls}) from prior research and two additional metrics (Acc_{str} and Acc_{dec}) that we devised to overcome gaps observed when scoring generated visualizations. Each of these three metrics are described below:

1. **Classification accuracy.** Acc_{cls} considers the results as a classification problem, calculating differences between model output and ground truth DSL code. Outputs are penalized for producing tokens differing from the ground truth and token sequences differing in length from the ground truth. Acc_{cls} can be calculated with Equation 2.1, where T_{false} is false predicted token and $\Delta(Len_{dsl})$ is the difference in length between the predicted DSL sequence and the ground truth DSL sequence.

$$Acc_{cls} = 1 - \frac{\sum(T_{false}) + \Delta(Len_{dsl})}{Len_{dsl}} \tag{2.1}$$

2. **Structural accuracy.** Acc_{str} evaluates DSL mistakes that can result in structural errors, including syntax errors and incorrect visualization types. For example, a model that predicts the wrong visualization type or gives an invalid feature token to a certain visualization type will receive a low structural accuracy score, no matter how accurate the predicted DSL code is in other areas. Acc_{str} can be calculated with Equation2.2;

$$Acc_{str} = \begin{cases} 0, & \text{if wrong semantic/plotting type} \\ 1, & \text{otherwise} \end{cases} \tag{2.2}$$

3. **Decoration accuracy.** Acc_{dec} scores the model's ability to understand local visualization features in the images, such as the desired color of lines or positions of legends, i.e., we want to evaluate if the model can choose the correct decoration tokens after generating the DSL structure. Acc_{dec} can be calculated with Equation2.3, where $\hat{T}_{dec}$ is a false predicted decoration token.

$$Acc_{dec} = \frac{\sum \hat{T}_{dec}}{\sum T_{dec}} \tag{2.3}$$

2.5 Experimental Results

The dataset used in our experiments with RNN and Transformer architectures was constructed from the three sources described in Table 2.1. We validated these tools with a smaller number of hand-drawn sketches. Table 2.1 shows the visualization features and augmentation methods we applied to generate the dataset.

We used style transfer mechanisms to create images that looked hand-drawn via (1) visualization tools (such as *Matplotlib* [36] and *RougViz.js* [35]) and (2) *Photo-Sketching* [34], which is a style transfer deep learning model. Likewise, we used Matplotlib to generate Line, Bar, Box, Scatter and Pie plots and *RougViz.js* to generate Line, Bar and Pie plots, as shown in Table 2.1. For simple style transfer approaches, we employed the "sketch-style" function "xkcd()" and for Javascript visualizations we used RoughViz.js's human-styling capabilities. We also employed Deep learning style transfer with *Photo-Sketching* on source images generated by Matplotlib. We adopted *Photo-Sketching* [34] in our dataset to generate simple monochromatic sketches, which is a different style compared to the visualizations generated with Matplotlib, shown in Figure 2.4. This dataset can be extended by adding more plotting types and parameters.

Visualization Tools	Plots	#Meta Parameters	#DSL Tokens
Matplotlib	Line	4	51
	Bar	7	28
	Box	5	9
	Scatter	3	46
	Pie	6	21
RoughViz+React	Pie	8	28
	Line	9	20
	Bar	9	22
	Scatter	8	15
Photo-Sketching	Line	\	10
	Bar	\	10
	Box	\	10
	Scatter	\	6
	Pie	\	10

Table 2.1: Sketch2Vis Dataset Single Plot Parameter Description

2.5.1 Dataset Statistics

The dataset used in our experiments with RNN and Transformer architectures was constructed from two visualization tools described in Section 2.3.3. We validated these tools with a smaller number of hand-drawn sketches. Table 2.1 shows the visualization features and augmentation methods we used to generate the dataset.

We used style transfer mechanisms to create images that looked hand-drawn via (1) visualization tools (such as *Matplotlib* [36] and *RougViz.js* [35]) and (2) *Photo-Sketching* [34], which is a style transfer deep learning model. We used Matplotlib to generate Line, Bar, Box, Scatter and Pie plots and *RougViz.js* to generate Line, Bar and Pie plots, as shown in Table 2.1. Deep learning style transfer was performed with *Photo-Sketching* on sources images generated by Matplotlib.

Matplotlib is a popular Python 2D visualization library that generates quality visualizations [36]. The project is widely used and the source code repository on GitHub has 10,800 watchers and 55,800 questions on StackOverflow. Matplotlib includes a ”sketch-style” function, *xkcd*(), which can convert any Matplotlib graph into a hand-drawn format.

We used the *xkcd*() function to generate hand-drawn style plots randomly using Python. For browser-based visualizations, we used *RougViz.js* to generate hand-drawn style visualizations with *D*3.*js*.

Unlike style transfer functions provided by visualization tools, deep learning mechanisms use a dataset containing hand-drawn sketches [42, 34] to learn to transfer a scene into a sketch. These models capture complex variations in how humans draw and can reproduce images that mimic these variations. We adopted *Photo-Sketching* [34] in our dataset to generate simple monochromatic sketches, which is a different style compared to the visualizations generated with Matplotlib.

Both source visualization tools (Matplotlib and *RoughViz.js* in our experiments) share the same mechanism for producing the randomized *metadata*. The number of meta parameters and tokens is determined by the variability in the underlying visualization feature. For example, compared to color-related DSL tokens that can have a wide range of values, a DSL token indicating if a Bar chart is horizontal has only two possible values: ”True” and ”False.”

Photo-Sketching focuses on preserving the contours of images. Deep learning transfer may lose some visual information when applied to a visualization. For example, colors or data point styles in the original visualization will be omitted by the Photo-Sketching style transfer model and only the core features, such as the visualization type are retained. Therefore, the size of the DSL token pool will be reduced correspondingly compared to sketches generated by visualization tools. In particular, the information in *metadata* may not be rendered in generated sketches.

As a result, our current dataset contains both *images with individually colored data series* and rich visualization information and *scrawled black and white sketches* with basic visualization features. Moreover, to help avoid overfitting during training, we also include images containing multiple visualization instances, which helped our models generalize better.

Table 2.2 shows the number of tokens in the current DSL files and their distribution, which corresponds to the usage of different visualization features in the training set. As

#Token	#T_{dec}	#Avg T_{dec}	Avg DSL length
1718138	438460	6	22

Table 2.2: Sketch2Vis Dataset Description

discussed in Section 2.4, we categorized our DSL tokens into structural tokens(T_{str}) and decoration tokens(T_{dec}). Table 2.2 shows there are 438,460 T_{dec} in the dataset, which comprises 25.5% of all tokens. On average, every image contains 6 T_{dec} and 16 T_{str}, which indicates that Acc_{cls} will more likely be affected by T_{str}.

Figure 2.5 shows the distribution of detailed DSL tokens based on token types. This

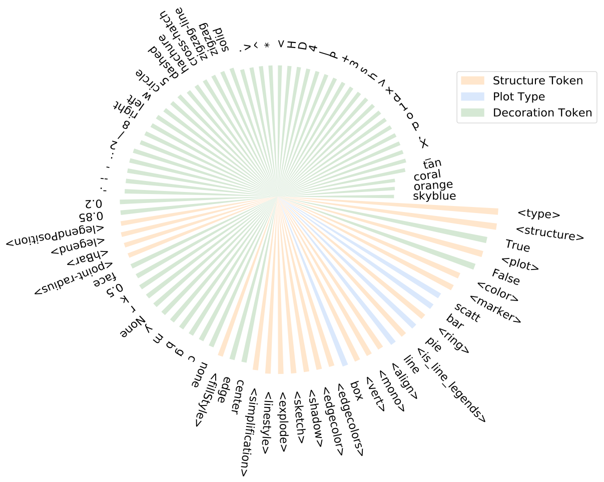


Figure 2.5: Distributions of DSL Tokens in the Sketch2Vis Dataset

dataset has balanced decoration tokens, stemming from the use of synthetic data generation, The dataset was split into training, validation, and test sets. The number of samples in each of these split datasets is shown in Table 2.3.

#All	#Train	#Valid	#Test
76942	63945	7105	5892

Table 2.3: Sketch2Vis Dataset Split Description

2.5.2 Deep Learning Architectures Explored

Sketch2Vis uses an encoder-decoder architecture, where the encoder extracts image features from input sketches via a feature extractor, which is often a Convolutional Neural Network (CNN). Image features may then be processed by a Recurrent Neural Network (RNN). The decoder portion is next applied to output sequences of DSL tokens.

Baseline model construction in our experiments was based on different pairings of encoders and decoders. We constructed and compared three baseline models based on two primary methods of processing sequenced data [43, 6, 44, 45], RNNs and Transformers, as shown in Figure **??** and described below.

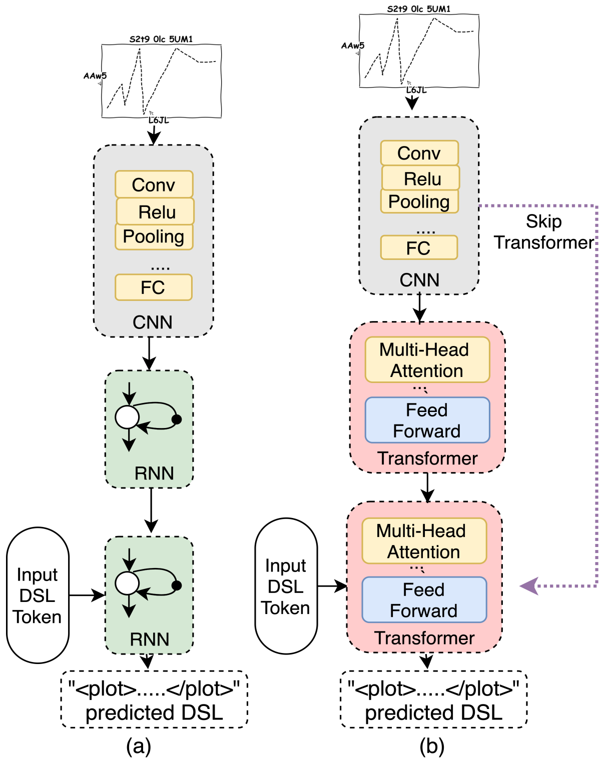


Figure 2.6: Sketch2Vis Baseline Model Construction and Comparison

2.5.2.1 RNN Model Construction

Figure **??**(a) shows the RNN model construction process [46]. At each timestamp in the encoder portion of an RNN, the output from the image feature extractor is split by its row and fed into recurrent units recursively. A token at time t, T_t, is predicted based on previous tokens, $T_{t-i}, T_{t-i+1}, \ldots T_{t-1}$, where i is the number of previous tokens in the DSL grammar. as shown in the following equation:

$$T_t = Output(T_{t-i}, T_{t-i+1}, ...T_{t-1} | F_{image}) \tag{2.4}$$

This model keeps predicting the next token in the sequence until it reaches a terminating operator.

2.5.2.2 Transformer Model Construction

Figure **??**(b) shows the Transformer model construction process [47]. Encoded inputs are used to calculate a self-attention matrix to combine with attentions from output sequences. This approach allows models to generate outputs with context.

It is not clear in the literature whether it is effective to use a Transformer to process the output of a CNN applied to a sketch image, which is not a sequenced input. In particular, prior research has not validated whether self-attention mechanisms in Transformer encoder should be applied in features extracted by a CNN. To address this question we next compare the performance of model architectures, as shown in Figure **??**.

2.5.3 Model Architecture Comparison Results

Model construction. Table 2.4 shows empirical results from baseline models we built using different selections of encoders and decoders on evaluation dataset of single plot. In general, the CNN Encoder and Transformer Decoder models show promising results in

Table 2.4: Evaluation Results of Baseline Models on Single-plot

Encoder	**Decoder**	**Acc_{cls}**	**Acc_{str}**	**Acc_{d}**
CNN/RNN	RNN	0.94	**0.98**	0.75
CNN/Linear	Transformer	**0.95**	0.97	**0.78**
CNN/Transformer	Transformer	0.77	0.74	0.49

generating visualization DSL code with the current dataset. Inaccurately predicted visualization instances were caused primarily by challenges in processing visualization features that comprised a smaller amount of visual information in the sketches (*i.e.*, a smaller number of pixels), as well as visualization features with more options.

The Transformer architecture generally showed better performance than RNN architectures in the field of natural language processing [6, 48]. By comparing RNN- and Transformer-based models in our test set, however, we found the RNN-based models performed similarly on the Sketch2Vis. In particular, both models reached a score of $\sim$0.97 on structural accuracy, which means $\sim$97% of predictions are legal DSL expressions and the correct visualization type, i.e., **$\sim$97% of the predictions produced valid executable code that generated the correct visualization.** Decoration accuracy reaches 0.78, which means 78% of visualization decoration features are legal decorations for the visualization type and are predicted correctly.

Moreover, inspired by results in image captioning problems, we also adopted the Transformer as a part of an encoder to process output features of the CNN, as shown in Figure **??**. However, our results show that image features encoded by a Transformer can hurt overall performance in the Sketch2Vis problem.

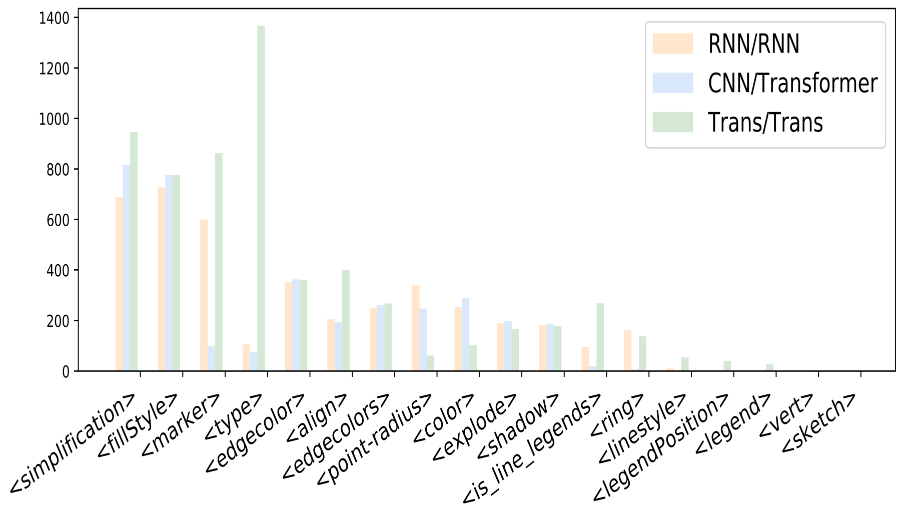


Figure 2.7: Distribution of Mispredicted Decoration Tokens

Error Analysis. Figure 2.7 shows the distribution of the incorrectly predicted visualization features in 3 models.

Errors in other visualization features are distributed more evenly and vary based on the size of the visualization feature's token vocabulary. The smaller a visualization feature's token vocabulary, the fewer errors the models make. For example, it is harder to predict the correct $< marker >$, which is the style of marker used for the data points in the visualization.

In contrast, $< is\_line\_legends >$ indicates if legends are visible in the visualization and performs much better since legends provide more information (e.g., cover a larger percentage of pixels) than the style of dots used for points. A trade-off exists between the model's usability (wider visualization capabilities in terms of the types of visualizations supported) and the model's accuracy. This trade-off shows the need to refine the DSL token vocabulary during dataset construction to only use tokens absolutely required for the needed visualizations.

The results in Figure 2.7 also indicate that we can combine the strengths of RNN and Transformer-based models into an ensemble to better support a wide range of visualization features and improve model performance. For example, a correcting mechanism can be appended to the model prediction stage, that weights both model's decisions.

Table 2.5 shows examples of the inputs and outputs in our experiments and the user-desired visualization. The "Training Sketch" shows an input created from *Rough.Viz* for encoder-decoder model training. The "Validation Sketch" shows a real hand-drawn sketch, where concrete text information is omitted. The "Generated DSL" is the predicted DSL code, which describes the visualization features, but is independent of the dataset specification (the data mapped to each axis, series, etc. can be plugged in separately). The "Generated Visualization" is a concrete visualization generated from our DSL using Matplotlib.

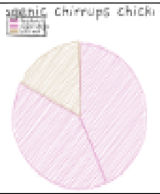
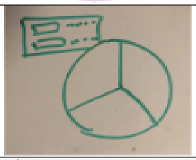
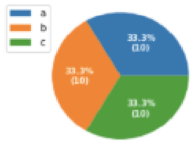
Training Sketch	
Validation Sketch	
Generated DSL	`<structure> <type> pie </type> <simplification> 0.2 </simplification> <ring> False </ring> <fillStyle> solid </fillStyle> <legend> True </legend> <legendPosition> left </legendPosition> </structure>`
Generated Visualization	

Table 2.5: Examples of Model Input and Output

2.5.4 Results Discussion and Performance on Hand-drawn Sketches

In addition to evaluating on the generated test set using a combination of style transfer approaches in Section 2.5.3, as discussed in Section 2.5.1, we also prepared 100 hand-drawn visualizations drawn on mobile devices for validation purposes. Each visualization was manually labeled with its matching DSL code. Examples of hand-drawn sketches are shown in Figure 2.8, where half of the sketches are scrawled and the other half are colored.

For images generated with style transfer mechanisms each sketch has an exact DSL description already determined. In contrast, sketches drawn by humans are hard to evaluate with Acc_{dec} since there multiple valid DSL source code implementations may exist. In this case, evaluation on Acc_{dec} relies on user feedback. We currently only test Acc_{str} on real hand-drawn sketch sets for validation purposes.

We include multi-visualization sketches in the training set. The model may therefore predict DSL code containing multiple visualization instances, even if there is only one

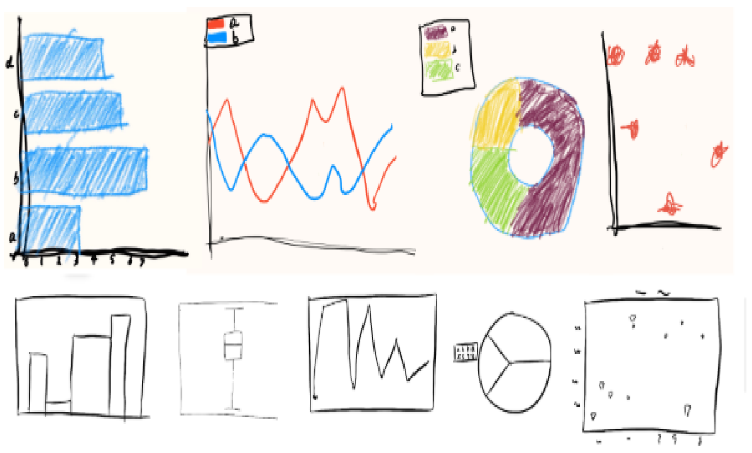


Figure 2.8: Examples of Hand-drawn Sketches from Humans in the Validation Set

visualization instance in the image. If multiple visualizations are generated, users can select a visualization instance from the predicted options. For scoring purposes, when evaluating the hand-drawn sketches, any predictions containing the correct visualization DSL code are considered as correct in the calculation of Acc_{str}.

Table 2.6 shows the results of Acc_{str} from the different models.

Encoder	**Decoder**	**Acc$_{\mathbf{str}}$**
CNN/RNN	RNN	0.66
CNN/Linear	Transformer	**0.95**

Table 2.6: Evaluation Results on Hand-drawn Images

This table shows how the Transformer-based model exhibits the best performance in processing real hand-drawn sketches, achieving 95% accuracy in identifying the visualization and correctly generating the correct code to implement the visualization. Although the RNN-based model performed well on the test set, it does not generalize well to hand-drawn sketches. This result indicates that the Transformer-based model is better at generalizing from style transfer images to real hand-drawn images.

An important result from our analysis is that the Acc_{str} is similar for both images generated using a style transfer approach and the hand-drawn images. This resulting accuracy indicates that generating the training images via style transfer is an effective approach for overcoming the challenge of obtaining human-labeled sketches. Since Transformer models were able to generalize to hand-drawn data using synthetic style transfer data, future work can be simplified significantly by relying on a much larger corpus of synthetically generated data for training and smaller sets of human-labeled data for validation. Although we could not automatically evaluate the decoration accuracy (*e.g.*, colors, line style), Acc_d, user feedback indicated that decoration accuracy was similar in accuracy to the synthetic dataset (we are validating this result in future work with human studies).

Integrating more style transfer mechanisms into our current dataset is essential to enhance the model robustness. We also incur bias in our current evaluation on real hand-drawn sketches due to the limited number of drawers. To enhance the accuracy of our evaluations we are collecting sketches from more people, as discussed in Section 2.7.

2.6 Related Work

This section compares related work with the techniques explored in the dissertation.

Data Visualization Efficiency. As an emerging capability in data-driven applications, data visualization has drawn attention far beyond the visualization community, in nearly every domain, ranging from healthcare to smart cities. Due to the broad expansion of users, data visualization efficiency has become a critical requirement. Various applications have been designed to reduce the visualization learning curve and enhance usability for engineers who interact with visualization tools.

For example, FlowSense [49] allows users to construct dataflow diagrams from English sentences. Arjun [23] implemented a tablet-based data exploration tool with multimodal (i.e., pen, touch and voice) interactions. NL4DV [50] is a Python tool that helps developers create specifications by taking natural language(s) as input. Data2Vis [51] adopted Vega-Lite [52] as an interpreter to generate visualizations automatically from data specifications to visualization specifications. Qin [53] explored three paths to enhance efficiency in data visualization:

1. Improve the designation and input of visualization specifications,
2. Provide various approaches for data visualization, and
3. Automatically correct or refine users' generated visualization.

Our work builds upon achievements in path #1 and mainly focuses on path #2. We are also adding complementary mechanisms to our current model inspired by path #3. Our goal is to produce adequately detailed visualizations from hand-drawn sketches, thereby saving time producing the initial visualization implementation that can be later refined manually.

Sequenced Output in Deep Learning. Prior research on human-computer interactions via natural means (such as sketches) relied on enforcing strict constraints on users due to the challenge of processing informal human input by machines [54]. For example, earlier pen-based computer interactions required users to learn an unnatural way to draw sketch diagrams so computers can understand them [55, 56]. To overcome this challenge, shape recognizers were proposed to interpret pen strokes in human-drawn sketches and trained via a cluster-based approach that groups streams of pen strokes [54, 57]. Shape recognizers grant computers the ability to read users' drawing with more flexibility and can be extended by improving recognizers with additional types of supporting shapes.

Deep learning networks provide a solution for computers to extract features from natural human input, such as images, text, and voices [5, 58, 48, 59]. Beyond image classification, researchers attempt to use images to explain more complex human-comprehensible output, such as natural language expressions [60]. The encoder-decoder structure was adopted in the image captioning problem [37], thereby combining advantages of CNNs in processing images and RNNs in processing sequences of natural language expressions.

As an object detection mechanism, faster R-CNNs [61] were introduced for models to output captions in the context of classified objects. Transformers [6] further enhanced model performance and training speed by substituting RNNs as encoder and decoder. In contrast, the image-to-code problem shows significant compatibility with GUI components.

For example, pix2code [33] was proposed to transfer UI screenshots of web applications into HTML components and was further extended to mobile app development[62].

2.7 Concluding Remarks

This dissertation analyzed the feasibility of generating data visualizations from hand-drawn sketches, a problem that we call Sketch2Vis. As shown in this dissertation, DSLs and style transfer approaches can be combined to help generate labeled sketches with associated source code. This combination helps to overcome the key challenge of scalably obtaining a dataset for training. Our experiments showed models capable of reaching 95% structural accuracy on hand-drawn sketches. In addition, the results show that synthetically generated images that use style transfer to make them look hand-drawn is an effective approach for generating a labeled training set.

The following is a summary of the lessons we learned from conducting the research presented in this dissertation:

- **Models can learn and generalize from synthetically generated sketches produced with style transfer approaches.** The models achieved an structural accuracy of 97% on this style transfer training set and 95% structural accuracy on the hand-drawn validation set – indicating that models can train on and generalize using synthetically produced sketches.

- **The CNN-Transformer structures performed best**. The results of on our experiments with RNN and Transformer models showed that a CNN-Transformer structure model had the best performance in processing input from multiple sources and generally demonstrated the feasibility of generating simple data visualizations from sketches. Even though more real hand-drawn sketches are needed for solid validation.

- **Labeling visualization instances with a DSL was a scalable way to address visu-**

alization automation problems with deep learning. Dataset construction methods in our experiments showed how describing visualization images with DSL instances enabled deep learning models to extract features from single plot, and potentially allow users to create subplots.

Chapter 3

Case study: A Lightweight Solution for Leveraging Raw Waveforms in Spoof Speech Detection

This chapter is adapted from ”Arawnet: A lightweight solution for leveraging raw waveforms in spoof speech detection” published in IEEE ICPR 2022 26TH International Conference on Pattern Recognition (ICPR) and has been reproduced with the permission of the publisher and my co-authors Quchen Fu, Jules White, Maria Powell, and Douglas C. Schmidt.

- Zhongwei Teng, Quchen Fu, Jules White, Maria Powell, and Douglas C. Schmidt. Arawnet: A lightweight solution for leveraging raw waveforms in spoof speech detection. In IEEE ICPR 2022 26TH International Conference on Pattern Recognition (ICPR). IEEE, 2022

3.1 Problem Overview

Fixed, handcrafted audio features, such as Mel-filter banks [63], have shown great performance in capturing strong audio features in aspects of both auditory and machine learning [64, 65]. However, since handcrafted features are often designed based on specific tasks, such as speech recognition, using these features to solve problems that they were not designed for may not be optimal. For example, Mel-filter banks [63] apply triangular filter banks on a Mel-scale to spectrograms calculated using short-term Fourier transform (STFT) to represent the non-linear perception of the human hearing. The Mel-scale is derived from a set of perception experiments on humans. As a result, Mel-filter banks are coarse-grained at high-frequencies since humans are less sensitive to high frequency sound. This loss of signal energy (information) in high frequencies may lead to poor performance on tasks that rely on information in these higher frequencies [65].

Extracting audio features with backpropagation provides an alternative way to represent raw waveforms by using deep neural networks to learn task-specific features. Task-specific features can be learned for many problems, such as voice recognition[66, 67] or automatic speaker verification (ASV) [68]. Directly learning features from raw waveforms grants greater flexibility in handling unknown tasks and, thus, overcomes some of the challenges of handcrafted features, which may lose signal energy needed by a specific task. Previous research indicates that representations learned from waveforms still have limitations on signal energy loss compared to the original raw signals they were learned from [64]. In the spoof speech detection task, models based on only raw waveforms perform better in specific spoof attacks, while shows weaker performance on other attacks compared to model based on handcrafted data [69, 70].

Instead of relying on raw waveforms independently, a potential option is to design a solution that can take advantage of both handcrafted and learned features. For example, lost phase information in handcrafted features can be complemented by features learned from raw waveforms. However, building an end-to-end CounterMeasure (CM) systems, containing multiple encoders to process raw waveforms and hand-crafted features independently, can result in big challenges in model size and complexity.

In this chapter, we propose the Auxiliary Rawnet (ARNet) architecture to combine learned features from raw waveforms with existing handcrafted features, by designing a lightweight auxiliary encoder. The proposed model was tested on the ASV Spoof 2019 dataset [71], where the model needs to defend against speech spoofing attacks from a variety of sources. The model shows promise in boosting the performance of handcrafted feature-based networks that warrants further investigation on additional data sets and tasks.

The key contributions of this chapter are as follows:

- We elaborate on the problem of concatenating raw waveforms and handcrafted features in the speech field and propose assumptions to solve this problem efficiently.

- Based on our assumption, we introduce the Auxiliary Rawnet architecture that can be used to attach a lightweight auxiliary encoder to a model that relies on handcrafted features, so that raw waveform data can supplement the information in handcrafted features.

- We show results that indicate that, by introducing the auxiliary raw encoder, model performance is boosted on the ASV spoof 2019 dataset. A light-weight auxiliary encoder boosts model performance over 10 of 13 spoof attacks, including 3 challenging attacks.

- We describe how our results show the potential of combining a light-weight waveform encoder with other encoders, providing an approach to balance the trade-off between performance and model complexity for models containing multiple encoders.

The remainder of this chapter is organized as follows: Section3.2 discusses prior work in audio signal feature representation. Section3.3 explains the problem analyzed in this chapter and describes the Auxiliary Rawnet structure. Section3.4 introduces the experimental dataset and tasks used in this chapter. Experimental results are analyzed in Section3.5. Section3.6 presents concluding remarks and lessons learned.

3.2 Related Work

Prior work has shown how the ”front-end” of models, which extract features from raw data, can be improved by using deep neural networks [64, 72, 67, 65, 73] to directly learn features from raw signal data. Directly applying standard CNNs to process raw waveforms [74] has shown promising results in speech recognition, spoofing detection, and speech separation. Convolutions on time-domain raw waveforms can be explained as finite impulse response filter banks [64]. Structured filters are applied to optimize standard CNNs based on digital signal processing theory, by initializing the first convolutional layer, which is believed to be the most important part, with known filter families [73, 75, 76], so that a custom filter bank can be designed for a specific task. Filter-based waveforms networks are emerging as excellent front-ends for many tasks [69, 65]. However, a theoretical analysis from Joakim et al. [64] has shown that signal energy loss is still inevitable for features extracted from raw waveforms by a CNN. Their results show extracted features can carry up to 94.5% signal energy compared to the original waveforms. On the other hand, empirical research also indicates that handcrafted features are still competitive in specific questions, such as speech commands [65], spoof speech detection [71, 77], and instrument classification [65].

In prior work on spoof detection, a deep neural network, called RawNet2 [69], used raw waveforms to enhance the performance of CM systems against certain types of spoof attacks, but at the expense of increased model size and computational complexity. The prior work showed that models relying solely on raw waveforms showed weaker performance on many types of spoof attacks, resulting in worse overall performance on spoof detection according to the ASV spoof dataset[69]. It is challenging to create an end-to-end network, which can take both raw waveforms and hand-crafted features as input due to the increase in model size that accompanies raw waveform use. This chapter considers the use of raw waveforms as a supplement to handcrafted features, rather than the main input, for spoof detection and investigates their potential to boost performance with little additional computational cost. To the best of our knowledge, our work is the first study on the bottleneck structure of hand-crafted features and raw waveforms in deep learning models for spoof detection. The chapter also presents the first architecture to apply both hand-crafted features and raw waveforms in an end-to-end model for spoof detection tasks.

3.3 Auxiliary RawNet

This section elaborates the research problem on combining raw waveforms and hand-crafted features and explains the structure of the ARNet architecture.

The proposed network architecture applies a light-weight encoder to process raw wave-

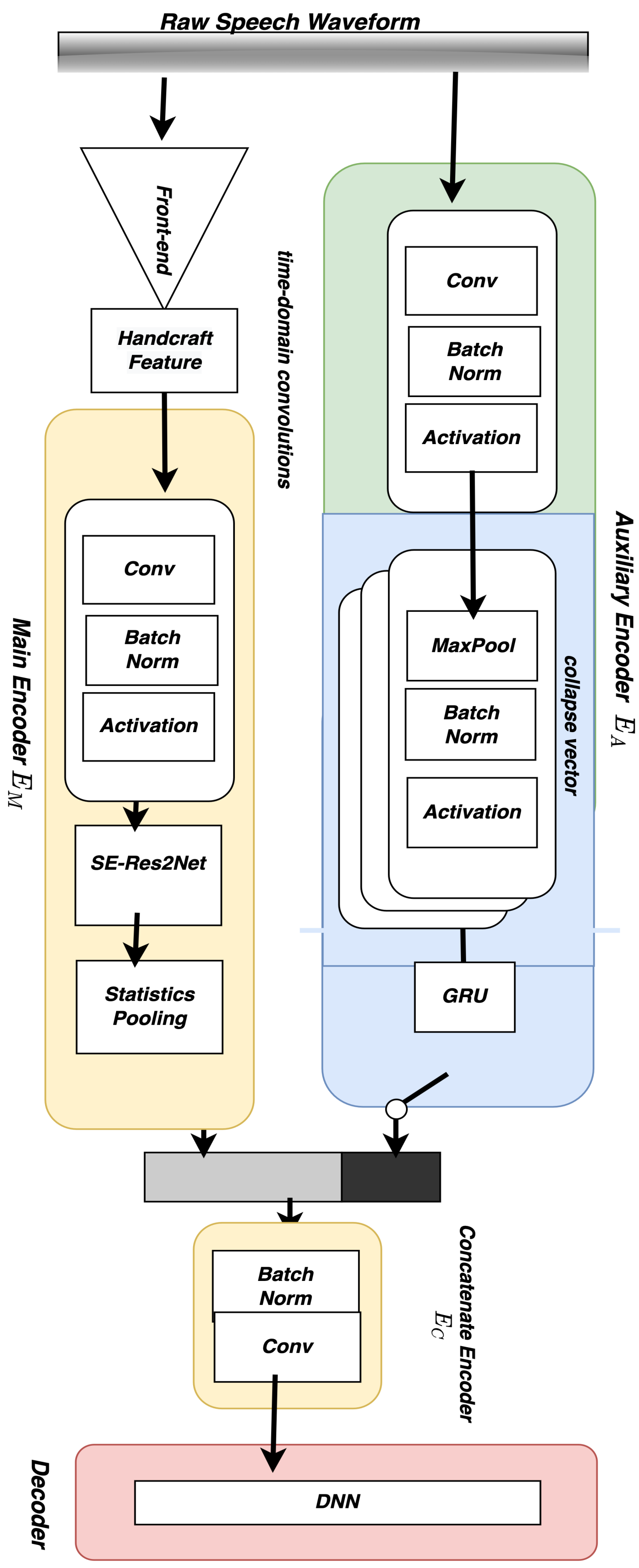


Figure 3.1: The ARNet Architecture. E_A contains one strided CNN, 3 continuous max-pooling layers and a GRU. A TDNN-based model is illustrated here as an example of the E_M.

forms with low computational cost as learned features, which are combined with existing speech classification models (Figure 3.1). To produce disentangled representations from different encoders, a narrow bottleneck is leveraged in the raw waveforms encoder without damaging the performance of the handcrafted encoder, as shown in Figure 3.2.

3.3.1 Problem Formulation

Before introducing the ARNet architecture, we first formalize the problem that it is intended to solve. Denote F_w as features of a raw waveform, and p as a problem to solve. We assume there is a constructive function f, which can map $F_{p_{mag}}$, $F_{p_{phase}}$ and $S_{p_{noise}}$ into F_w, as described in Equation 3.1, where $F_{p_{mag}}$ is the ideal magnitude information needed to solve p, $F_{p_{phase}}$ is the ideal phase information needed to solve p, and $S_{p_{noise}}$ are signals with limited contribution to solving p (e.g., background noise).

$$F_w = f(F_{p_{mag}}, F_{p_{phase}}, S_{p_{noise}}) \tag{3.1}$$

Empirical studies [65] have shown the ability of handcrafted features to represent the strongest audio features for a variety of problems. Based on our assumption, the calculation of handcrafted features can be denoted as a mapping function g, which can retrieve approximations of $F_{p_{mag}}$ or $F_{p_{phase}}$. For example, mel-spectragrams can be described by the following equation:

$$F_{p_{mag}} \approx F_{mel} = g_{mel}(|STFT(F_w)|^2)) \tag{3.2}$$

When concatenating raw waveform data and handcrafted features to enhance model performance, our work is essentially to find a function, h, so that the total loss of $g(F_w)$ and $h(F_w)$ is smaller than a single $g(F_w)$. In other words, we want to find representations closer to the ideal solution $F_{p_{mag}} + F_{p_{phase}}$, as describe in Equation 3.3.

$$concat(g(F_w), h(F_w)) \approx F_{p_{mag}} + F_{p_{phase}} > g(F_w) \tag{3.3}$$

However, it is not clear how $g(F_w)$ interacts with $h(F_w)$. Inspired by observations from results regarding $g(F_w)$ and $h(F_w)$ on various tasks [65, 69], we make the following assumption about combining learned features and handcrafted features:

Assumption 1 (A1): *If a handcrafted feature, $g(F_w)$ shows strong results solving problem p, then there exists a $h(F_w)$ with size less than N in $concat(g(F_w), h(F_w))$ that will enhance overall performance. In other words, $h(F_w)$ can be an auxiliary component of $g(F_w)$ to improve performance with a bounded cost.*

3.3.2 The Auxiliary RawNet Structure

Based on the assumptions presented in section 3.3.1, we propose the ARNet architecture. An overview of the ARNet architecture is shown in Figure 3.1. E_A, which processes the raw waveform, has a smaller bottleneck than E_M which processes handcrafted audio features, to make the raw waveforms play a supplementary role and bound the computational cost (e.g., bound N).

The Encoders. There are 3 encoders in the ARNet: the Main Encoder(E_M), Auxiliary Encoder(E_A), and Concatenate Encoder(E_C). E_M denotes the main encoder, whose inputs are the original handcrafted features that have shown good performance in solving the target problem. E_A is the encoder used to encode the raw waveforms in a light-weight way to compress F_w into F_a, where F_a are the features extracted by the auxiliary encoder. F_a and F_m (hand crafted features from the main encoder) are then concatenated in channels and further encoded by E_C.

Figure 3.1 shows details of the encoders used in our experiments on the ASVspoof 2019 dataset. We select the strided convolutional layer[67] as the first layer to directly process

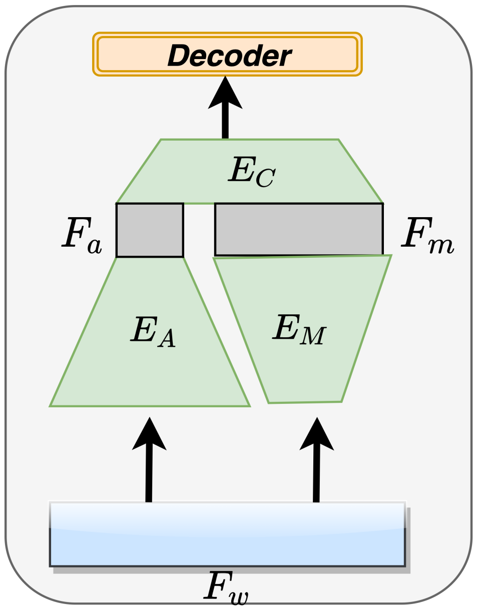


Figure 3.2: Overview of the ARawNet Architecture. The model consists of a Main Encoder(E_M), Auxiliary Encoder(E_A), and Concatenate Encoder(E_C). E_A has a smaller bottleneck than E_M.

the raw waveforms. However, unlike previous raw waveforms networks, which include multiple CNN blocks with large kernels, the strided convolutional layer is only followed by 3 continuous pooling blocks to collapse vectors and remove any frame variance without further convolution. A GRU is used to encode frame-level features into utterance-level embeddings by keeping output vectors from the last time step.

The main encoder keeps layers before the statistical pooling layer, which will output utterance-level embeddings. Based on our assumption 1, we chose a narrow bottleneck for E_A. The dimension of the utterance-level embedding from E_A is designed to be smaller than the output dimension from E_M. In the end, E_C only contains a single Conv1d to encode concatenated results from E_A and E_M.

The full architecture and model hyper-parameters are explained in Table 3.1.

The Decoder. In our problem, the decoder is a linear classifier layer that decodes embeddings from E_C to target classification.

Encoders	Blocks
Auxiliary Encoder	Conv(3,3,128) BN&LeakyReLu MaxPooling BN&LeakyReLu GRU(512)
Concatenate Encoder	BN Conv1D(1,1,256)

Table 3.1: The architecture of Auxiliary Encoder and Concatenate Encoder.

3.3.3 Why does a light-weight encoded raw waveforms augment handcrafted features?

(1) Compared to the current filter-based architectures as discussed in Section 3.2, we chose the strided convolutional receptive field, which is a standard CNN, as the first layer to process the raw waveforms. The strided convolutional layer consists of a set of time-domain convolutions, where all parameters(CNN kernel), are learned from the data. Calculation of the first CNN layer can be described as the following Equation [73], where x[n] is raw waveforms, h[n] is the filter and y[n] is filtered output:

$$y[n] = x[n] * h[n] = \sum_{0}^{L-1} x[l] \cdot h[n-l] \tag{3.4}$$

As discussed in Section 3.3.1, concatenating $g(F_w)$ and $h(F_w)$ requires each encoder to have different attention to features in the raw waveforms so that they can complement each other. The standard convolutional layer with small kernels gives the E_A the least information about the signal processing mechanisms in $g(F_w)$, and thus potentially grants it the most flexibility to extract features, which do not overlap with $g(F_w)$.

(2) In contrast to previous waveform-based networks [67, 69], the CNN blocks used in between the strided convolution layer and the GRU are completely removed, and only 3 continuous max-pooling layers with batch normalization are kept to collapse frame-level features step-by-step.

The first convolutional layer is considered the most critical part in processing raw waveforms. In deep networks it is also the most vulnerable to problems, such as vanishing gradients, without initializing filters [73]. However, based on our assumption 1, only significant frame-level features need to be kept, indicating networks without deep CNN blocks can be used for E_A. Max pooling layers are used to collapse vectors and find significant pattern information that can be visualized after 3 pooling layers, as shown in Figure 3.3.

(3) We test our assumption 1 based on the Theorem [78] from speech conversion problems, that if information bottlenecks between different encoders are precisely set, the model will decompose and produce disentangled representations of input speech signals. In our model, this Theorem can be described by the following equation:

$$E_M(F_w) = g(F_w), E_A(F_w) = h(F_w) \tag{3.5}$$

Thus, a narrow bottleneck is designed for E_A, which means the dimension of utterance-level embeddings dim_{E_A} is much smaller than dim_{E_M}.

(4) Output embeddings from E_M and E_A are concatenated in the utterance level, where segment-level layers in single encoder are removed and are replaced with concatenate encoder E_C. This is a critical features for the ARawNet, as well as for our assumption 1, that original bottleneck layers from E_M and E_A should be replaced with a E_C. We concluded the following hypothesis as we designed the network:

Assumption 2 (A2): *Given raw waveform F_w and hand-crafted features $h(F(w))$, segment-level layers need to be designed after the concatenating layer to represent disentangled representations.*

3.4 Experimental Setup

3.4.1 Experimental Dataset

The ASVspoof 2019 logical access (LA) dataset was developed to improve research on the growing threat of voice spoofing attacks on automated speech verification systems [71].

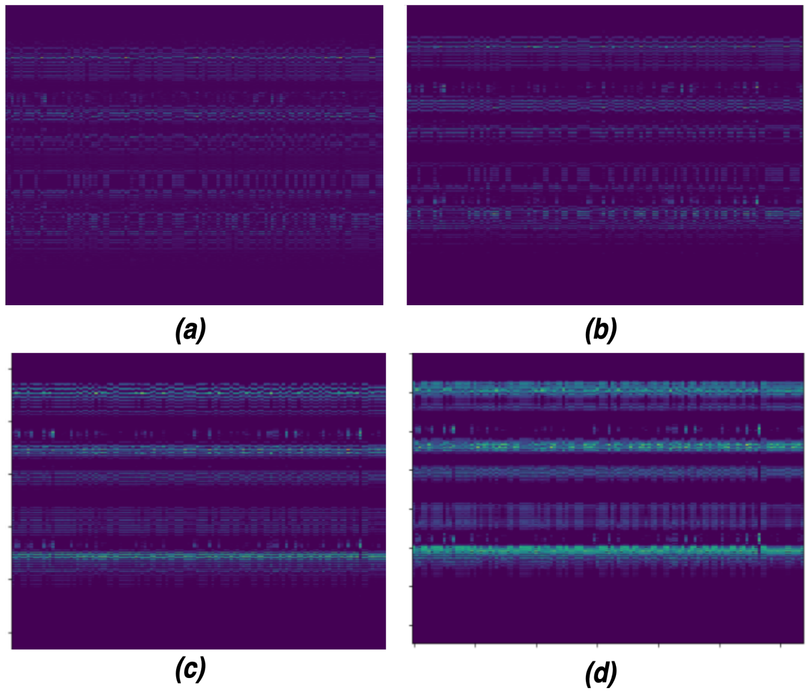


Figure 3.3: Outputs visualization of the strided convolution layer and pooling layers. Outputs after 3 pooling layers(d) shows signification pattern information.

This dataset contains human-recorded audios and spoof audios generated from 19 sources (A01 - A19), including speech synthesis, voice conversion, and hybrid algorithms.

50,224 records in the training and development data consist of spoof attacks generated by A01-A06. Another 71,237 spoof audio files in the evaluation data are generated by A07-A19, which are unpredictable spoofing attacks for CounterMeasure (CM) systems. Detailed statistics of the ASVSpoof 2019 is shown in Table 3.2.

Subsets	#Bonafide	#Spoofed	Spoof Source
Training	2580	22800	A01-A06
Development	2548	22296	A01-A06
Evaluation	7355	63882	A07-A19

Table 3.2: Statistics of the ASV2019 dataset

We chose the ASVspoof 2019 LA dataset to validate the performance of our proposed model since:

- The performance of handcrafted features is limited by the difference in spoofing sources between the training and evaluation data. Spoofing types are highly unpredictable while the performance of CM systems relies on known spoofing attacks in training data and shows worse performance on unknown spoof attacks.

- Current results on the ASVspoof 2019 challenge [71, 69] indicate that correct handcrafted features still provide the most competitive results from a single model compared raw waveforms approaches.

- Waveforms-based network outperforms on specific types of spoof attacks with worse pooling results compared to handcrafted feature based networks [69].

3.4.2 Evaluation Metrics

Two metrics are used to evaluate the ASVspoof 2019 LA dataset including *min t-DCF* as the primary metric and $EqualErrorRate(EER)$ as a secondary metric, as described in [71]:

3.4.2.1 *min t-DCF*

The Tandem Detection Cost Function (t-DCF) [79] extends the conventional Detection Cost Function (DCF) in voice verification systems for spoofing attacks. The t-DCF measures the overall effect of CM systems combined with existing ASV systems. The CM system acts as a gateway for the ASV system and this metric measures the overlapping of the two, a smaller value indicates better protection against spoofing. min t-DCF can be calculated in Equation 3.6 [79], where $P_{\text{miss}}^{\text{cm}}(s)$ is the CM miss and $P_{\text{fa}}^{\text{cm}}(s)$ is the false alarm rates.

$$\mathrm{t-DCF}_{\text{norm}}^{\text{min}} = \min_{s}\{\beta P_{\text{miss}}^{\text{cm}}(s) + P_{\text{fa}}^{\text{cm}}(s)\} \tag{3.6}$$

Table 3.3: Model Performance by Spoof Category.

Spoof Attack [71]		A7	A8	A9
Category		TTS	TTS	TTS
Acoustic mode		LSTM-RNN	AR LSTM-RNN	LSTM-RNN
Waveform generator		WORLD+GAN	Neural source-filter	Vocaine
EER	CQT+Aux	0.28521	2.4786	0.08149
	CQT	0.3667	1.72823	0.146
	Effect(%)	22.2	-43.4	44.1
min-tDCF	CQT+Aux	0.00895	0.07071	0.0022
	CQT	0.01168	0.04895	0.00424
	Effect(%)	23	-44	48

Spoof Attack [71]		**A10**	A11	A12
Category		TTS	TTS	TTS
Acoustic mode		Attention seq2seq	Attention seq2seq	-
Waveform generator		WaveRNN	Grin-Lim	WaveNet
EER	CQT+Aux	**0.46516**	0.30218	0.24447
	CQT	**0.79111**	0.77414	0.3667
	Effect(%)	**41.2**	60.9	33.3
min-tDCF	CQT+Aux	**0.01351**	0.00951	0.00703
	CQT	**0.02326**	0.02527	0.01226
	Effect(%)	**41**	62	42

Spoof Attack [71]	**A13**	A14	A15
Category	TTS-VC	TTS-VC	TTS-VC
Acoustic mode	Moment matching NN	LSTM-RNN	LSTM-RNN

Waveform generator		Waveform filtering	STRAIGHT	WaveNet
EER	CQT+Aux	**0.12223**	0.12223	0.30218
	CQT	**0.22069**	0.28521	0.5127
	Effect(%)	**44.6**	57.1	41
min-tDCF	CQT+Aux	**0.00376**	0.00376	0.00907
	CQT	**0.00702**	0.00874	0.01654
	Effect(%)	**46**	56	45

Spoof Attack [71]		A16	A17	**A18**
Category		TTS	VC	VC
Acoustic mode		-	VAE	i-vector/PLDA
Waveform generator		Waveform concat	Waveform filtering	MFCC-to-waveform
EER	CQT+Aux	0.24447	3.11354	**0.8726**
	CQT	0.22069	2.40391	**5.28996**
	Effect(%)	-10.7	-29.5	**83.5**
min-tDCF	CQT+Aux	0.00768	0.09041	**0.02837**
	CQT	0.00649	0.07587	**0.16156**
	Effect(%)	-18	-19	82

Spoof Attack [71]		A19		
Category		VC		
Acoustic mode		GMM-UBM		
Waveform generator		Spectral filtering		
EER	CQT+Aux	1.05935		
	CQT	2.09493		

	Effect(%)	49.4		
min-tDCF	CQT+Aux	0.0361		
	CQT	0.06432		
	Effect(%)	43		

3.4.2.2 *EER*

EER indicates the threshold of a CM system where the false positive and false negative rates are equal each to other.

3.4.3 Hand-crafted feature selection

To validate assumption 1, as well as the performance of our ARawNet implementation, hand-crafted features need to be selected carefully in our experiments. According to our assumption, it is important to ensure that the selected hand-crafted features show strong results on the target dataset, so that we can use a lightweight network to enhance the overall performance without a large increase in network complexity.

To reduce the performance bias of a single hand-crafted feature in our experiment and choose appropriate features, we carefully reviewed prior work on the ASVspoof 2019 challenge and chose the state-of-the-art model as our benchmark, so that the features we selected have validated performance on the ASVspoof 2019 dataset. By choosing validated hand-crafted features with state-of-the-art performance, we can be more confident in the analysis and exploration of our assumption on augmentation with raw waveforms.

Specifically, 3 hand-crafted features were selected in our experiments as described below:

- **Mel-Spectrogram**. Mel-Spectrogram is a Short-Time Fourier Transform (STFT) based frontend, which is used to represent the non-linearity of the human ear's sensitivity to different frequencies by applying filters in a mel-scale, as shown in Equation 3.7.

$$m = 2595 \log_{10}\left(1 + \frac{f}{700}\right) \tag{3.7}$$

- **Mel-frequency cepstral coefficients(MFCC)**. MFCC is a popular feature for speech recognition, which is a set of coefficients of the mel-frequency cepstrum. MFCC is calculated by applying Discrete Cosine Transform (DCT) to Mel-Spectrograms to decorrelate them.

- **Constant Q Transform (CQT)**. CQT is a feature to improve downstream tasks, where mel-scale does not perform well, such as music recognition. The CQT center frequency calculation is shown in Equation 3.8 [80].

$$f_k = f_0 2^{\frac{k}{B}} \tag{3.8}$$

3.4.4 Baseline Setup

Our experiments include one handcrafted feature-based system and one raw waveform-based system respectively:

- **Res2net Architecture**. The Res2net architecture [70] is the state-of-the-art single system in the ASVspoof 2019 challenge, which tested the performance of 3 handcrafted features: log power magnitude spectrogram (Spec), linear frequency cepstral coefficients (LFCC), and constant-Q transform (CQT).

- **RawNet2**. The RawNet2 [69] is the first anti-spoofing model, which only relies on the raw waveforms as input.

3.5 Results and Analysis

Table 3.4 shows the experimental results of the ARawNet on the ASVSpoof 2019 dataset.

Front-end	Main Encoder	E_A	EER	min-tDCF	Improve
Spec[70]	Res2Net[70]	-	8.783	0.2237	
LFCC[70]		-	2.869	0.0786	
CQT[70]		-	2.502	0.0743	
Raw waveforms[69]	Rawnet2[69]	-	5.13	0.1175	
Mel-Spectrogram	XVector	✓	**1.32**	0.03894	43%
		-	2.39320	0.06875	
Mel-Spectrogram	ECAPA-TDNN	✓	**1.39**	0.04316	32.8%
		-	2.11	0.06425	
CQT	XVector	✓	**1.74**	0.05194	45.3%
		-	3.39875	0.09510	
CQT	ECAPA-TDNN	✓	**1.11**	0.03645	28.2%
	.	-	1.72667	0.05077	
MFCC	XVector	✓	**1.39**	0.03830	45.1%
	.	-	2.45	0.06981	
MFCC	ECAPA-TDNN	✓	**1.33**	0.04260	37.7%
	.	-	2.41	0.06838	

Table 3.4: Results on the ASVspoof 2019 dataset

- **Results demonstrate the effectiveness of adding a light-weight auxiliary encoder to the main encoder.** Three handcrafted features, Mel-spectrogram, CQT [80] and MFCC, as well as two state-of-the-art models in the speaker verification problem (XVector [81, 82] and ECAPA-TDNN [83, 82]) are selected as main encoders in the ARNet architecture. Without modifying the hyper-parameters in the main encoder, we add the auxiliary encoder, as described in Table 3.1, in the network to evaluate our assumption. Overall, by introducing the auxiliary encoder, both pooled EER and $min-tDCF$ are reduced by $\tilde{5}0\%$ in all combinations of front-end and main encoders. Specifically, CQT/ECAPA-TDNN with auxiliar encoder reaches the best performance on EER of 1.11% and $min-tDCF$ of 0.0364, which is reasonable since single CQT perform best in given benchmark[70].

- **Performance of the CM system on challenging attacks (A10, A13, and A18) is boosted when using CQT as an input feature.** Table 3.3 explains the category of different attacks in the evaluation set and breaks down the performance of our model in a different subset of spoof attacks. Overall, detection of 10 of 13 spoof attacks was improved with the lightweight auxiliary model, and the auxiliary model boosts detection of most attacks generated from LSTM-based models. Among those attacks, A10 [84], A13 [85], and A18 [86] are considered as high risks to ASV systems as well as challenging attacks for CM systems to detect [71]. As shown in Table 3.3, the ARawNet largely decreased the EER and the min-tDCF for A10, A13, and A18. Especially for attack A18, model performance improves by over 80%, the EER reduced from 5.29% to 0.87% and min-tDCF reduced from 0.162 to 0.028. However, we also noticed that there are negative effects of including a light-weight raw waveform encoder on certain spoof attacks. For example, the A17 attack, which is generated by variational autoencoder(VAE) [87], is hard for CM systems to detect even though it is a minor threat to the ASV system [71]. Prior works show that a raw waveform-based network with deep layers shows a better performance on this type of attack [69]. This result indicates that we may need to enlarge the network to enhance model performance on specific types of spoof attacks.

- **By introducing the lightweight raw-waveform encoder, the model is less sensitive to highly non-linear information from hand-crafted features.** Mel-spectrograms have become more popular than MFCCs in recent research, since deep neural networks are less likely to be weakened by highly correlated input. After whitening the mel-spectrogram, highly non-linear information is removed in MFCCs, which may be useful for networks. Results in our experiments show that, without raw waveforms, mel-spectrogram shows better performance than the MFCC, indicating that highly non-linear information plays an important role in recognizing spoof attacks. While results of both features tend to be similar after introducing the lightweight raw-waveform encoder. It implies that the auxiliary encoder disentangles raw waveforms and helps the model complement lost highly correlated information in MFCCs.

- **Improper concatenation of the raw waveform encoder output and hand-crafted encoder output can lead to worse results.** Table 3.5 shows the performance com-

Features	Concatenate	EER	min-tDCF	Parameters
CQT	Before segment layer	1.11	0.03645	10.4 M
	After segment layer	1.36	0.04052	7.18 M

Table 3.5: Model Performance with different Concatenating Layers

parison of models with different concatenating strategies. As we discussed in hypothesis 2, in spoof attack tasks, segment-level layers need to be placed after concatenating utterance-level features from different encoders. Using CQT as features and ECAPA-TDNN as the main encoder, we adjust the model by concatenating segment-level features rather than utterance-level features. Results shows EER increased by 36% with 45% increment on the trainable parameters.

- **ARawNet adds raw waveform information to the spoof detection tasks with a smaller network size than prior approaches.** Table 3.6 compares the number of trainable parameters, model complexity, and multiply-and-accumulates (MACs) in our experiments. Compared to encoding handcrafted features (Res2Net), directly encoding raw waveforms (Rawnet2) increases model size and complexity by 2400% and 600%. On the other hand, our auxiliary waveforms encoder only takes up 1.15M trainable parameters, which is a 19% increase in ECAPA-TDNN and the model complexity increases from 2.36 GMac to 3.19 GMac. In other words, the performance of our model increases by 28.2% with increments of 35.1% MACs.

  The smaller network size allows us to train the model on 2 Nvidia 2080 Ti GPUs with the batch size set to 24.

3.6 Concluding Remarks

This chapter discussed the problem of combining learned and handcrafted features to build deep neural networks for the spoof voice detection task. Based on our assumption that hand-crafted features and raw waveforms may complement each other without sacri-

Main Encoder	Auxiliary Encoder	Parameters	MACs
Rawnet2	-	25.43 M	7.61 GMac
Res2Net	-	0.92 M	1.11 GMac
XVector	✓	5.81 M	2.71 GMac
XVector	-	4.66M	1.88 GMac
ECAPA-TDNN	✓	7.18 M	3.19 GMac
ECAPA-TDNN	-	6.03M	2.36 GMac

Table 3.6: Comparison of model complexity (MACs) of various spoof detection systems

ficing model complexity, we investigated the concatenation of multiple encoders and proposed ARawNet, which includes both hand-crafted features and raw waveforms as inputs, while maintaining a relatively small network size. We tested 3 hand-crafted features (Mel-spectrogram, MFCC, an CQT) and 2 state-of-the-art models (XVector and ECAPA-TDNN) as the main encoder with our Auxiliary Encoder. Experiment results show raw waveforms have the ability to complement CQT for detection of most spoof attacks in the ASVspoof 2019 dataset, as well as its ability to complement the highly non-linear information for MFCC features.

Chapter 4

Case study: An End-to-End Spoof-Aggregated Spoofing-Aware Speaker Verification System

This chapter is adapted from ”SA-SASV: An end-to-end spoof-aggregated spoofing-aware speaker verification system” published in Interspeech 2022-23rd Annual Conference of the International Speech Communication Association. ISCA, 2022 and has been reproduced with the permission of the publisher and my co-authors Quchen Fu, Jules White, Maria Powell, and Douglas C. Schmidt.

- Zhongwei Teng, Quchen Fu, Jules White, Maria Powell, and Douglas C. Schmidt. SA-SASV: An end-to-end spoof-aggregated spoofing-aware speaker verification system. In Interspeech 2022-23rd Annual Conference of the International Speech Communication Association. ISCA, 2022

4.1 Problem Overview

Automatic speaker verification (ASV) systems have shown the ability to provide biometric authentication of users for applications that require robust reliability in changing acoustic environments, including resistance to malicious attacks [88, 89, 81, 83]. However, current ASV systems are still vulnerable to spoofing attacks, such as text-to-speech (TTS) [90, 91, 92] and voice conversion (VC) [93]. ASV systems can also be deceived and manipulated by malicious entities using generated speech.

To overcome bottlenecks in spoofing and countermeasure research for ASVs, a series of ASVSpoof challenges have been proposed since 2015 to help encourage the development of robust countermeasure (CM) systems [94, 95, 71, 96], which can complement ASV systems with an anti-spoof model. The anti-spoof model provides a ”spoof confidence" score to help filter out spoofing attacks. Metrics on the ASVSpoof challenge are based on the minimum tandem detection cost function (t-DCF) [97], which can evaluate the performance of CM systems on fixed ASV systems with pre-determined output scores. Rather than developing CM and ASV systems independently, a neglected research question is whether we can develop an integrated system where CM and ASV system can be optimized together, so that a single verification score is able to determine whether an input speech sample is a target speaker, while also accounting for potential spoofing attacks.

To encourage research on integrated Spoofing-Aware Speaker Verification (SASV) systems, the SASV Challenge 2022 [98] was proposed using the ASVSpoof 2019 Logical Access Dataset with new metrics, SASV-EER. In the challenge, a single score determines if the input speech sample is the target speaker. Non-target inputs include zero-effort and spoofed impostors. The SASV challenge provides two baseline systems by applying different fusion strategies (score-level fusion and embedding-level fusion) to pre-trained ASV and CM systems.

Figure 4.1 shows potential solutions to the SASV problem. Red/green lines indicate the following training stages: (a) Cascaded ASV/CM systems, (b) Fusions of scoring prediction, (c)Fusions of scoring and feature embedding, (d)Fusions of feature embedding, and (e)End-to-End SASV systems.

This chapter proposes a fully trainable end-to-end SASV system, called Spoof-Aggregated Spoofing Aware Speaker Verification System (SA-SASV), that combines a pre-trained ASV system with a lightweight raw waveform encoder to form the overall encoder [99]. This chapter expands upon our prior experience that showed how encoding can be a key aspect of these types of anomaly detection problems [99, 100, 101]. Multiple classifiers and spoof-source-based triplet loss functions are employed to enhance model performance in generating the shared SASV feature space.

The remainder of the chapter is organized as follows: Section 4.2 reviews related research on SASV systems; Section 4.3 discusses the model architecture of our SA-SASV

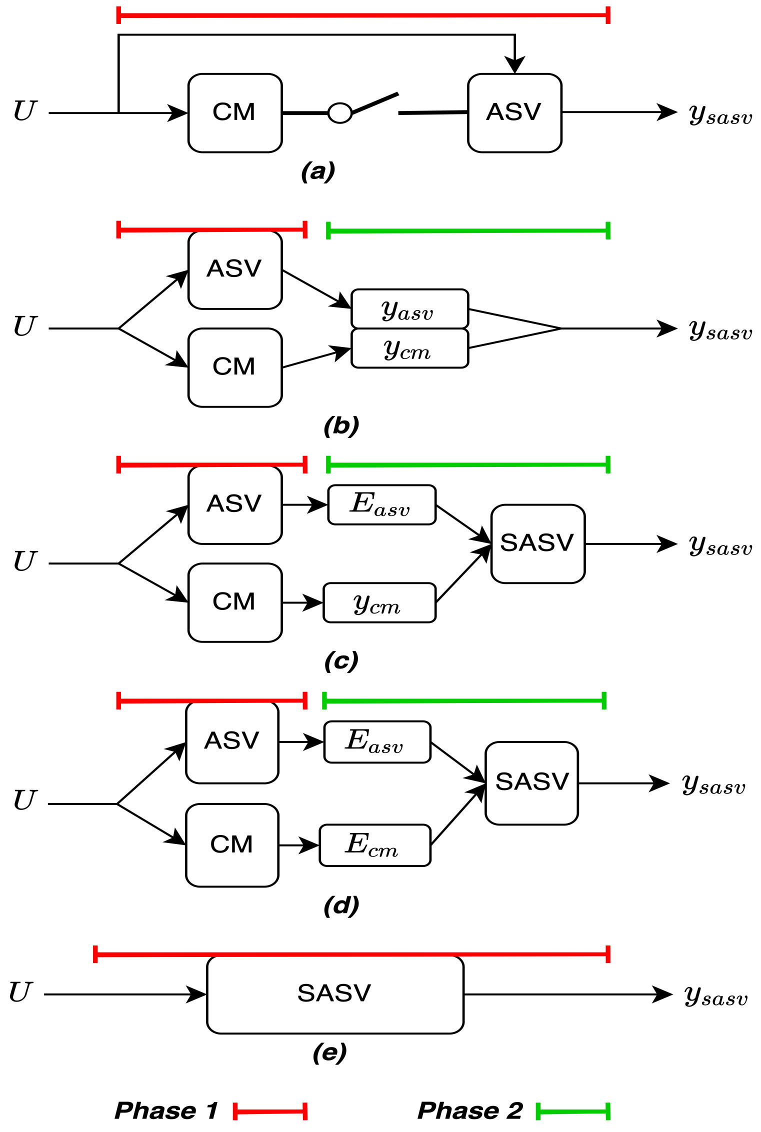


Figure 4.1: Feasible Solutions to Build Integrated SASV Systems.

Systems; Section 4.4 analyzes experiment results; and Section 4.5 presents concluding remarks.

4.2 Related Work

A SASV system aims to build a single system to reject utterances from zero-effort and spoofed speech. Previous work focused on two solutions to this problem: ensemble SASV solutions and integrated single system solutions.

Ensemble SASV solutions take fixed outputs from pre-trained ASV and CM systems and apply varying fusion strategies to generate a single SASV score for both tasks. Sizov et al. [102] was the first to apply i-vectors and a PLDA back-end for joint modeling of speaker verification and spoof detection. At the score level, Todisco et al. [103] proposed a two-dimensional score modeling method to get a single score threshold for both ASV and CM systems.

Shim et al. [104] discusses a back-End modular approach to train embeddings from pre-trained fixed ASV systems and spoofing predictions from CM systems to predict final SASV scores. In addition to scoring ensembles, fusions based on embeddings from different models have also been tested. For example, Gomez-Alanis et al. [105] proposed DNN-based integration methods to train three types of embeddings from ASV and CM systems jointly.

The target task of an integrated single SASV system is to build an end-to-end system that simultaneously classifies speech based on whether or not it is from the target speaker and is authentic non-spoofed speech. Zhao et al. [106] built an SR-ASV system with two classifiers to get CM scores and ASV scores from shared layers and the final decision is based on both the CM and ASV scores. Li et al. [107] applied speaker-based triplet loss to train multi-task classification networks to make a joint decision on anti-spoofing and ASV.

As a form of integrated single SASV system, our method explores the shared feature space of SASV tasks. To obtain proper embeddings for speech from the multiple encoders that we employ, both hand-crafted features and raw waveforms are input into SA-SASV. We first discuss the feasibility of optimizing the SASV feature space by aggregating spoofed voice samples based on their spoofing sources. The proposed model was trained with multiple loss functions, including spoof source-based triplet loss. The final decision by our model is based on cosine similarity and CM scores from same model.

4.3 SA-SASV Model Architecture

Compared to independent CM and ASV models, the ideal feature space learned from SASV models should have the following characteristics: (1) spoofed and bonafide speech should be densely aggregated so that obvious margins can be drawn to separate them and (2) in the clusters of bonafide speech sources from different speakers should be sparsely distributed so that models can distinguish between different speakers. Figure 4.2 shows how the SASV system integrates the CM and ASV systems so that there are two types of boundaries to separate spoof/bonafide speech and target/non-target speakers.

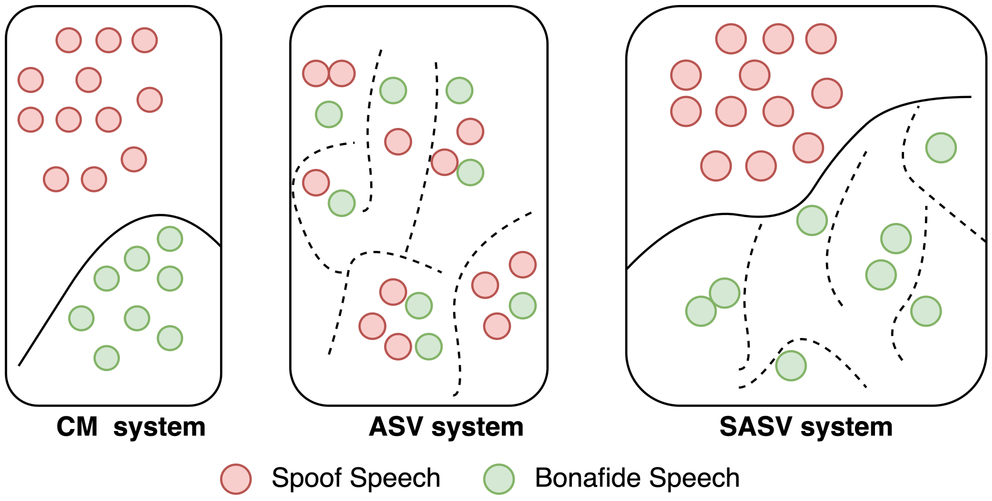


Figure 4.2: Desired Speech Sample Classification Distribution of Different Spoof Detection Systems.

To achieve optimized feature space in a SASV system, we propose the so-called SA-SASV model,, whose decode consists of three parts: multi-task classifiers, spoof aggregators, and spoof-source-based triplet loss, as shown in Figure 4.3. This figure shows how

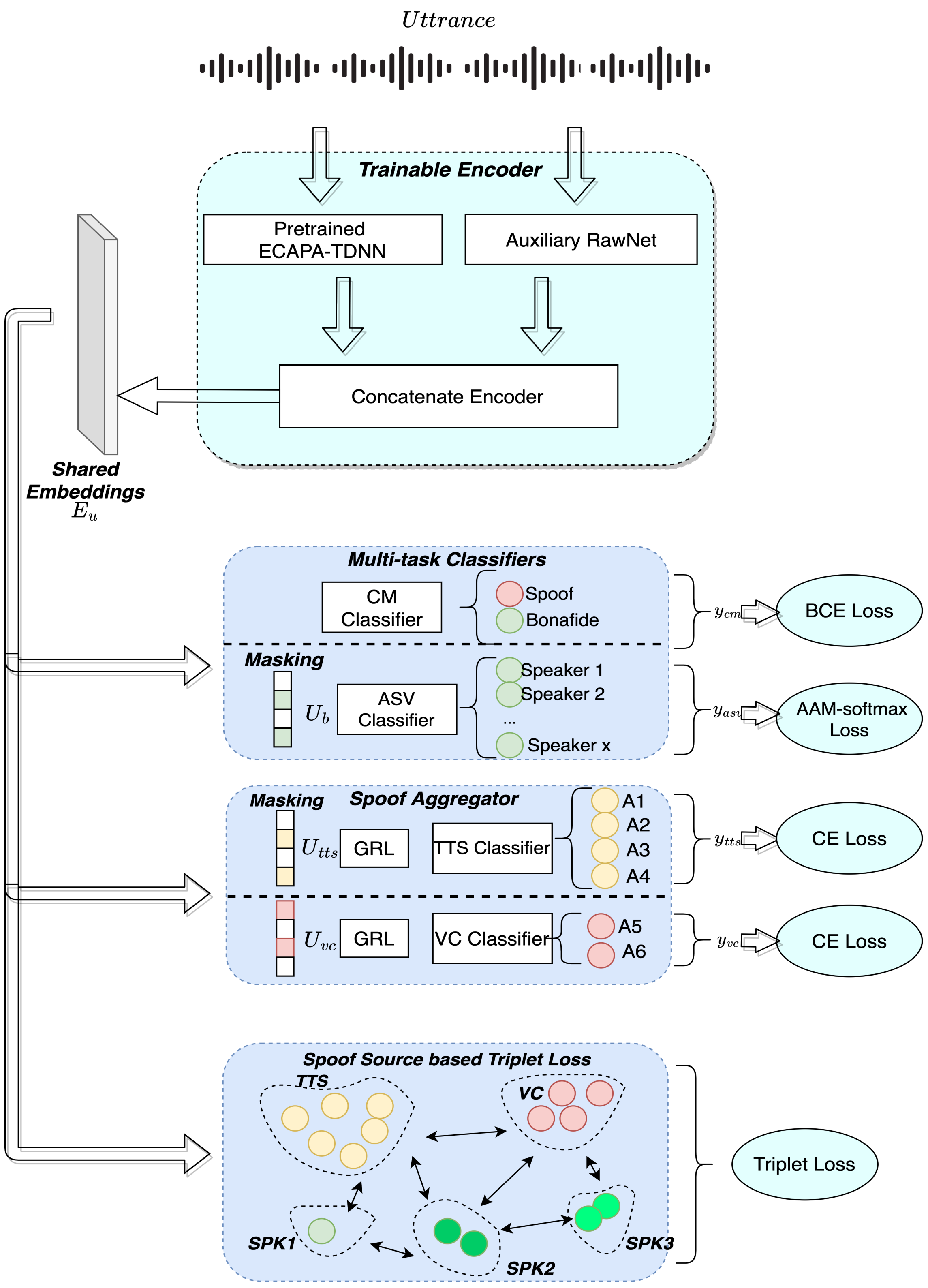


Figure 4.3: Model Structure of the SA-SASV System.

shared embedding is fed into multiple classifiers and how the feature space from the encoders is optimized by combinations of various loss functions. This fully trainable model takes both raw waveforms and hand-crafted features as input and multiple losses are used to optimize feature embedding.

4.3.1 The ARawNet Encoder

Previous research shows that the best-performing ASV systems [83] and CM systems [108], take hand-crafted features and raw waveforms, respectively, indicating distinctive features among each type of input that are useful for identifying speakers and spoofing attacks. It is hard, however, to simply merge existing state-of-the-art ASV and CM systems together to develop an end-to-end model, due to the resulting large model size and high computational complexity. We use our previously published ARawNet architecture [99] to help overcome this limitation. Our encoder combines a pre-trained ASV system (ECAPA-TDNN) and a lightweight raw waveform encoder to enable simultaneous analysis of both learned features and raw wave forms.

We denote input utterance as U. An utterance's embedding, E_u, can be described as shown in Equation 4.1, where F_{asv} is a pre-trained ECAPA-TDNN, F_{raw} is an un-trained auxiliary raw encoder, and F_c is a concatenating encoder that handles outputs from F_{asv} and F_{raw}.

$$E_u = F_c(F_{asv}(U), F_{raw}(U)) \tag{4.1}$$

4.3.2 Multi-task Classifiers

Since end-to-end SASV systems need to determine if input speech is bonafide—and if so, if it is the target speaker—this problem is formulated as a multi-task classification problem. Two classifiers are used to predict spoof attacks and speaker id independently, with shared feature embeddings from the encoder. The CM classifier C_{cm} receives all inputs and predicts confidence scores, indicating if the input is believed to represent a spoofing attack. A bonafide mask layer is placed before the ASV classifier, C_{asv}, so that losses produced by the ASV classifier are only from bonafide speech. Binary cross entropy(BCE) loss and AAM-softmax loss are used for the CM and ASV classifiers, as described in Equation 4.3 and Equation 4.2[109] respectively.

$$L_{asv} = -\frac{1}{N}\Sigma_{i=1}^{N} log \frac{e^{s(cos(\theta_{y_i,i})+m)}}{e^{s(cos(\theta_{y_i,i})+m)} + \Sigma_{j=1,j\neq i}^{j} e^{s(cos(\theta_{j,i}))}} \tag{4.2}$$

$$L_{cm} = -\frac{1}{N}\Sigma_{i=1}^{N} y_i^{cm} log C_{cm}(E) + (1-y_i^{cm}) log(1-C_{cm}(E)) \tag{4.3}$$

4.3.3 Spoof Aggregator

In the SASV task, utterances, U, consist of spoof attack samples, U_s, and bonafide speech samples, U_b. As shown in Figure 4.2, U_s should have a relatively dense distribution in the shared feature space. It is hard, however, to aggregate the various spoofing attacks together due to their intrinsic differences in speech generation methods. This inherent difficulty in separating the two is evidenced by analyzing U_s from different sources using agglomerative clustering [110], as shown in Figure 4.4. These results indicate that U_{tts} (which represents produced with Text-to-Speech(TTS)) and U_{vc} (which represents samples from Voice Conversion(VC)), tend to be closer in corresponding feature space. We therefore conjecture that U_{tts} and U_{vc} should be aggregated into two clusters in the feature space of SASV systems.

We use two adversarial learning layers to construct a spoof aggregator so that U_{tts} and U_{vc} aggregate separately. We labeled the U_s as $A1 \ldots A6$, representing the spoof type, where $A1$ to $A4$ are from U_{tts} and $A5$ to $A6$ are from U_{vc}. Followed by a masking layer, E_{tts} and E_{vc} are sent to C_{tts} and C_{vc}, where each independently attempts to predict what spoof type

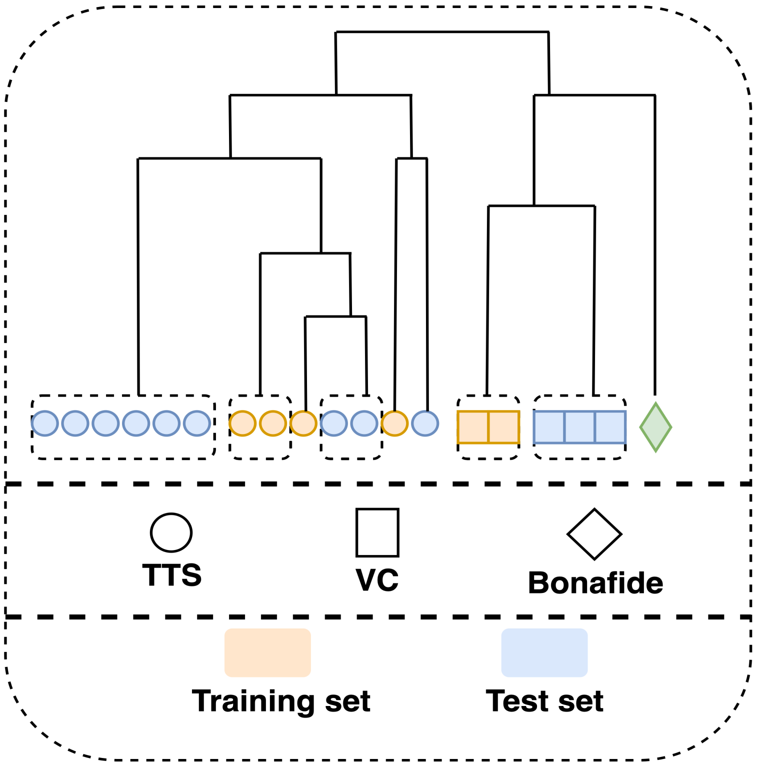


Figure 4.4: Results of Agglomerative Clustering on the ASVSpoof 2019 LA Dataset.

U_s corresponds to. The cross entropy loss for both classifiers is shown in Equation 4.4

$$L_{tts} = L_{vc} = \frac{1}{N}\Sigma_i^N y_i^A logC_{spoof}(E) \tag{4.4}$$

Since we want our embedding, E, to mix spoof attacks from the same types of generation mechanisms together, so that C_{tts} and C_{vc} fail to distinguish different spoofing attack types, a gradient reverse layer(GRL) is added before the classifiers to maximize L_{tts} and L_{vc}.

4.3.4 Spoof source based triplet loss

The shared feature space from SASV systems tends to be differentiated by U_{tts}, U_{vc}, and different speakers U_{spk_i}. In other words, the goal is for E with the same labels to be relatively compactly clustered and the overall cluster separated from E samples with

Subsets	#Bonafide	Spoofed	
		#TTS	#VC
Training	2580	A1-A4 15200	A5-A6 7600
Development	2548	A1-A4 14864	A5-A6 7432
Evaluation	7335	A7-A16 49140	A17-A19 14742

Table 4.1: Statistics of ASVSpoof 2019 LA Dataset

different labels. Boundaries between the E samples with different labels should be distinct. To help achieve this outcome, rather than applying speaker-based triplet loss, we applied spoof source-based Triplet loss. Conventional triplet loss is described as Equation 4.5:

$$L_t = \|E^a - E^p\| - \|E^a - E^n + m\| \tag{4.5}$$

As shown in Figure 4.3, E_i is labeled as TTS, VC and SPK_i, where SPK_i indicates the ith speaker. The goal is to cluster, E_i samples, with same labels as densely as possible and scatter SPK_i to make it far away from SPK_j , TTS and VC, as shown in Figure 4.5. This figure shows that positive samples (utterances with the same labels) are pulled closer and negative samples are pushed away. Thus, for an utterance from speaker i, U_{spk_i}, the spoof source based triplet loss is updated as shown in Equation 4.6.

$$\begin{aligned} L_{st} = L_t(E_a, E_p, E_{tts}) + L_t(E_a, E_p, E_{vc}) \\ + \Sigma_{j=0}^{N} L_t(E_a, E_p, E_{spk_j}) i \neq j \end{aligned}$$

(4.6)

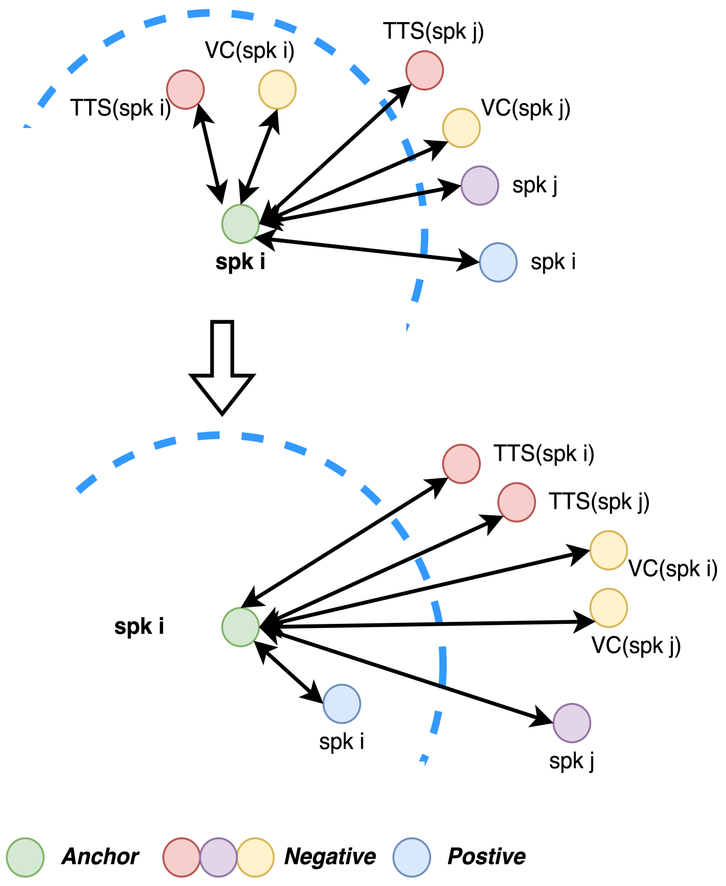


Figure 4.5: Training Based on Spoof-source Based Triplet Loss.

4.3.5 Overall Loss Function

As shown in Figure 4.3, the overall loss for SA-SASV is determined by all of its constituent decoders, which includes five different loss functions, as shown in Equation 4.7.

$$L_{sasasv} = L_{cm} + \lambda_1 L_{asv} + \lambda_2 L_{tts} + \lambda_3 L_{vc} + \lambda_4 L_{ts} \tag{4.7}$$

4.4 Analysis of Experimental Results

This section analyzes the results of the experiments we conducted. We analyze our model performance with an ablation study and compare the obtained results with prior research in the SASV problem.

4.4.1 Experiment Setting

Table 4.2: Comparison on characteristics and performance of different SASV systems.

Models		SASV Baseline1 [98]	SASV Baseline2 [98]	Cascaded CM/ASV [105]	2-stage PLDA [105, 102]
Inputs		raw waveforms, Fbanks	raw waveforms, Fbanks	MFCC STFT	MFCC
Encoders		ECAPA-TDNN, AASIST	ECAPA-TDNN, AASIST	LC-GRNN, X-Vector	X-Vector
Training	Phase1	ASV, CM systems	ASV, CM systems	ASV, CM systems	PLDA

	Phase2	\	concatenated embeddings	\	PLDA
Ensemble		Score	Embeddings	\	\
EER-SASV		19.15	8.76	7.67	28.40

Models		Triplet TDNN [105, 107]	INN(AUE) [105]	**SA-SASV**	
Inputs		MFCC, CQCC	MFCC, STFT	**raw waveforms, Fbanks**	
Encoders		TDNN	LC-GRNN, B-Vector	**ECAPA-TDNN ARawNet**	
Training	Phase1	TDNN	ASV, CM systems	**SA-SASV**	
	Phase2	PLDA (CM), PLDA (ASV)	concatenated embeddings	\	
Ensemble		Score	Embeddings	\	
EER-SASV		8.99	6.05	**4.86**	

Dataset. The SASV challenge permits the VoxCeleb2 dataset [111] and the ASVspoof 2019 LA dataset [71] for training the ASV and CM models. The VoxCeleb2 database consists of over 1 million utterances from 6,112 speakers and is designed for the ASV task, without spoofed data. The ASVspoof 2019 LA dataset, on the other hand, is prepared for the CM tasks, containing 6 types of spoof attacks in the training set and another 11 types of spoof attacks in the evaluation set, where the SASV models are tested. We use the VoxCeleb2 dataset to pre-train the ECAPA-TDNN and our model is fine-tuned on the ASVspoof 2019 LA dataset. As shown in Table 4.1, the models need to generalize training attacks (A1-A6) to unseen attacks (A7-A19).

Metrics. We evaluated our model performance based on the SASV-EER, which is the primary metric in the SASV challenge. As shown in Table 4.3, only target speakers are labeled as positive and both non-target bonafide and spoof attacks are labeled as negative in the SASV-EER. The SV-EER and SPF-EER, are complements to SASV-EER, and reflect models' capability in different subsets of the full trials. Compared to the EER used in the ASVSpoof challenge, the SPF-EER only tests model performance in trials based on bonafide target speakers with spoofed speech.

	Target	Non-target	Spoof
SV-EER	+	-	
SPF-EER	+		-
SASV-EER	+	-	-

Table 4.3: Metrics to evaluate SASV systems.

Baseline. The SASV challenge provides two baseline models using state-of-the-art ASV and CM systems with different fusion strategies. **Baseline1** adopts a score-sum ensemble, which uses a naive sum function to integrate non-calibrated scores from the ASV and CM systems. While this method does not consider the difference between scores from different systems, scores of ASV systems are cosine similarity and scores of CM systems are from classifiers. **Baseline2** uses an extra network as a fusion strategy that takes embeddings from pre-trained ASV and CM systems to produce the final scores.

Configuration		SASV	SV	SPF
ECAPA-TDNN		22.38	0.83	29.32
SASV-Baseline1		19.15	35.1	0.5
SASV-Baseline2		8.75	16.01	12.23
Ours	SA-SASV	**4.86**	8.06	**0.50**
	w/o triplet	5.82	9.14	2.12
	w/o spoof aggregator	5.90	9.96	0.68
	naive multi-task classifier	5.58	9.05	0.83

Table 4.4: Ablation study on the AS-SASV system.

4.4.2 Results Discussion

4.4.2.1 Ablation Study on the Proposed Model

Configuration. An ablation study was conducted to investigate the effects of the different components on the performance of the SA-SASV system. As shown in Table 4.4, we evaluated our model with varying configurations of (1) just spoof source-based triplet loss, (2) just spoof aggregator, (3) and the two combined. Results indicate the absence of either component will reduce the SASV-EER of the SA-SASV model and configurations with all proposed sub-structures provide the best results on the ASVSpoof 2019 Dataset. By comparing SPF-EER, we can find spoof-source-based triplet loss boosts model performance in the countermeasure task in multi-task classification model. Our best results improve all three metrics and the SASV-ERR was improved from 8.75% (baseline) to 4.86%.

The proposed model shows different generalization capabilities in SV and SPF tasks. Even though the SV-EER of the model reaches 0 in the training stage, it has limited ability to generalize the SV task to the evaluation set, which only contains unseen speakers. As a result, the overall model performance drastically decreased due to SV-EER. We also noticed that, due to the overfitting problem, compared to SPF-EER, SV-EER in all models with different configurations tends to have unstable results. However, the SPF-EER of the model shows consistency from training to evaluation set, the best SPF-EER reaches 0.5, which is better than the baseline single CM system.

In conclusion, the model can detect unseen spoof attacks and has trouble distinguishing unknown speakers in the evaluation set. We conjecture the performance difference stems from data distribution in the training set. Only 40 speakers are contained in the training set and the ASV task usually requires a larger number of speakers to build features of human utterance, e.g., 5,994 speakers are included in the VoxCeleb2 dataset.

Although parts of our encoder are pre-trained on the VoxCeleb2 dataset, it only gave our model a feasible initializing strategy. During the training stage, the bonafide cluster in our new feature space is highly overfitted. The results of SPF-EER and SV-EER therefore show a different tendency in the training and evaluation stages. We believe it is a reasonable solution to train end-to-end SASV systems on complete ASV and CM datasets to avoid the overfitting problem.

Visualization. To observe the updates of the features space produced by our encoder, we visualized utterances in the evaluation set using the t-SNE, as shown in Figure 4.6. The

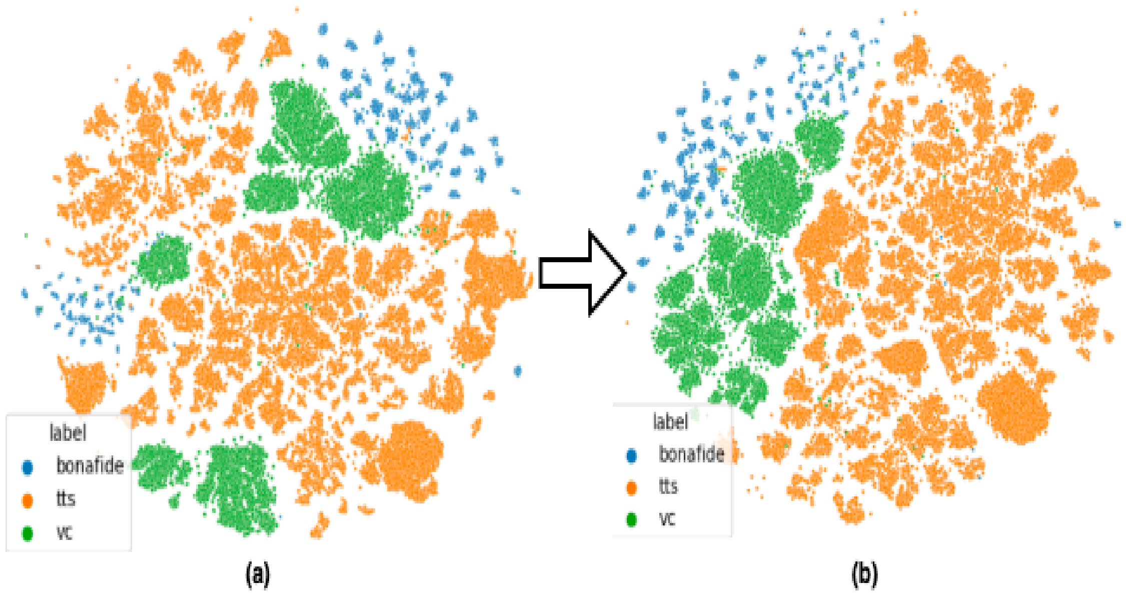


Figure 4.6: Visualization of the Feature Space in SA-SASV Using t-SNE.

left side (labeled (a)) shows the distribution of samples from the naive multi-task classifier without spoof-source-based triplet loss and the spoof aggregator. The right side (labeled (b)) shows the updated distribution using SA-SASV on the evaluation set. Compared to naive the multi-task classifier, both spoof attacks from TTS and VC tend to have denser clustering and cleaner boundaries, making TTS, VC, and bonafide easier to differentiate.

4.4.2.2 Model Comparison with other SASV systems

We compared the characteristics and performance on the ASVSpoof 2019 LA dataset of SA-SASV with other SASV systems as shown in Table 4.2. Compared to other ensemble-based systems, SA-SASV takes advantage of a single training phase, intending to build a single representation in the feature space for utterances from different sources.Our SA-SASV improves SASV-EER from 6.05% (the prior best-performed INN(AUE) system) to 4.86%.

4.5 Concluding Remarks

We proposed an end-to-end SA-SASV model, which is optimized with multiple loss functions to aggregate TTS, VC, and different speakers separately. Results show that the feature space of SA-SASV is better able to distinguish spoof attacks and identify speakers versus prior published approaches. Further, the SASV-EER is improved from the 6.05% produced by prior state of the art approaches to 4.86% without an ensembling strategy. A larger dataset and different encoders would likely boost the performance of the SV-EER and we will explore this in future work.

Chapter 5

Conclusion

Deep learning research has transitioned from a research context to an industry environment, and deep learning applications have become ubiquitous. As one of the in-the-wild challenges, frugality pursues a balance between costs and performance, reaching better model predictions with fewer costs. This thesis research formulates frugality challenges by discussing different development stages of deep learning applications and identifies them as three key research challenges. 1) Solving new in-the-wild problems with deep learning requires expensive costs for data generation, especially for domain-specific applications. 2) In pursuit of better performance, deep learning applications tend to have more complex structures or ensemble models, which require higher computational costs. 3) Deep learning applications tend to become a multi-model structure to solve new challenges when problems become more complex. A summary of our research contributions are discussed below.

5.1 Summary of the Research Contributions

- An scalable dataset generation solution for Sketch2Vis problems.

    1. Propose and formulate the Sketch2Vis challenges, translating hand-drawn sketches into visualization code.
    2. Present a novel approach for combining a Domain-Specific Language (DSL) and style transfer to generate training data for the Sketch2Vis challenge automatically.
    3. Validate that models trained on synthetically generated data can effectively generalize to hand-drawn sketches.

4. Compare and evaluate the performance of the recurrent neural network (RNN)-based and Transformer-based networks on the Sketch2Vis problem, built the baseline model of Sketch2Vis with a transformer-based approach.
5. Discuss challenges in existing evaluation metrics and propose new evaluation metrics to score generated visualizations.
6. Conclude the proposed solution as a reverse engineering method, which can be generalized to dataset generation for other programming language generation problems.

- A solution for reducing the computation complexity of ensemble models in the ASVSpoof problem.

1. Propose assumptions to concatenate raw waveforms and handcrafted features in a single model to reach less model complexity.
2. Introduce the Auxiliary Rawnet model architecture to use the raw waveform and handcrafted features in the ASVSpoof problem with a lightweight encoder to process raw waveform.
3. Improve models' resistance towards 10 of 13 spoof attacks in ASVSpoof 2019 challenges.
4. The model performance is similar to Stage-of-the-art models with minor computation complexity.
5. Discuss the potentiality of combining a lightweight waveform encoder with multiple encoders in other speech problems with unexpected inputs.

- A model structure to simplify system complexity for the SASV problem.

1. Discuss a generalization solution to fuse multiple models with the same type of inputs in multi-module problems.

2. Propose a fully trainable end-to-end model structure, SA-SASV, to aggregate spoofing attacks with multiple loss functions.
3. Show the feasibility of simplifying the SASV problem into a single training stage.

5.2 Future Work

5.2.1 Automatic data generation and the Sketch2Vis Challenge

Our future work focuses on optimizing and extending our current dataset by integrating more visualization tools to enlarge sketch styles for the Sketch2Vis model. In our current work, we did not focus on textual processing information (such as variable names) and still use a manual mapping step to map these textual names to data series. In future work, we intend to work on automatically translating axis and series labels into data queries. We are also refining feature selections in our DSL model by investigating human demands and problems during the visualization process from online communities (such as StackOverflow) to enhance our model's usability by enlarging its supported features without significantly expanding the DSL vocabulary size. Finally, we are preparing more hand-drawn sketches for validation.

5.2.2 Lightweight ensemble solution and the ASVSpoof Challenge

Evaluate proposed model to similar speech-related challenges. In our research, we implemented ARawNet to validate our assumption on combining raw waveform and handcrafted features. The model shows excellent performance on the ASVSpoof problem in distinguishing spoofing attacks and bonafide speeches, reaching the state-of-the-art when we publish our work. However, we need more experiments and model implementation in the context of different problems to validate our assumptions further.

Investigate relationship between model performance and different spoof types. Even though the overall performance (pooled-EER) of the ARawNet model is excellent, there are unexplainable parts in our performance evaluations by spoof category. We expected our model to enhance the performance of the primary encoder in all spoofing attacks. In contrast, the performance of the complementary models is still inadequate in three types of attacks. In the future, we must investigate our model performance based on different acoustic models.

5.2.3 Simplified multi-module structure and the SASV Challenge

The overfitting problem limits experimental results in our research and a potential simplification in other multi-module problems. A large, better-balanced dataset remains a challenge for our SA-SASV models. End-to-end SASV systems need to be trained on complete ASV and CM datasets to avoid the overfitting problem. In the future, we need to explore how to generate a dataset containing an appropriate ratio of bonafide speakers and spoofing attacks.

5.3 Summary of Publications

1. Zhongwei Teng, Quchen Fu, Jules White, Maria Powell, and Douglas C. Schmidt. Sa-sasv: An end-to-end spoof-aggregated spoofing-aware speaker verification system. In Interspeech 2022-23rd Annual Conference of the International Speech Communication Association. ISCA, 2022

2. Quchen Fu, Zhongwei Teng, Jules White, Maria E Powell, and Douglas C Schmidt. Fastaudio: A learnable audio front-end for spoof speech detection. In ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 3693–3697. IEEE, 2022

3. Zhongwei Teng, Quchen Fu, Jules White, Maria Powell, and Douglas C. Schmidt. Arawnet: A lightweight solution for leveraging raw waveforms in spoof speech de-

tection. In IEEE ICPR 2022 26TH International Conference on Pattern Recognition (ICPR). IEEE, 2022

4. Zhongwei Teng, Fu Quchen, White Jules, and Douglas C Schmidt. Sketch2vis: Generating data visualizations from hand-drawn sketches with deep learning. In 2021 20th IEEE International Conference on Machine Learning and Applications(ICMLA). IEEE, 2021

5. Zhongwei Teng, Peng Zhang, Xiao Li, William Nock, Denis Gilmore, Marcelino Rodriguez-Cancio, Jules White, Jonathan Carl Nesbitt, and Douglas Craig Schmidt. Authentication & integration approaches for mhealth apps from a usability view. Advances in Electrical and Electronic Engineering, 19(1):74–89, 2021

6. Zhongwei Teng, Peng Zhang, Xiao Li, William Nock, Marcelino Rodriguez-Cancio, Jules White, Douglas C Schmidt, Denis Gilmore, and Jonathan C Nesbitt. Authentication and usability in mhealth apps. In 2018 IEEE 20th International Conference on e-Health Networking, Applications and Services (Healthcom), pages 1–6. IEEE, 2018

7. Zhongwei Teng, William Nock, and White Jules. Checklist usage secure software development. In 10th International Conference on Software Engineering and Applications (SEAPP 2021), 2021

8. Yao Pan, Fangzhou Sun, Zhongwei Teng, Jules White, Douglas C Schmidt, Jacob Staples, and Lee Krause. Detecting web attacks with end-to-end deep learning. Journal of Internet Services and Applications, 10(1):1–22, 2019

9. Mayank Agarwal, Tathagata Chakraborti, Quchen Fu, David Gros, Xi Victoria Lin, Jaron Maene, Kartik Tala-madupula, Zhongwei Teng, and Jules White. Neurips 2020 nlc2cmd competition: Translating natural language to bash commands. In PMLR Volume 133: NeurIPS 2020 Competition and Demonstration Track, 2021

10. Fu Quchen, Zhongwei Teng, White Jules, and Douglas Schmidt. A transformer-based approach for translating natural language to bash commands. In 2021 20th IEEE International Conference on Machine Learning and Applications(ICMLA). IEEE, 2021

11. Quchen Fu, Ramesh Chukka, Keith Achorn, Thomas Atta-fosu, Deepak R Canchi, Zhongwei Teng, Jules White, and Douglas C Schmidt. Deep learning models on cpus: A methodology for efficient training. arXiv preprint arXiv:2206.10034, 2022

12. Quchen Fu, Zhongwei Teng, Marco Georgaklis, Jules White and Douglas C. Schmidt. NL2CMD: An Updated Workflow for Natural Language to Bash Commands Translation. Journal of Machine Learning Theory, Applications and Practice (Accepted)

BIBLIOGRAPHY

[1] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models. *arXiv preprint arXiv:2112.10741*, 2021.

[2] Michael Groves and Klaus Mundt. Friend or foe? google translate in language for academic purposes. *English for Specific Purposes*, 37:112–121, 2015.

[3] Veton Këpuska and Gamal Bohouta. Comparing speech recognition systems (microsoft api, google api and cmu sphinx). *Int. J. Eng. Res. Appl*, 7(03):20–24, 2017.

[4] Yong Bai and Yinggang Chen. Human motion analysis and action scoring technology for sports training based on computer vision features. *Journal of Intelligent & Fuzzy Systems*, (Preprint):1–9, 2021.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[7] Thilo Stadelmann, Mohammadreza Amirian, Ismail Arabaci, Marek Arnold, Gilbert François Duivesteijn, Ismail Elezi, Melanie Geiger, Stefan Lörwald, Benjamin Bruno Meier, Katharina Rombach, et al. Deep learning in the wild. In *IAPR Workshop on Artificial Neural Networks in Pattern Recognition*, pages 17–38. Springer, 2018.

[8] Andrew Ng. Machine learning yearning. *URL: http://www. mlyearning. org/(96)*, 139, 2017.

[9] Carlos Perez. *The deep learning AI playbook.* Lulu. com, 2017.

[10] Wojciech Samek, Thomas Wiegand, and Klaus-Robert Müller. Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. *arXiv preprint arXiv:1708.08296*, 2017.

[11] Jaegul Choo and Shixia Liu. Visual analytics for explainable deep learning. *IEEE computer graphics and applications*, 38(4):84–92, 2018.

[12] Sushil Kumar Singh, Young-Sik Jeong, and Jong Hyuk Park. A deep learning-based iot-oriented infrastructure for secure smart city. *Sustainable Cities and Society*, 60:102252, 2020.

[13] Saba Amiri, Sara Salimzadeh, and Adam SZ Belloum. A survey of scalable deep learning frameworks. In *2019 15th International Conference on eScience (eScience)*, pages 650–651. IEEE, 2019.

[14] Eunkyeong Kim, Jinyong Kim, Hansoo Lee, and Sungshin Kim. Adaptive data augmentation to achieve noise robustness and overcome data deficiency for deep learning. *Applied Sciences*, 11(12):5586, 2021.

[15] Erdenebayar Urtnasan, Jung Hun Lee, Byungjin Moon, Hee Young Lee, Kyuhee Lee, Hyun Youk, et al. Noninvasive screening tool for hyperkalemia using a single-lead electrocardiogram and deep learning: Development and usability study. *JMIR Medical Informatics*, 10(6):e34724, 2022.

[16] Mikhail Evchenko, Joaquin Vanschoren, Holger H Hoos, Marc Schoenauer, and Michèle Sebag. Frugal machine learning. *arXiv preprint arXiv:2111.03731*, 2021.

[17] Michael Friendly. A brief history of data visualization. In *Handbook of data visualization*, pages 15–56. Springer, 2008.

[18] Pengcheng Yin and Graham Neubig. A syntactic neural model for general-purpose code generation. *arXiv preprint arXiv:1704.01696*, 2017.

[19] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D^3 data-driven documents. *IEEE transactions on visualization and computer graphics*, 17(12):2301–2309, 2011.

[20] tableau. Tableau. https://www.tableau.com/, 2019.

[21] Michael Diamond and Angela Mattia. Data visualization: An exploratory study into the software tools used by businesses. *Journal of Instructional Pedagogies*, 18, 2017.

[22] Jagoda Walny, Christian Frisson, Mieka West, Doris Kosminsky, Søren Knudsen, Sheelagh Carpendale, and Wesley Willett. Data changes everything: Challenges and opportunities in data visualization design handoff. *IEEE transactions on visualization and computer graphics*, 26(1):12–22, 2019.

[23] Arjun Srinivasan, Bongshin Lee, Nathalie Henry Riche, Steven M Drucker, and Ken Hinckley. Inchorus: Designing consistent multimodal interactions for data visualization on tablet devices. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2020.

[24] Xiaojun Xu, Chang Liu, and Dawn Song. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*, 2017.

[25] Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. Sqlizer: query synthesis from natural language. *Proceedings of the ACM on Programming Languages*, 1(OOPSLA):1–26, 2017.

[26] Prasetya Utama, Nathaniel Weir, Fuat Basik, Carsten Binnig, Ugur Çetintemel, Benjamin Hättasch, Amir Ilkhechi, Shekar Ramaswamy, and Arif Usta. An end-to-end neural natural language interface for databases. *arXiv preprint arXiv:1804.00401*, 2018.

[27] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.

[28] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[29] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, et al. The open images dataset v4. *International Journal of Computer Vision*, pages 1–26, 2020.

[30] Peter Anderson, Basura Fernando, Mark Johnson, and Stephen Gould. Spice: Semantic propositional image caption evaluation. In *European conference on computer vision*, pages 382–398. Springer, 2016.

[31] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

[32] Michael Denkowski and Alon Lavie. Meteor universal: Language specific translation evaluation for any target language. In *Proceedings of the ninth workshop on statistical machine translation*, pages 376–380, 2014.

[33] Tony Beltramelli. pix2code: Generating code from a graphical user interface screen-

shot. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, page 3. ACM, 2018.

[34] Mengtian Li, Zhe Lin, Radomír Mˇ ech, , Ersin Yumer, and Deva Ramanan. Photo-sketching: Inferring contour drawings from images. *WACV*, 2019.

[35] Jared Wilber. roughviz.

[36] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[37] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: Lessons learned from the 2015 mscoco image captioning challenge. *IEEE transactions on pattern analysis and machine intelligence*, 39(4):652–663, 2016.

[38] Xu Jia, Efstratios Gavves, Basura Fernando, and Tinne Tuytelaars. Guiding the long-short term memory model for image caption generation. In *Proceedings of the IEEE international conference on computer vision*, pages 2407–2415, 2015.

[39] Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. Bottom-up and top-down attention for image captioning and visual question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6077–6086, 2018.

[40] Xinxin Zhu, Lixiang Li, Jing Liu, Haipeng Peng, and Xinxin Niu. Captioning transformer with stacked attention modules. *Applied Sciences*, 8(5):739, 2018.

[41] Jun Yu, Jing Li, Zhou Yu, and Qingming Huang. Multimodal transformer with multi-view visual representation for image captioning. *IEEE transactions on circuits and systems for video technology*, 30(12):4467–4480, 2019.

[42] Patsorn Sangkloy, Nathan Burnell, Cusuh Ham, and James Hays. The sketchy

database: learning to retrieve badly drawn bunnies. *ACM Transactions on Graphics (TOG)*, 35(4):1–12, 2016.

[43] Yuntian Deng, Anssi Kanervisto, and Alexander M Rush. What you get is what you see: A visual markup decompiler. *arXiv preprint arXiv:1609.04938*, 10:32–37, 2016.

[44] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

[45] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander Rush. OpenNMT: Open-source toolkit for neural machine translation. In *Proceedings of ACL 2017, System Demonstrations*, pages 67–72, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[46] Yuntian Deng, Anssi Kanervisto, Jeffrey Ling, and Alexander M Rush. Image-to-markup generation with coarse-to-fine attention. In *International Conference on Machine Learning*, pages 980–989. PMLR, 2017.

[47] Christoph Stumpf Martin Krasser. Image captioning transformer. https://github.com/krasserm/fairseq-image-captionin, 2020.

[48] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[49] Bowen Yu and Cláudio T Silva. Flowsense: A natural language interface for visual data exploration within a dataflow system. *IEEE transactions on visualization and computer graphics*, 26(1):1–11, 2019.

[50] Arpit Narechania, Arjun Srinivasan, and John Stasko. Nl4dv: A toolkit for generating analytic specifications for data visualization from natural language queries. *IEEE Transactions on Visualization and Computer Graphics*, 2020.

[51] Victor Dibia and Çağatay Demiralp. Data2vis: Automatic generation of data visualizations using sequence-to-sequence recurrent neural networks. *IEEE computer graphics and applications*, 39(5):33–46, 2019.

[52] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. Vega-lite: A grammar of interactive graphics. *IEEE transactions on visualization and computer graphics*, 23(1):341–350, 2016.

[53] Xuedi Qin, Yuyu Luo, Nan Tang, and Guoliang Li. Making data visualization more efficient and effective: a survey. *The VLDB Journal*, 29(1):93–117, 2020.

[54] Levent Burak Kara and Thomas F Stahovich. Hierarchical parsing and recognition of hand-sketched diagrams. In *Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 13–22, 2004.

[55] Manuel J Fonseca, César Pimentel, and Joaquim A Jorge. Cali: An online scribble recognizer for calligraphic interfaces. In *AAAI spring symposium on sketch understanding*, pages 51–58, 2002.

[56] James A Landay and Brad A Myers. Sketching interfaces: Toward more human interface design. *Computer*, 34(3):56–64, 2001.

[57] Brandon Paulson and Tracy Hammond. Paleosketch: accurate primitive sketch recognition and beautification. In *Proceedings of the 13th international conference on Intelligent user interfaces*, pages 1–10, 2008.

[58] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[59] Shinji Takaki and Junichi Yamagishi. A deep auto-encoder based low-dimensional feature extraction from fft spectral envelopes for statistical parametric speech synthesis. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5535–5539. IEEE, 2016.

[60] MD Zakir Hossain, Ferdous Sohel, Mohd Fairuz Shiratuddin, and Hamid Laga. A comprehensive survey of deep learning for image captioning. *ACM Computing Surveys (CsUR)*, 51(6):1–36, 2019.

[61] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[62] Sen Chen, Lingling Fan, Ting Su, Lei Ma, Yang Liu, and Lihua Xu. Automated cross-platform gui code generation for mobile apps. In *2019 IEEE 1st International Workshop on Artificial Intelligence for Mobile (AI4Mobile)*, pages 13–16. IEEE, 2019.

[63] Gustav Fechner. Elements of psychophysics. vol. i. 1966.

[64] Joakim Andén and Stéphane Mallat. Deep scattering spectrum. *IEEE Transactions on Signal Processing*, 62(16):4114–4128, 2014.

[65] Neil Zeghidour, Olivier Teboul, Félix de Chaumont Quitry, and Marco Tagliasacchi. Leaf: A learnable frontend for audio classification. *arXiv preprint arXiv:2101.08596*, 2021.

[66] Jee-Weon Jung, Hee-Soo Heo, IL-Ho Yang, Hye-Jin Shim, and Ha-Jin Yu. Avoiding speaker overfitting in end-to-end dnns using raw waveform for text-independent speaker verification. *extraction*, 8(12):23–24, 2018.

[67] Jee-weon Jung, Hee-Soo Heo, Ju-ho Kim, Hye-jin Shim, and Ha-Jin Yu. Rawnet: Advanced end-to-end deep neural network using raw waveforms for text-independent speaker verification. *arXiv preprint arXiv:1904.08104*, 2019.

[68] Andy T Liu, Shang-Wen Li, and Hung-yi Lee. Tera: Self-supervised learning of transformer encoder representation for speech. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 2021.

[69] Hemlata Tak, Jose Patino, Massimiliano Todisco, Andreas Nautsch, Nicholas Evans, and Anthony Larcher. End-to-end anti-spoofing with rawnet2. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6369–6373. IEEE, 2021.

[70] Xu Li, Na Li, Chao Weng, Xunying Liu, Dan Su, Dong Yu, and Helen Meng. Replay and synthetic speech detection with res2net architecture. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6354–6358. IEEE, 2021.

[71] Massimiliano Todisco, Xin Wang, Ville Vestman, Md Sahidullah, Héctor Delgado, Andreas Nautsch, Junichi Yamagishi, Nicholas Evans, Tomi Kinnunen, and Kong Aik Lee. Asvspoof 2019: Future horizons in spoofed and fake audio detection. *arXiv preprint arXiv:1904.05441*, 2019.

[72] Steffen Schneider, Alexei Baevski, Ronan Collobert, and Michael Auli. wav2vec: Unsupervised pre-training for speech recognition. *arXiv preprint arXiv:1904.05862*, 2019.

[73] Mirco Ravanelli and Yoshua Bengio. Speaker recognition from raw waveform with sincnet. In *2018 IEEE Spoken Language Technology Workshop (SLT)*, pages 1021–1028. IEEE, 2018.

[74] Dimitri Palaz, Mathew Magimai Doss, and Ronan Collobert. Convolutional neural networks-based continuous speech recognition using raw speech signal. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4295–4299. IEEE, 2015.

[75] Paul-Gauthier Noé, Titouan Parcollet, and Mohamed Morchid. Cgcnn: Complex gabor convolutional neural network on raw speech. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7724–7728. IEEE, 2020.

[76] Randall Balestriero, Romain Cosentino, Hervé Glotin, and Richard Baraniuk. Spline filters for end-to-end deep learning. In *International conference on machine learning*, pages 364–373. PMLR, 2018.

[77] Quchen Fu, Zhongwei Teng, Jules White, Maria E Powell, and Douglas C Schmidt. Fastaudio: A learnable audio front-end for spoof speech detection. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3693–3697. IEEE, 2022.

[78] Kaizhi Qian, Yang Zhang, Shiyu Chang, Mark Hasegawa-Johnson, and David Cox. Unsupervised speech decomposition via triple information bottleneck. In *International Conference on Machine Learning*, pages 7836–7846. PMLR, 2020.

[79] Tomi Kinnunen, Kong Aik Lee, Héctor Delgado, Nicholas Evans, Massimiliano Todisco, Md Sahidullah, Junichi Yamagishi, and Douglas A Reynolds. t-dcf: a detection cost function for the tandem assessment of spoofing countermeasures and automatic speaker verification. *arXiv preprint arXiv:1804.09618*, 2018.

[80] Christian Schörkhuber and Anssi Klapuri. Constant-q transform toolbox for music processing. In *7th sound and music computing conference, Barcelona, Spain*, pages 3–64, 2010.

[81] David Snyder, Daniel Garcia-Romero, Gregory Sell, Daniel Povey, and Sanjeev Khudanpur. X-vectors: Robust dnn embeddings for speaker recognition. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5329–5333. IEEE, 2018.

[82] Mirco Ravanelli, Titouan Parcollet, Peter Plantinga, Aku Rouhe, Samuele Cornell, Loren Lugosch, Cem Subakan, Nauman Dawalatabad, Abdelwahab Heba, Jianyuan Zhong, Ju-Chieh Chou, Sung-Lin Yeh, Szu-Wei Fu, Chien-Feng Liao, Elena Rastorgueva, François Grondin, William Aris, Hwidong Na, Yan Gao, Renato De Mori, and Yoshua Bengio. SpeechBrain: A general-purpose speech toolkit, 2021. arXiv:2106.04624.

[83] Brecht Desplanques, Jenthe Thienpondt, and Kris Demuynck. Ecapa-tdnn: Emphasized channel attention, propagation and aggregation in tdnn based speaker verification. *arXiv preprint arXiv:2005.07143*, 2020.

[84] Ye Jia, Yu Zhang, Ron J Weiss, Quan Wang, Jonathan Shen, Fei Ren, Zhifeng Chen, Patrick Nguyen, Ruoming Pang, Ignacio Lopez Moreno, et al. Transfer learning from speaker verification to multispeaker text-to-speech synthesis. *arXiv preprint arXiv:1806.04558*, 2018.

[85] Yujia Li, Kevin Swersky, and Rich Zemel. Generative moment matching networks. In *International Conference on Machine Learning*, pages 1718–1727. PMLR, 2015.

[86] Tomi Kinnunen, Lauri Juvela, Paavo Alku, and Junichi Yamagishi. Non-parallel voice conversion using i-vector plda: Towards unifying speaker verification and transformation. In *2017 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5535–5539. IEEE, 2017.

[87] Chin-Cheng Hsu, Hsin-Te Hwang, Yi-Chiao Wu, Yu Tsao, and Hsin-Min Wang.

Voice conversion from unaligned corpora using variational autoencoding wasserstein generative adversarial networks. *arXiv preprint arXiv:1704.00849*, 2017.

[88] Najim Dehak, Patrick J Kenny, Réda Dehak, Pierre Dumouchel, and Pierre Ouellet. Front-end factor analysis for speaker verification. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(4):788–798, 2010.

[89] Georg Heigold, Ignacio Moreno, Samy Bengio, and Noam Shazeer. End-to-end text-dependent speaker verification. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5115–5119. IEEE, 2016.

[90] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[91] Masanori Morise, Fumiya Yokomori, and Kenji Ozawa. World: a vocoder-based high-quality speech synthesis system for real-time applications. *IEICE TRANSACTIONS on Information and Systems*, 99(7):1877–1884, 2016.

[92] Marc Schröder, Marcela Charfuelan, Sathish Pammi, and Ingmar Steiner. Open source voice creation toolkit for the mary tts platform. In *12th Annual Conference of the International Speech Communication Association-Interspeech 2011*, pages 3253–3256. ISCA, 2011.

[93] Driss Matrouf, J-F Bonastre, and Corinne Fredouille. Effect of speech transformation on impostor acceptance. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, volume 1, pages I–I. IEEE, 2006.

[94] Zhizheng Wu, Tomi Kinnunen, Nicholas Evans, Junichi Yamagishi, Cemal Hanilçi, Md Sahidullah, and Aleksandr Sizov. Asvspoof 2015: the first automatic speaker verification spoofing and countermeasures challenge. In *Sixteenth annual conference of the international speech communication association*, 2015.

[95] Tomi Kinnunen, Md Sahidullah, Héctor Delgado, Massimiliano Todisco, Nicholas Evans, Junichi Yamagishi, and Kong Aik Lee. The asvspoof 2017 challenge: Assessing the limits of replay spoofing attack detection. 2017.

[96] Junichi Yamagishi, Xin Wang, Massimiliano Todisco, Md Sahidullah, Jose Patino, Andreas Nautsch, Xuechen Liu, Kong Aik Lee, Tomi Kinnunen, Nicholas Evans, et al. Asvspoof 2021: accelerating progress in spoofed and deepfake speech detection. *arXiv preprint arXiv:2109.00537*, 2021.

[97] Tomi Kinnunen, Héctor Delgado, Nicholas Evans, Kong Aik Lee, Ville Vestman, Andreas Nautsch, Massimiliano Todisco, Xin Wang, Md Sahidullah, Junichi Yamagishi, et al. Tandem assessment of spoofing countermeasures and automatic speaker verification: Fundamentals. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28:2195–2210, 2020.

[98] Jee-weon Jung, Hemlata Tak, Hye-jin Shim, Hee-Soo Heo, Bong-Jin Lee, Soo-Whan Chung, Hong-Goo Kang, Ha-Jin Yu, Nicholas Evans, and Tomi Kinnunen. Sasv challenge 2022: A spoofing aware speaker verification challenge evaluation plan. *arXiv preprint arXiv:2201.10283*, 2022.

[99] Zhongwei Teng, Quchen Fu, Jules White, Maria Powell, and Douglas C Schmidt. Complementing handcrafted features with raw waveform using a light-weight auxiliary model. *arXiv preprint arXiv:2109.02773*, 2021.

[100] Yao Pan, Fangzhou Sun, Zhongwei Teng, Jules White, Douglas C Schmidt, Jacob Staples, and Lee Krause. Detecting web attacks with end-to-end deep learning. *Journal of Internet Services and Applications*, 10(1):1–22, 2019.

[101] Quchen Fu, Zhongwei Teng, Jules White, Maria Powell, and Douglas C Schmidt. Fastaudio: A learnable audio front-end for spoof speech detection. *arXiv preprint arXiv:2109.02774*, 2021.

[102] Aleksandr Sizov, Elie Khoury, Tomi Kinnunen, Zhizheng Wu, and Sébastien Marcel. Joint speaker verification and antispoofing in the *i*-vector space. *IEEE Transactions on Information Forensics and Security*, 10(4):821–832, 2015.

[103] Massimiliano Todisco, Héctor Delgado, Kong Aik Lee, Md Sahidullah, Nicholas Evans, Tomi Kinnunen, and Junichi Yamagishi. Integrated presentation attack detection and automatic speaker verification: Common features and gaussian back-end fusion. In *Interspeech 2018-19th Annual Conference of the International Speech Communication Association*. ISCA, 2018.

[104] Hye-jin Shim, Jee-weon Jung, Ju-ho Kim, and Ha-jin Yu. Integrated replay spoofing-aware text-independent speaker verification. *Applied Sciences*, 10(18):6292, 2020.

[105] Alejandro Gomez-Alanis, Jose A Gonzalez-Lopez, S Pavankumar Dubagunta, Antonio M Peinado, and Mathew Magimai Doss. On joint optimization of automatic speaker verification and anti-spoofing in the embedding space. *IEEE Transactions on Information Forensics and Security*, 16:1579–1593, 2020.

[106] Yuanjun Zhao, Roberto Togneri, and Victor Sreeram. Multi-task learning-based spoofing-robust automatic speaker verification system. *Circuits, Systems, and Signal Processing*, pages 1–22, 2022.

[107] Jiakang Li, Meng Sun, Xiongwei Zhang, and Yimin Wang. Joint decision of anti-spoofing and automatic speaker verification by multi-task learning with contrastive loss. *IEEE Access*, 8:7907–7915, 2020.

[108] Jee-weon Jung, Hee-Soo Heo, Hemlata Tak, Hye-jin Shim, Joon Son Chung, Bong-Jin Lee, Ha-Jin Yu, and Nicholas Evans. Aasist: Audio anti-spoofing using integrated spectro-temporal graph attention networks. *arXiv preprint arXiv:2110.01200*, 2021.

[109] Xu Xiang, Shuai Wang, Houjun Huang, Yanmin Qian, and Kai Yu. Margin matters: Towards more discriminative deep neural network embeddings for speaker recognition. In *2019 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 1652–1656. IEEE, 2019.

[110] Xin Wang, Junichi Yamagishi, Massimiliano Todisco, Héctor Delgado, Andreas Nautsch, Nicholas Evans, Md Sahidullah, Ville Vestman, Tomi Kinnunen, Kong Aik Lee, et al. Asvspoof 2019: A large-scale public database of synthesized, converted and replayed speech. *Computer Speech & Language*, 64:101114, 2020.

[111] Joon Son Chung, Arsha Nagrani, and Andrew Zisserman. Voxceleb2: Deep speaker recognition. *arXiv preprint arXiv:1806.05622*, 2018.

Appendix A

Sketch2Vis Dataset DSL Grammar Notebook

The Sketch2Vis adopts multiple synthetic mechanisms to generate hand-drawn-style data visualizations to overcome dataset challenges in training deep learning models. This appendix shows the complete DSL grammar of the current dataset used in our experiments with examples.

By adopting this idea, the Sketch2Vis dataset can be easily extended from the current configuration by implementing more complex visualization types and parameters.

The DSLs in our experiments are generated with the following sources

1. **Matplotlib**

```
<plot>
    <structure>
        <type> line </type>
        <marker> X </marker>
        <linestyle> -- </linestyle>
        <is_line_legends> False </is_line_legends>
    </structure>
</plot>
```

2. **roughViz**

```
<plot>
    <structure>
        <type> bar </type>
        <simplification> 0.5 </simplification>
        <hBar> False </hBar>
        <fillStyle> zigzag </fillStyle>
    </structure>
</plot>
```

3. **Photo-Sketching**

```
<plot>
    <structure>
        <type> box </type>
        <mono> True </mono>
        <vert> True </vert>
    </structure>
</plot>
```

Name	**Description**
<plot>	Starter of a Sketch2Vis DSL file
<structure>	Starter of a visualization instance, which works when there are multiple visualizations in one plot.
<type>	Plotting type of a visualization instance.

A.1 Tokens and Grammar

Matplotlib

Matplotlib is a popular Python 2D visualization library. We apply XKCD() function to generate hand-drawn style visualization.

The following Table shows available parameters in our current dataset. Unpredictable parameters are used to generate random visualization.

Parameter	**Description**	**Values**	**Apply**	**Predictable**
align	Alignment of the bars to the x coordinates	[’center’, ’edge’]	Bar	True
color	The colors of the bar/marker faces	[’b’,’g’,’r’, ’c’,’m’,’y’, ’k’]	Bar, Scatter	True
edgecolor	The colors of the edges.	[’b’,’g’,’r’, ’c’,’m’,’y’, ’k’,’w’, ’face’, ’none’]	Bar, Box	True
line_style	Style of plotted line	[’-’, ’–’, ’-.’, ’:’, ”]	Line	True
line_color	Color of plotted line	[’b’, ’g’, ’r’, ’c’, ’m’, ’y’, ’k’]	Line	True
line_marker	Style of plotted marker	[”.”, ”,”, ”o”, ”v”, ”^”, ”<”, ”>”, ”1”, ”2”, ”3”, ”4”, ”8”, ”s”, ”p”, ”P”, ”*”, ”h”, ”H”, ”+”, ”x”, ”X”, ”D”, ”d”, ””, ”_”, None]	Line, Scatter	True
islinelegends	Show legends in plots	[True, False]	Line	True

explode	Offsetting a pie slice	[True, False]	Pie	True
ring	Pie chart or donut chart	[True, False]	Pie	True
sketch	Draw pie chart without color	[True, False]	Pie	True
shadow	Draw a shadow beneath the pie	[True, False]	Pie	True
vert	Vertical boxes or horizontal boxes	[True, False]	Box	True
title	Title of visualization	random text	Bar, Line, Box	False
x_label	Labels of x variables	random text	Bar, Line	False
y_label	Labels of y variables	random text	Bar, Line	False
line_legends	Text of legends in Line charts	random text	Line	False
notch	Notched box plot or rectangular boxplot	[True, False]	Box	False
showfliers	Show the outliers beyond the caps	[True, False]	Box	False

startangle	The angle by which the start of the pie is rotated, counterclockwise from the x-axis.	Number from 0 to 90	Pie	False
fontSize	Size of font	Number of font size	Bar, Line, Box, Pie, Scatter	False
font	Handwriting-style font	String of font	Bar, Line, Box, Pie, Scatter	False
textPosition	Position of text	Position based on (x,y)	Bar, Line, Box, Pie, Scatter	False

RoughViz

roughViz.js is a JavaScript library to generate hand-drawn style visualizations.

The following table shows available parameters in our current dataset. Unpredictable parameters are used to generate random visualization.

Parameter	**Description**	**Values**	**Apply**	**Predictable**
simplification	Chart simplification	[0.2, 0.5, 0.85]	Bar, Line, Pie	True
hBar	Vertical bars or horizontal bars	[True, False]	Bar	True
fillStyle	Bar/Pie fill-style	[’hachure’, ’cross-hatch’, ’zigzag’, ’dashed’, ’solid’, ’zigzag-line’]	Bar, Pie	True
legend	Show legends in plots	[True, False]	Line, Pie	True
marker	Whether or not to add circles to chart.	[none, circle]	Line	True
legendPosition	Position of legends	[left, right]	Line, Pie	True
ring	Pie charts or donuts charts	[True, False]	Pie	True
color	Colors for markers	[’coral’, ’skyblue’, ’tan’, ’#8da0cb’, ’/’, ’tan’, ’orange’]	Scatter	True

point-radius	Radius of circles points	[2, 5, 8]	Scatter	True
xLabel	Labels of x axis	random text	Line, Scatter	False
yLabel	Labels of y axis	random text	Line, Scatter	False
title	Title of charts	random text	Line, Scatter, Bar, Pie	False
roughness	Roughness level of chart	Number from 1 to 10	Line, Scatter, Bar, Pie	False
axisRoughness	Roughness for x and y axes	Number from 1 to 10	Line, Scatter	False
circleRoughness	Roughness of circles	Number from 1 to 10	Line	False
fillWeight	Weight of inner paths’ color	Number from 0 to 1	Line, Scatter, Bar, Pie	False
font	Font-family to use	Handwriting Font	Line, Scatter, Bar, Pie	False

stroke	Color of lines' stroke	String of a color	Line	False

Photo-Sketching

Photo-Sketching is a style transfer deep learning model that we performed on sources images generated by Matplotlib to create simple monochromatic sketches.

Parameter	**Description**	**Values**	**Apply**	**Predictable**
mono	This is a monochromatic sketches	[True]	Scatter, Pie, Bar, Box, Line	True
ring	Pie charts or donut charts	[True, False]	Pie	True
align	Alignment of the bars to the x coordinates	[center, edge]	Bar	True
vert	Vertical boxes or horizontal boxes	[True, False]	Box	True
islinelegends	Show legends in plots	[True, False]	Line	True