FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Bc. Vladislav Stankov

Speech-Informed Inverse Text Normalization

Institute of Formal and Applied Linguistics

Supervisor of the master thesis:	doc. RNDr. Ondřej Bojar, Ph.D.
Study programme:	Computer Science
Study branch:	Artificial Intelligence

Prague 2024

I would like to thank my supervisor, doc. Ondřej Bojar, for his inspiration and invaluable guidance, feedback, and supervision throughout this project. His expertise in speech recognition and data analysis was crucial in helping me navigate and overcome all the challenges I encountered. I truly appreciate all the time he dedicated to guiding this work.

Further, I want to thank my wife, Natalia, who motivated me to complete this thesis. Thanks also to my friends and colleagues, especially Martin Bachura, Jakub Náplava, Jiří Bednář, Josef Vonášek, Petra Barančíková, Aleksandra Pereverzeva, Ekaterina Egorova, Jakub Střelský and Jan Bodnár for their humorous remarks about how long it took me to finish this thesis. Finally, I am deeply grateful to my parents for their valuable support throughout my master's degree.

Title: Speech-Informed Inverse Text Normalization

Author: Bc. Vladislav Stankov

Department: Institute of Formal and Applied Linguistics

Supervisor: doc. RNDr. Ondřej Bojar, Ph.D., Institute of Formal and Applied Linguistics

Abstract: In the domain of Automatic Speech Recognition (ASR), Inverse Text Normalization (ITN) is applied after the speech recognition step to transform recognized verbalized text into written form. This process includes converting verbalized numbers into digits, formatting dates and monetary amounts, and applying correct capitalization and inserting punctuation marks. As ITN systems serve as post-processing modules for ASR outputs, integrating the original audio input as an additional signal into the ITN system is also possible. In this thesis, we explore the impact of the speech signal on the performance of ITN neural models and create a dataset for training and evaluating speech-informed ITN models. Our best model demonstrates a significant improvement in the precision and recall of inserting periods, commas, and question marks, as well as in adding letter casing, when compared to the text-only baseline. Improvements are also observed in less frequent punctuation symbols, though they are not statistically significant.

Keywords: inverse text normalization, multimodality, automatic speech recognition, natural language processing, deep learning

Contents

1. Introduction

Text normalization is a crucial step in both natural language processing (NLP) and automatic speech recognition (ASR) systems. Text normalization converts text to a uniform format to facilitate better model performance. While the steps in NLP primarily involve the removal of case distinctions, punctuation removal, and lexical simplification through stemming or lemmatization, ASR faces unique challenges that require additional steps like converting numerals and dates into text, and standardizing abbreviations.

For the practical use of ASR systems, the reverse process called Inverse Text Normalization (ITN) is essential because it converts the machine-generated, verbalized speech transcription into a format resembling standard written text. This is important for ensuring that downstream NLP models can process the transcribed text without performance degradation. The key subtasks of ITN include converting numeric expressions to digits, standardizing date formats, and formatting other specialized entities like monetary amounts and addresses to their conventional written forms. Additionally, ITN may involve the removal of spoken disfluencies and even style adjustments to align with written language norms.

Speech-informed ITN represents an advancement in ITN systems by incorporating the original ASR audio input into the inverse normalization process. This approach utilizes acoustic cues from speech, such as intonation, pauses, and pitch, to enhance punctuation inference, sentence boundary detection, and the removal of disfluencies like self-corrections and repetitions. By integrating these audio cues, the resulting ASR-ITN chain can produce more accurate and contextually appropriate text transcriptions.

However, the development of neural speech-informed ITN systems is challenging, primarily due to the specificity of the required training and evaluation data. Consider the case of punctuation restoration with capitalization, for the traditional text-only ITN models, we can create training data from any written text by treating all capital letters and punctuation marks as labels to be restored from the normalized version of the same text. In contrast, speech-informed ITN models require training data that includes both text and corresponding audio inputs. This dual requirement complicates data collection, as most available datasets are text-only or specifically tailored for ASR with aligned sentence boundaries and possibly pre-normalized text (e.g., lowercased without punctuation). These constraints partly explain why text-only approaches remain more popular than their speech-informed counterparts.

The goal of this thesis is to investigate the impact of the speech signal for the ITN task, in terms of punctuation restoration and capitalization, by comparing it with text-only methods. An important prerequisite for our work will be to create a suitable multimodal dataset that includes unnormalized, natural text—text complete with casing and punctuation—along with corresponding speech recordings. This dataset will be derived from ParCzech 3.0 [Kopp et al., 2021], chosen for its extensive size in terms of hours and the availability of unnormalized transcriptions. Additionally, we will modify ParCzech 3.0 to better meet the requirements of the ITN task. During this modification we will train our ASR model to recognize all the audio files in ParCzech 3.0 and align the recognized transcripts with the original transcripts. After creating the dataset, we will develop a text-only ITN baseline model and perform experiments to integrate sound information. The key result of this thesis will be a multimodal cross-attention model component to fuse sound and text inputs. Moreover, we will evaluate various audio encoders to select the one that best fits the ITN task. Finally, we propose implementing multimodal positional embeddings that will result in soft attention alignment between text tokens and audio frames.

The structure of the thesis is as follows. Chapter 2 introduces the key concepts necessary for understanding the thesis and reviews recent research in the ITN field. In Chapter 3, we discuss ParCzech 3.0 and create a custom dataset for training and evaluating ITN neural models. Chapter 4 describes the training of an ASR model used to create the ITN-oriented dataset. In Chapter 5, we develop a text-only ITN baseline model and experiment with integrating audio information into the ITN neural model, concluding with an evaluation of all configurations using the previously created test set. Chapter 6 discusses the results and potential further improvements of our best multimodal ITN model. Finally in Appendix A, we provide a very brief summary of the code released with this thesis on Github.

2. Background

The following chapter outlines the key concepts relevant for the thesis.

2.1 Approaches to ITN

In a broader sense, inverse text normalization includes many subtasks: punctuation and case restoration, conversion of verbalized numerical expressions into their digit form ("twenty twenty-one" into "2021"), standardization of dates and times ("the first of January" becomes "January 1"), general conversion of special words to their symbolic equivalents ("section sign" into "§"), removal of disfluencies, and even style transfer. Because of its complexity the ITN task is often reduced to only part of its subtasks, e.g. punctuation and case restoration, or disfluency removal or style transfer.

An important aspect of ITN is the restoration of punctuation in textual inputs. Specifically, the PunKtuator work [Chordia, 2021] is dedicated to punctuation restoration task for high-resource (German, English, French) and low resource languages (Hindi and Tamil). PunKtuator's architecture combines Transformer's encoder part [Vaswani et al., 2017] with BiLSTM [Graves and Schmidhuber, 2005] and Neural Conditional Random Field components [Do and Artieres, 2010], further incorporating language classifiers and text mode classification (distinguishing between "Spoken" and "Written" text). This composite model effectively restores punctuation by treating the task as sequence labeling, jointly trained across multiple languages to reduce the dependency on monolingual data.

Another text-only work [Huang et al., 2021] uses multi-task objective to improve punctuation restoration by addressing data imbalance in punctuation classes. The paper proposes a novel method that utilizes token-level supervised contrastive learning (SCL) [Khosla et al., 2020], aiming to enhance the differentiation of punctuation marks in the latent space. The SCL loss is then combined with cross-entropy loss for the sequence labeling. This approach achieved a significant improvement in test set performance, demonstrating the effectiveness of supervised contrastive learning in handling imbalanced data for punctuation restoration tasks comparing to the models trained with cross-entropy loss only.

Additionally, punctuation restoration task can be efficiently coupled with case restoration task. Specifically, the paper "Robust Prediction of Punctuation and Truecasing for Medical ASR" [Sunkara et al., 2020b] describes usage of pre-trained masked language models like BERT [Devlin et al., 2019a] and RoBERTa [Liu et al., 2019] in medical domain and proposes a novel approach by jointly learning punctuation and capitalization as a sequence labeling problem. Moreover, to counteract the issues related to ASR errors, the authors propose a data augmentation approach utilizing n-best lists from ASR outputs, improving the model's ability to deal with speech recognition inaccuracies. The proposed model achieves approximately a 5% absolute improvement in F1 score on ground truth text and approximately a 10% improvement on ASR outputs over baseline models.

Moving forward to the multi-modal approaches, the paper "Attentional Parallel RNNs for Generating Punctuation in Transcribed Speech" [Öktem et al., 2017] introduces an innovative approach to generating punctuation in transcribed speech using recurrent neural networks (RNNs) [Rumelhart et al., 1986]. The paper proposes a novel method that incorporates both prosodic (e.g., pauses, speech rate, pitch) and lexical information through parallel processing with RNNs to predict punctuation marks in raw ASR output. By using an attention mechanism over parallel sequences of prosodic cues aligned with transcribed speech, the model can improve the accuracy of punctuation generation. The experiments show that including an attention mechanism that focuses on both prosodic and lexical inputs significantly improves the model's performance in punctuation restoration.

Another multi-modal approach is presented in the paper "Multimodal Punctuation Prediction with Contextual Dropout" [Silva et al., 2021]. The model is built upon a transformer architecture for text-based features, combined with a CNN [LeCun et al., 1989] for encoding spectrogram features from audio into a latent representation. An attention mechanism is used to fuse text and audio features before passing through fully-connected layers to predict the punctuation class. Additionally, a unique training scheme named contextual dropout is introduced to handle variable amounts of future context at test time, improving model performance with varying future information. As the result, the multimodal model, which incorporates both text and audio, demonstrated an 8% improvement over the text-only model on an internal dataset.

In the similar vein, the paper "Multimodal Semi-supervised Learning Framework for Punctuation Prediction in Conversational Speech" [Sunkara et al., 2020a] explores an approach for improving punctuation prediction in conversational speech through a multimodal semi-supervised learning method. The proposed model, referred to as MuSe, incorporates three primary components: an acoustic encoder, a lexical encoder, and a fusion block. The acoustic encoder processes audio signals to generate frame-level embeddings, while the lexical encoder deals with textual input to produce word-level embeddings. These embeddings are then combined using either forced alignment or an attention mechanism to predict punctuation marks accurately. The proposed semi-supervised learning architecture pre-trains lexical and acoustic encoders on extensive unpaired text and audio data, improving upon conventional supervised methods that rely solely on labeled data. Finally, to improve the model's robustness to ASR errors, the paper utilizes data augmentation techniques with n-best lists from ASR outputs. The MuSe model achieves approximately 6-9% absolute improvement in F1 score on reference transcripts and about 3-4% on ASR outputs over the baseline BiLSTM model on the Fisher corpus [Cieri et al., 2004].

2.2 Sequence Labeling

Sequence labeling is a fundamental task in NLP where the goal is to assign a label to each element in a sequence of tokens. This task is used in a variety of NLP applications, including part-of-speech tagging and named entity recognition. In essence, sequence labeling involves processing a sequence of words and annotating each word with a tag or category that describes its role or meaning within the context of the sentence.

The core challenge of sequence labeling lies in capturing the dependencies between tokens and labels, taking into account the fact that the appropriate label for a given token often depends on the labels of surrounding tokens. For instance, in named entity recognition, the identification of a multi-word entity requires understanding the relationship between consecutive words to classify them as part of the same entity.

Models designed for sequence labeling tasks typically generate a probability distribution over possible labels for each token in the input sequence. To train these models effectively, we need a loss function that can compare the predicted sequence of labels with the true sequence of labels and provide a measure of how well the model is performing. Cross-entropy loss is the most common measure used for these purposes.

2.2.1 Cross-Entropy Loss

Cross-entropy loss is a widely used loss function in classification tasks, including sequence labeling. It measures the dissimilarity between the true distribution of labels and the predicted distribution. For sequence labeling, cross-entropy loss is applied at each position in the input sequence, comparing the predicted probability distribution over labels for each token against the true label. The loss for a single token t is given by:

$$-\sum_{c=1}^{M} y_{t,c} \log(p_{t,c}) \tag{2.1}$$

where M is the number of labels, $y_{t,c}$ is a binary indicator of whether label c is the correct classification for token t, and $p_{t,c}$ is the predicted probability of token t being of class c.

The total loss for a sequence is then the sum of the losses for each token, normalized by the sequence length. This ensures that the model's performance is measured consistently across sequences, providing a clear objective for the model to minimize during training.

By minimizing the cross-entropy loss across all sequences in the training data, the model learns to predict the most likely label for each token, taking into account the context provided by the entire sequence. This process not only helps the model to correctly label individual tokens but also to learn the patterns and dependencies that define the structure of the sequence.

2.3 Automatic Speech Recognition

Automatic Speech Recognition (ASR) is a computational task that involves converting spoken language into text. The primary goal of ASR systems is to accurately and efficiently transcribe human speech, regardless of the speaker's accent, speech rate, or background noise, making it accessible for further processing or direct use. Modern ASR systems leverage deep learning models, which are trained on large datasets of spoken audio paired with corresponding transcriptions to learn the complex mappings from audio features to textual representations.

Training ASR models involves significant challenges due to the variability in human speech, including differences in pronunciation, dialects, and language features, as well as the presence of background noises and overlapping speech in real-world settings. Techniques such as unsupervised pre-training, where models learn from vast amounts of unlabelled audio data, followed by supervised fine-tuning on smaller, labeled datasets, have shown promising results in improving the robustness and accuracy of ASR systems. The main objective is to develop models that can perform with high accuracy across diverse conditions, enabling natural human-machine communication through speech.

2.3.1 Word Error Rate

Word Error Rate (WER) is a common metric in the evaluation of automatic speech recognition systems, providing a quantitative measure of the transcription accuracy of a given model. Essentially, WER compares the sequence of words produced by the ASR system against a reference transcription considered to be the ground truth. By doing so, it captures the performance of the system in terms of its ability to accurately recognize and transcribe spoken language.

The calculation of WER involves counting the number of substitutions (S), deletions (D), and insertions (I) of words that are needed to transform the ASR-generated transcription into the reference transcription. These errors are then summed up and divided by the total number of words (N) in the reference transcription. The formula for WER is expressed as:

$$\text{WER} = \frac{S + D + I}{N}. \tag{2.2}$$

This formula results in a ratio that gives the percentage of words that were incorrectly transcribed by the ASR system.

WER is particularly valuable because it offers a straightforward and interpretable assessment of an ASR system's accuracy. A lower WER indicates better performance, with a WER of 0% representing perfect transcription accuracy. However, it is important to note that while WER is a useful indicator of overall performance, it does not account for the variability in the seriousness of errors. For instance, misrecognizing a key word could be more harmful to understanding than misrecognizing a less critical word, but both errors contribute equally to the WER.

2.3.2 CTC Loss

In the field of automatic speech recognition, Connectionist Temporal Classification (CTC) loss [Graves et al., 2006] serves as a key component in training models to convert audio signals into textual transcriptions. CTC loss was developed to help overcome the difficulties of perfectly matching audio frames with their text outputs. It allows for the training of end-to-end deep learning models without having to line up the audio and text exactly.

CTC introduces an augmented set of characters, adding a CTC blank symbol to the standard character set. This extended set serves as the label space, allowing each input frame to be probabilistically associated with a label by the ASR model. To illustrate CTC's functionality, consider the task of transcribing the phrase "small hat", which consists of 9 characters, from an audio input with $N \geq 9$ frames. The label set for this task includes the English alphabet, a whitespace symbol, and the special blank symbol, represented as "-". Given that the number of frames exceeds or equals the length of the transcription, multiple frames may be assigned the same label, leading to various possible extensions of the original phrase. For instance, valid extensions such as "sss-m-aaal-ll[ws]-hhhaa–tt-" or "–sm-aa-lll-l[ws][ws]–hhh-aaa-t" can be condensed back to the original phrase by collapsing consecutive identical labels and then eliminating the blank symbols. This process might reintroduce certain repetitions, such as the double "l" in "small". This is intentional in CTC design to handle uncertainties and separate repeated letters in transcriptions. Note that sometimes letter repetitions may change the meaning of the word, such as the Czech words "raci" (crayfish) vs. "racci" (seagulls).

For an input with N frames, the CTC loss considers M^N potential extensions of the target transcription, where M denotes the size of the augmented CTC vocabulary. The ASR model outputs a probability distribution over this augmented vocabulary for each frame, which can be represented as a matrix $P \in [0,1]^{M,N}$. This matrix, similar to a grid, allows for the tracing of paths from the initial to the final frame. Each path can lead to a longer transcription that might match an extended version of the target phrase. Given that P is a matrix of probabilities, the likelihood of any given path can be easily calculated. The goal of the CTC loss will be to force the model to output such matrix P that maximizes the likelihood of all possible extensions of the target transcription.

To compute the likelihood of extensions efficiently, CTC uses dynamic programming. It employs the forward-backward algorithm to calculate the sum of probabilities of all possible paths that lead to a given transcription. This approach ensures that every valid extension, contributes to the final probability of the transcription, enabling the model to learn from the full spectrum of temporal variations in speech.

2.3.3 Decoding Strategies

In the field of Automatic Speech Recognition, decoding strategies play a crucial role in translating the probabilistic outputs of a model into clear, readable text. Among the various approaches, greedy decoding and beam search decoding are two widely employed methods, each with its own advantages.

Greedy decoding is the simplest form of decoding used in ASR systems. At each step of the sequence, it selects a token with the highest probability according to the model's predictions. This process is repeated for each step in the sequence until a termination condition is met, such as completing a predefined sequence length. The main advantage of greedy decoding is its computational efficiency, as it does not consider multiple ASR hypotheses at a time. However, this simplicity comes at a cost. Greedy decoding can often lead to suboptimal choices because it does not consider the overall sequence probability and can not revise past decisions. In scenarios where the context significantly influences the meaning or the choice of words, greedy decoding may fail to capture the best transcription.

Beam search decoding [Sutskever et al., 2014], on the other hand, offers a more advanced method to overcome the limitations of the simpler greedy strategy. Instead of selecting the single best token at each step, beam search considers multiple hypotheses simultaneously. The beam width parameter defines the number of hypotheses kept at each step of the decoding process. At every step, the model expands each hypothesis in the beam by one additional token, calculates the likelihood of these extended sequences, and retains only the most probable hypotheses within the defined beam width. This process continues until the termination condition is reached for all hypotheses in the beam.

Beam search effectively balances between the brute-force exploration of all possible sequences and the overly simplistic approach of greedy decoding. By considering multiple hypotheses, it increases the chance of finding a more accurate transcription, especially in complex or ambiguous situations where context plays a significant role. The beam width parameter allows for control over the trade-off between decoding accuracy and computational complexity: a wider beam increases the chances of finding a better sequence but requires more computational resources.

Despite its advantages, beam search is not without drawbacks. The choice of beam width significantly impacts performance and computational cost. A small beam width might not sufficiently explore the space of possible sequences, leading to errors similar to those seen with greedy decoding. Conversely, a very large beam width can make the decoding process computationally expensive and slow, canceling out the benefits of increased accuracy.

Additionally, beam search decoding can be enhanced by integrating a language model to improve the accuracy of ASR transcriptions. A language model predicts the likelihood of a sequence of words occurring together in a given language, providing contextual information that can guide the decoding process towards more grammatically correct and semantically meaningful transcriptions. This is particularly effective for resolving ambiguities in speech recognition, where multiple interpretations of an audio signal are possible. This integration leverages the strengths of both the acoustic model, which interprets the audio signals, and the language model, which understands the linguistic patterns.

A statistical n-gram language model is a common type of language model used in this context. It estimates the probability of a word based on the occurrence of the preceding $n-1$ words, where n represents the size of the "window" of words considered. For example, a trigram (3-gram) language model would predict the probability of a word occurring given the two preceding words. These models are trained on large corpora of text data, allowing them to capture common linguistic patterns and structures found in a language.

When beam search is used with a language model, the decision at each step in the decoding process is influenced not only by the acoustic model's predictions but also by the language model's scoring. For each candidate sequence in the beam, the language model's probability score is combined with the acoustic model's score to evaluate the overall likelihood of the sequence. This combination typically involves a weighted sum, where weights can be adjusted to balance the influence of the acoustic and language models based on the task or domain.

In practice, the choice between greedy decoding and beam search decoding, as well as the configuration of parameters like beam width, depends on the specific requirements and constraints of the ASR task at hand. For tasks where speed is crucial and the linguistic context is restricted to a specific domain, greedy decoding might suffice. In contrast, for tasks requiring high accuracy and where contextual clues are important for disambiguating the speech signal, beam search decoding is often preferred.

2.4 Transformer Architecture

The Transformer is a neural network architecture [Vaswani et al., 2017] designed for processing sequences on the input and also returning sequences on the output. Since 2017 it has become the foundation of many state-of-the-art models in natural language processing and beyond.

The Transformer architecture, as depicted in Figure 2.1, consists of an encoder and a decoder, each constructed from a series of identical layers. The encoder processes the input sequence of tokens, converting it into a continuous representation that encapsulates the interrelations among all components of the input. This representation is subsequently utilized by the decoder to iteratively generate the output sequence. Notably, each layer within both the encoder and decoder consists of two principal sub-

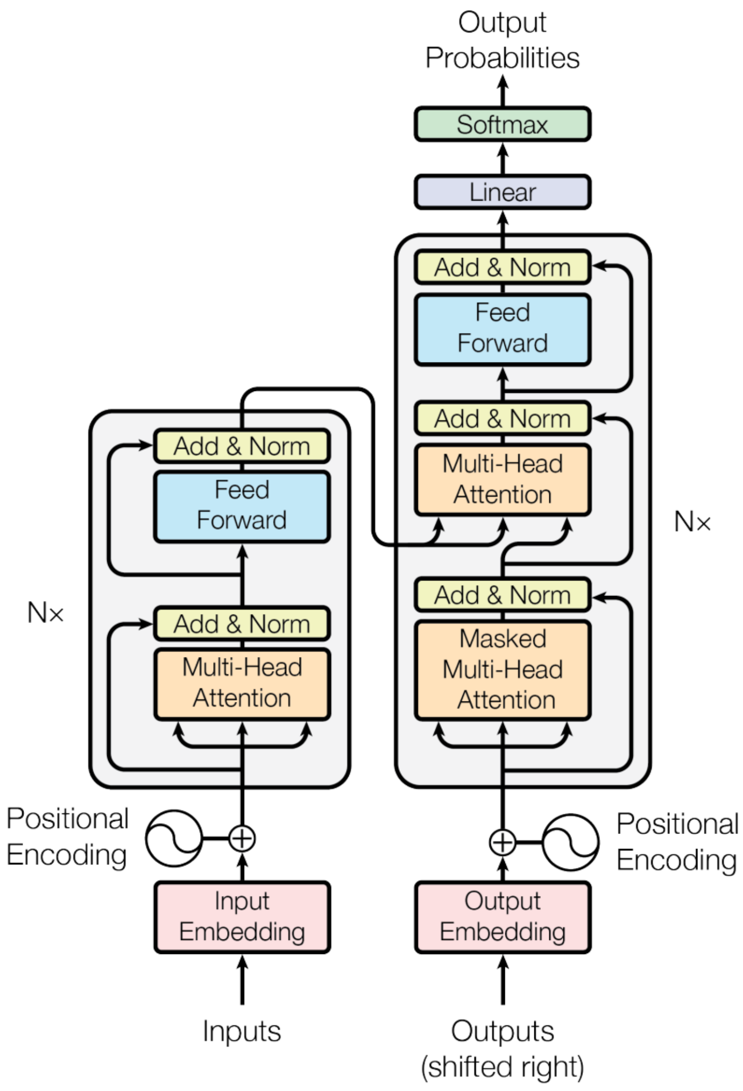


Figure 2.1: Transformer architecture overview from Figure 1 in the original research paper [Vaswani et al., 2017].

components: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network.

At its core, self-attention mechanism operates on a set of queries, keys, and values, which are all derived from the same input sequence. Given an input sequence of N tokens, we obtain an input matrix $X \in \mathbf{R}^{N,d_x}$, from which queries ($Q \in \mathbf{R}^{N,d_k}$), keys ($K \in \mathbf{R}^{N,d_k}$), and values ($V \in \mathbf{R}^{N,d_v}$) are derived through linear projections using learned weight matrices $W^Q \in \mathbf{R}^{d_x,d_k}, W^K \in \mathbf{R}^{d_x,d_k}$, and $W^V \in \mathbf{R}^{d_x,d_v}$, respectively. Specifically, for the input sequence X , we compute: $Q = XW^Q, K = XW^K, V = XW^V$. Here, the variables d_x, d_k, and d_v denote the latent dimensionality for the original tokens, keys, and values, respectively, with the dimensions for keys and queries being identical.

The core of the self-attention mechanism is to compute the attention scores, which indicate how much focus elements of the input sequence should put on the rest of the sequence. The attention scores are calculated by taking the dot product of the query with all keys, followed by a scaling factor of $1/\sqrt{d_k}$. This scaling factor is used to prevent the dot product from growing too large in magnitude, which could lead to computational difficulties during training. The attention scores are then passed through a softmax function to obtain the attention weights, which are probabilities representing the importance of each value:

$$\text{Attention}(Q, K, V) = \text{softmax}\Big(\frac{QK^T}{\sqrt{d_k}}\Big)V \tag{2.3}$$

The softmax operation ensures that the weights sum to 1, allowing them to be interpreted as probabilities. Each output element is a weighted sum of the values, where the weights are determined by the attention scores. This allows each position in the output sequence to dynamically attend to all positions in the input sequence, thereby capturing their contextual relationships.

Multi-head attention further extends the self-attention mechanism by paralleling multiple attention heads, each with its own set of weight matrices W^Q, W^K, and W^V. This design enables the model to capture information from different representation subspaces at different positions. The outputs of all heads are concatenated and linearly transformed to produce the final output:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O, \tag{2.4}$$

where $\text{head}_i = \text{Attention}(XW_i^Q, XW_i^K, XW_i^V)$, parameter h represents the number of heads, and $W^O \in \mathbf{R}^{hd_v, d_x}$ is a weight matrix used to combine the outputs of different heads. In the source paper the parameters set as follows $h = 8$ and $d_k = d_v = d_x/h = 64$.

For the decoder part, during the training time the self-attention computations slightly differs from Equation (2.3). At a closer look, the self-attention allows each token in the input query every other preceding or subsequent token. This behaviour is undesired in the decoder component that during inference generates one token at a time and has acces only to the previously generated tokens. That is why during training we multiply attention scores by binary lower triangular matrix, that ensures that each token has access to only previous positions. The modified multi-head attention for the decoder part is called masked multi-head attention, and this masking strategy is usually denoted as causal masking.

Also to fuse encoded input into the decoder, Transformer architecture utilize multi-head cross attention between decoder and encoder. In this slight modification matrix $Q = X_d W^Q$, where X_d stands for the decoded output sequence and the other matrices $K = X_e W^K, V = X_e W^V$ are generated from the encoded input sequence. After this we compute attention scores similar to Equation (2.3) on these matrices.

The feed-forward network applies two linear transformations with a ReLU activation in between, processing the output of the multi-head attention mechanism. Specifically, we have $\text{FFN}(Z) = \max(0, ZW_1 + b_1)W_2 + b_2$, where $Z \in \mathbf{R}^{N, d_x}$ is the output of the multi-head block and W_i, b_i are learnable parameters of the linear transformations. Also the first transformation usually maps the input to higher dimension and the second one transforms it back.

This gives us all the components for the transformer building blocks. Encoder consists of N layers were each layer has a multi-head self-attention part and the feed forward part. Decoder also consists of M layers, where each has masked multi-head attention part followed by multi-head cross attention and feed forward part. Original Transformer architecture offers making decoder and encoder of the same size, with both having 6 layers. Additionally, additive skip connections with Layer Normalization [Ba et al., 2016] are used inside each layer between all its parts.

Positional encoding is another important component of the Transformer, as it injects information about the order of the sequence into the model. Since the self-attention mechanism does not inherently consider the sequence order, positional encodings are added to the input embeddings of the encoder and decoder. This ensures that the model can take into account the position of each word or element in the sequence.

One of the key benefits of the Transformer's design is that it allows effective parallelization during training compared previous approaches, drastically reducing training times for large models. The other one is that its design makes it easy to scale the the network size, making the architecture applicable in situations with tight requirements and also in tasks where large computation is traded for better results. In general making it possible balance performance and computational requirements. Finally, as we will see in the following chapters, the architecture was so successful, that even its components, specifically decoder and encoder, can be used alone to solve more specific tasks. On the other hand, the major drawback of Transformers is the quadratic memory requirement with respect to the sequence length. The source of high memory utilization is the self-attention mechanism. As can easily be seen from Equation (2.3), the product QK^T, will consume quadratic memory with respect to the input length. This drawback leads to application limitations of the Transformer architecture on longer input sequences.

2.5 BERT Architecture

BERT (Bidirectional Encoder Representations from Transformers) [Devlin et al., 2019a] is a neural network architecture for natural language processing introduced by Google in 2018. BERT leverages the Transformer architecture, specifically its encoder mechanism, to achieve state-of-the-art performance on a wide range of NLP tasks without task-specific architectural modifications. Since Transformer was designed to support model scaling by changing various hyper-parameters, BERT architecture offers two versions: base with total of 12 layers, $d_x = d_v = d_k = 768$ and 12 attention heads; and large with 24 layers, $d_x = d_v = d_k = 1024$ and 16 attention heads.

Unlike previous models that processed context in a sequential manner (from left to right, from right to left or from both directions), in BERT each token has access to every other token in the sequence in constant time, which directly follows from the attention mechanism Equation (2.3), specifically from QK^T product. This made possible to introduce "masked language modeling" (MLM) pre-training objective, where a certain percentage of the input tokens are randomly masked out and the model is trained to predict these masked tokens based the unmasked context. This approach allows BERT to capture a deep, bidirectional understanding of language context.

BERT's architecture consists of multiple layers of Transformer encoder stacked on top of each other. The model is pre-trained on a large corpus of text, e.g. the entire Wikipedia, using the MLM task alongside a next sentence prediction (NSP) task. The NSP task trains the model to predict whether two given segments of text naturally follow each other, which helps BERT understand the relationships between sentences.

The MLM and NSP tasks allowed models with BERT architicture to be pre-trained without any supervised data, afterwards the pre-trained model, that already gained solid knowledge of the natural language, could be fine-tuned for specific NLP tasks, by adding a signle linear layer on top of BERT. During fine-tuning, the pre-trained BERT model is adapted to a specific NLP task with relatively minimal additional training, leveraging the deep contextual representations learned during pre-training.

However, since BERT at its core relies on the attention mechanism, the architecture suffers from the same limitations as Transformer architecture. Specifically, because of the quadratic memory requirements with respect to the input length, BERT models work on sequences with limited length and can not directly generalize to arbitrary sequence lengths.

In summary, BERT's introduction marked a significant milestone in NLP research, showing the power of pre-trained models and the effectiveness of bidirectional context understanding. By using the Transformer architecture's encoder and adding the MLM objective, BERT has set a new standard for NLP models, enabling more effective language understanding applications.

2.5.1 RoBERTa

RoBERTa (Robustly optimized BERT approach) [Liu et al., 2019] builds upon BERT's foundation, keeping its architecture but incorporating several key modifications to the training procedure that significantly improve its performance across a wide range of natural language processing tasks.

One of the primary improvements in RoBERTa is the removal of the next sentence prediction task during pre-training. The original BERT model was pre-trained using both the MLM task and the NSP task. However, the RoBERTa team found that eliminating the NSP task and focusing solely on a more thoroughly optimized MLM task led to better performance.

Additionally, RoBERTa significantly increases the size of the pre-training data and the overall scale of pre-training. It is trained on a dataset that is ten times larger than that used for BERT, and with longer training sequences. This extensive pre-training enables RoBERTa to develop a deeper understanding of language nuances and complexities. Furthermore, RoBERTa uses dynamic masking rather than static masking, meaning that the masked tokens are changed during the training iterations, providing a more diverse training signal. The model also benefits from larger batch sizes and more training steps, which contribute to its improved performance.

2.5.2 RobeCzech

RobeCzech [Straka et al., 2021] is introduced as a Czech contextualized language representation model built upon the RoBERTa architecture, exclusively trained on Czech data, making it a monolingual model. RobeCzech was trained using a large collection of Czech texts that includes newspaper and magazine articles, web corpus documents, and texts extracted from Czech Wikipedia, totaling approximately 4.9 billion tokens.

In comparison to existing models such as multilingual BERT [Devlin et al., 2019b], XLM-RoBERTa [Conneau et al., 2020], Slavic BERT [Arkhipov et al., 2019] and Czert [Sido et al., 2021], RobeCzech demonstrated superior performance across five evaluated NLP tasks, including morphological tagging, dependency parsing, named entity recognition, semantic parsing, and sentiment analysis. Notably, it set new state-of-the-art results in four of these tasks and showed significant improvements in sentiment analysis, where it was only outperformed by the significantly larger XLM-RoBERTa model.

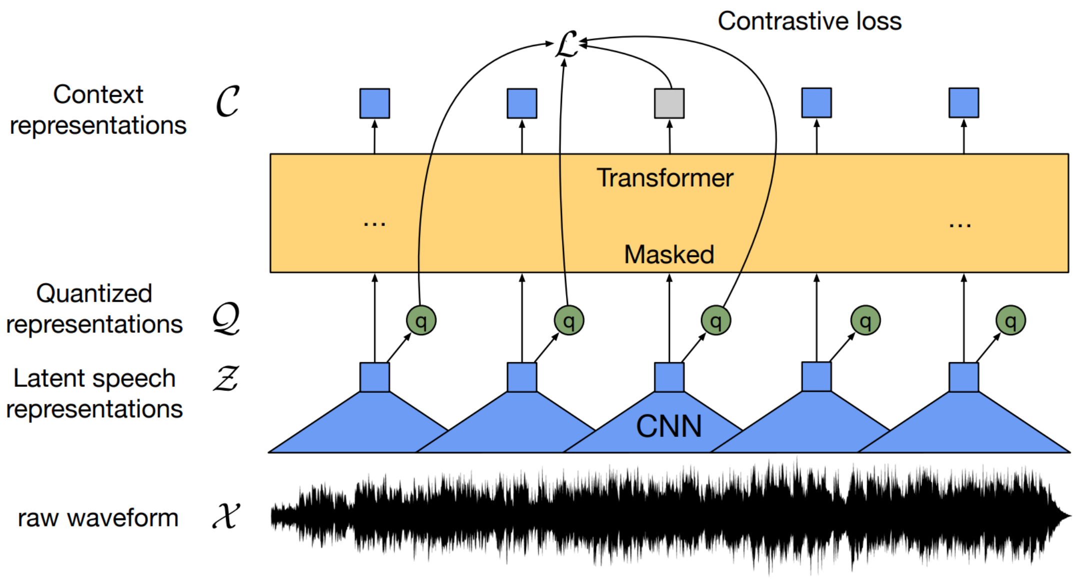


Figure 2.2: Wav2vec 2.0 architecture overview from Figure 1 in the original research paper [Baevski et al., 2020].

These results highlight RobeCzech's effectiveness and its potential as a powerful tool for Czech language processing applications.

2.5.3 Wav2Vec 2.0

Wav2vec 2.0 [Baevski et al., 2020], introduced by Facebook AI, represents a significant advancement in the domain of speech processing. The core idea behind wav2vec 2.0 is to train the model in two stages: pre-training and fine-tuning. During pre-training, the model learns valuable representations of the audio signal in an unsupervised manner. It does this by masking parts of the audio input and predicting the masked portions based on the unmasked context similar to idea of MLM objective in NLP domain. A key aspect of wav2vec 2.0 unsupervised pre-training is its use of contrastive loss. This loss function encourages the model to distinguish between the true audio signal and a set of negative samples (i.e., incorrect audio pieces). The pre-training phase enables the model to understand the underlying structure and features of speech without requiring labeled data.

Figure 2.2 illustrates wav2vec 2.0 framework, that learns contextualized speech representation using Transformer's encoder component and a usage of the discretized speech units. The wav2vec 2.0 architecture is primarily composed of two parts: the feature encoder and the context network. The feature encoder takes raw audio waveforms as input and apply several blocks of convolutions coupled with non-linearities and normalization producing latent representations of the audio samples. The context network, which is built upon the Transformer's encoder architecture, processes these latent representations to capture the temporal context from the audio sequence. Additionally, unlike the NLP domain where the input is a sequence of a discrete elements, namely tokens, in speech domain the input is continuous signal, so to model unsupervised MLM-like objective the speech should be discretized. That is why during the unsupervised pre-training phase the output of the feature encoder is quantized to a finite set of speech representations via product quantization [Jégou et al., 2011], that amounts to choosing discrete representations from multiple codebooks and concatenating them together.

For fine-tuning, wav2vec 2.0 applies a supervised approach where the model is further trained on a set of labeled data, e.g. with CTC loss function. The combination of unsupervised pre-training and supervised fine-tuning allows wav2vec 2.0 to leverage the strengths of both learning paradigms. By learning general audio representations in an unsupervised manner and then fine-tuning on task-specific labeled data, the model achieves remarkable accuracy in speech recognition tasks, requiring less data for the supervised phase.

Unsupervised pre-training, as employed in the wav2vec 2.0 architecture, is particularly advantageous when dealing with low-resource languages. As in case of BERT architecture with invented MLM objective, the authors show that the large amount of unsupervised pre-training makes possible to achieve high performance in ultra-low resource fine-tuning scenario. Specifically, for English language and Librispeech dataset [Panayotov et al., 2015] with 960 hours of transcribed data used for pre-training without transcriptions, the wav2vec 2.0 Base (95M parameters) with 4-gram LM achieves 5.5/4.3/3.4 WER on the clean test set while using only 1/10/100 hours of labeled data.

3. Data

In this chapter, we explore available data for punctuation and case restoration, create a new dataset that will be more suitable for training and evaluating future speech-informed ITN models.

3.1 Data Requirements for Speech-Informed ITN

Consider a model capable of restoring punctuation and casing for an arbitrary sequence of words. The primary use case for such a model comes from speech recognition tasks, where the output is often an unsegmented, lowercase text stream without punctuation. While humans can generally understand these textual outputs, their utility diminishes when used as inputs for downstream tasks such as summarization or translation. Typically, models for these text-to-text tasks are trained on well-structured written text, complete with casing and punctuation, as these elements significantly impact downstream model performance. The absence of casing and punctuation in inputs can lead to substantial performance degradation. This issue becomes more pronounced in streaming scenarios, where input speech is not divided into distinct segments. Consider a streaming case in a spoken language translation task, where the solution consists of a cascade of neural models: a speech recognition model followed by a translation model. If the translation model, likely trained on text pairs, receives unsegmented, lowercase text without punctuation as input, the quality of the translation will be severely affected. In summary, an ITN model serves to adapt recognized spoken language into a written text format, thereby providing downstream models with better-quality inputs.

Ideally, punctuation and case restoration model should also perform effectively in streaming scenarios, independent of initial text segmentation provided by audio. The streaming scenario can be modeled by a small buffer that accumulates recognized text. Consequently, the model operates only on a small window of text at a time. The buffer size introduces a trade-off between model accuracy and overall latency of the predictions. A larger buffer corresponds to more accurate results but with higher latency.

Following the discussion of the suitable model for the task of punctuation and case restoration, now we focus on the desired data for this model. Since the model should function independently of audio-based segmentation of input text, the training data should not be aligned with sentence or paragraph boundaries. Ideally, the golden-truth text should represent a segment of recognized audio, without further assumptions. In this scenario, the resulting model will be capable of functioning in both offline and online scenarios (assuming a small window). This approach also eliminates bias, where the input invariably starts with a capital letter and ends with punctuation denoting sentence ending. Notably, such an environment will be more challenging compared to situations with complete sentences as inputs, but it represents a more realistic scenario for the model's use case.

Having access to a large amount of well-structured text, such as paragraphs from Wikipedia, can help in the creation of the previously described dataset. However, these textual data lack corresponding audio representations, making it impossible to estimate the importance of sound for punctuation and case restoration. The desired experimental setup forces reliance solely on available speech recognition data. Ideally, the data should be created from large textual fragments spoken by various speakers. If the words have corresponding timestamps, we can create textual segments of varying sizes, independent of sentence boundaries, and extract the underlying audio. Segmenting data into windows of different sizes will train the model to work in an online-like scenario with short inputs.

3.2 Exploring ParCzech 3.0

A potential dataset that has most of the desired characteristics is ParCzech 3.0 [Kopp et al., 2021]: A Large Czech Speech Corpus with Rich Metadata. ParCzech 3.0 includes a speech corpus of Czech parliamentary speech recordings from The Czech Chamber of Deputies, spanning from 25th November 2013 to 1st April 2021.

We contributed to ParCzech 3.0 by creating this large speech dataset, suitable for developing and evaluating speech recognition neural models. This dataset includes segments aligned with sentence boundaries derived from the official transcriptions. Each segment included was recognized by a speech recognition model to ensure consistency with the official transcript. The detailed process of creating this speech dataset will be described in the upcoming section.

The official website of The Czech Chamber of Deputies includes stenographic texts from the chamber's sittings, organized into terms, meetings, sittings, and agenda items. These texts and corresponding audio files were extracted through web scraping. However, the data encoding across the website is not consistent. In cases where data is missing on a page, the downloading procedure attempts to locate it in the list of audio files.[1] If not listed, the script infers the missing URL from the date and time mentioned on the imperfect web page, thereby maximizing the extraction of audio data.

Subsequently, all stenographic transcripts were automatically annotated with UDPipe 2 [Straka, 2018], providing tokenization, morphological, and syntactic analysis.

It is important to note that the official transcripts may not always perfectly align with the audio files' content. Discrepancies are noticeable when playing audio files alongside their corresponding transcriptions. This divergence results from the fact that the transcribed texts were sometimes modified during transcription to enhance formality and eliminate factual errors.

Table 3.1 presents the original data statistics. The number of stenographic transcripts slightly exceeds that of the audio files, with 100 documents lacking a corresponding transcript. Additionally, the total length of the audio files stated in Table 3.1 includes overlaps, as the audio data from consecutive meetings slightly overlap to ensure no information loss due to the division of audio into multiple files. This implies, that the actual useful duration will be smaller than the sum of all durations.

3.2.1 Speech Recognition Data

We processed the stenographic texts from ParCzech 3.0 to extract word timings from the audio using a GMM-based ASR system [Krůza, 2020]. The ASR model generates a time-stamped sequence of words, which, while closely resembling the expected transcript, may not be identical due to either corrections in the data or actual ASR errors. Consequently, we aligned the recognized text and official transcripts using the

[1] e.g., `https://psp.cz/eknih/2013ps/audio/index.htm`

Number of unique persons	486
Number of source audio files	20 674
Number of steno source web pages	20 775
Source audio length (hours incl. overlaps)	4 815.31
Time period	25th Nov 2013 – 1st Apr 2021

Table 3.1: Original data statistics of ParCzech 3.0. This table is reprinted from [Kopp et al., 2021].

	Original data	Filtered data
Hours	3 071.57	1 332.38
Segments	1 391 785	606 540
Average segment duration in seconds	7.94±11.53	7.90±7.14
Average number of words in a segment	15.91±16.32	16.72±13.73
Words	22 153 778	10 146 591
Aligned words percentage	89.6%	96.3%
Unique Speakers	475	474
Segment size range	[1, 1058]	[2, 138]
Duration range	[0.0, 720.76]	[0.82, 53.99]

Table 3.2: Comparative statistics of the original and filtered ParCzech 3.0 [Kopp et al., 2021] data.

Needleman-Wunsch algorithm with affine gap penalties [Needleman and Wunsch, 1970], treating each word as an atomic unit. This alignment strategy aims to maximize the length of contiguous sequences between the recognized words and their corresponding stenographic transcripts, rather than aligning each recognized word individually. This approach effectively minimizes the number of small gaps in aligned sequences, where a gap means either a recognized or transcribed word aligned to a blank.

Each recognized word is associated with specific start and end times. After the alignment, the original audio files are segmented to match sentence boundaries of the original transcript.

Given that segment-level alignment may not be perfect, we equip each segment with a file containing statistics for potential corpus filtering. These statistics include the number of words and characters, the percentage of missed words and characters, and sound coverage, calculated as the percentage of sound duration where words were recognized.

Additionally, we provide the global statistics for each audio file, reflecting the overall alignment quality. These statistics include the percentage of missed words, median normalized edit distance, the 80th percentile of normalized edit distance, and normalized continuous gap count. In the continuous gap count a sequence of gaps is counted as a single continuous gap, number of continuous gaps allows to detect transcripts with a highly fragmented alignment.

ParCzech 3.0 offers both the original and a filtered version of the data, with one training set and distinct test and validation splits. The corpus is also available in an unfiltered form, allowing for custom filtering to balance corpus size and quality. Table 3.2 presents the data statistics before and after filtering. Notably, the hours listed in the table no longer include starting and ending overlaps, which gave approximately 15% in the total length of all audio files.

3.2.2 Discussion of Data in ParCzech 3.0

Referring to Table 3.2, it is observed that after filtering, only 44% of the original audio data remains. This significant reduction, amounting to 56% of the original data size in terms of hours, suggests that differences between the transcripts and the sound may not be the sole factor. The errors introduced by the GMM-based ASR model could also contribute to this reduction. Another potential reason for the data reduction is the segmentation strategy, which focuses on complete sentences or smaller groups of sentences. Consequently, accurately recognized text segments within larger sentences might be disregarded due to recognition errors in other parts of the sentences. The table further illustrates that, despite the filtering process, the cleaned data still have some misalignment with the official stenographic transcript.

As outlined in the data requirements section Section 3.1, an ideal dataset for our purposes would consist of extensive texts with punctuation and casing, spoken by diverse speakers. Each word in these texts should ideally have its associated timestamps, enabling segmentation into various lengths independent of sentence boundaries. ParCzech 3.0 nearly fulfills these criteria. However, a notable limitation is that the dataset is segmented in alignment with sentence boundaries. Despite this, the release of the corpus with original, unsegmented data allows for custom segmentation and filtering according to our specific requirements.

This flexibility in processing the original data of ParCzech 3.0 allows us to modify the dataset to better suit the needs of our work. By re-segmenting and filtering the data, we can create a corpus that more closely aligns with the desired characteristics for effective training and evaluation of punctuation and case restoration models.

3.3 Development of a Segment-Based Dataset from ParCzech 3.0

Building upon the original data of ParCzech 3.0, our objective is to construct a dataset containing segments that disregard sentence boundaries. To achieve this, we replicate the steps used in ParCzech 3.0 but employ a custom-trained ASR model based on wav2vec 2.0, coupled with a beam search decoding to enhance speech recognition capabilities. The specifics of training of the wav2vec 2.0 model with beam search will be described in Chapter 4. For the present discussion, it is assumed that we have at our disposal a trained Czech ASR model capable of processing the input audio files.

3.3.1 Preparing the Data

The initial phase involves performing speech recognition on the provided audio files. Given that each file averages 13 minutes in length, direct recognition may be impractical due to the wav2vec 2.0 model's quadratic memory requirements. Therefore, we segment the audio files into shorter fragments, approximately 30 seconds each. During segmentation, we also implement a 5-second overlap between consecutive segments to mitigate the risk of splitting words. This overlap is sufficiently large to enable connection of adjacent audio parts by identifying overlapping recognized words at the segment boundaries.

Speech Recognition and Timestamp Extraction

The first step in speech recognition involves processing the audio fragments through the wav2vec 2.0 model to obtain unnormalized scores, or logits, for each grapheme at every timestamp. To convert logits into recognized text, we employ beam search decoding augmented with an N-gram statistical language model. This model is used in rescoring the recognition hypotheses generated by wav2vec 2.0.

Following the speech recognition phase, our next step is to extract corresponding timestamps for the recognized text. The algorithm employed for this purpose is based on the methodology outlined in the paper [Kürzinger et al., 2020] and the implementation provided in a torchaudio's [Yang et al., 2021] IPython notebook.[2] The setup for the algorithm is as follows:

- An extended set of graphemes $\mathbf{G} = \mathbf{G}' \cup \{b\}$, where $\mathbf{G}'$ is the set of graphemes derived from the lower-cased Czech alphabet, including white-space. This set, of size $N-1$, is extended with a special CTC-blank symbol denoted as b.
- The recognized transcript $\mathbf{C}$, of length M, where $c_i \in \mathbf{G}'$, for $i \in \{1, \ldots, M\}$.
- The input sound $\mathbf{S}$, of length T'.
- The logits matrix $\mathbf{L} \in \mathbb{R}^{T,N}$, where $T \leq T'$ represents the number of audio frames after processing through the wav2vec 2.0 model.

The first step is to create a probability matrix $\mathbf{P}_{t,i} = p(c_i, |t, S), c_i \in \mathbf{G}, i \in \{1, \ldots, N\}, t \in \{1, \ldots, T\}$ and a trellis matrix $\mathbf{K} \in [0,1]^{T+1, M+1}$. The probability matrix $\mathbf{P}$ is computed from the logits by applying softmax, $\mathbf{P}_{t,\bullet} = \text{softmax}(\mathbf{L}_{t,\bullet}), t \in \{1, \ldots, T\}$. The elements of the trellis matrix, denoted as $k_{t,j}, t \in \{0, \ldots, T\}, j \in \{0, \ldots, M\}$ are computed using the following rule:

$$k_{t,j} = \begin{cases} \max\left\{k_{t-1,j} \cdot p(\mathrm{b}|t, S),\ k_{t-1,j-1} \cdot p(c_j|t, S)\right\} & \text{if } t > 0 \text{ and } j > 0 \\ 0 & \text{if } t = 0 \text{ and } j > 0 \\ 1 & \text{if } j = 0 \end{cases} . \tag{3.1}$$

As the rule suggests, $k_{t,j}$ represents the joint probability of emitting the first j characters of the transcript $\mathbf{C}$ up to time t. When moving to time t, we select the most probable scenario between emitting a CTC-blank b and staying at the same grapheme in the transcript, or emitting the next grapheme from the transcript. The transition cost for staying at the first character is set to zero to align the transcription start with an arbitrary point in the audio file.

Upon computing the trellis, we create the character-wise alignment $\mathbf{A}$, such that $a_t \in \{1, \ldots, M\}$ for $t \in \{1, \ldots, T\}$. The alignment process starts from the most probable temporal position for the last grapheme in the transcript c_M, denoted as $t_{\text{start}} = \arg\max_\tau k_{\tau,M}$. Subsequently, we sequentially decrease t towards 0. For $t \leq t_{\text{start}}$, the alignment is determined using the following rule:

$$a_t = \begin{cases} M-1 & \text{if } t \geq \arg\max_\tau k_{\tau,M-1} \\ a_{t+1} & \text{if } k_{t,a_{t+1}} \cdot p(\mathrm{b}|t+1, S) > k_{t,a_{t+1}-1} \cdot p(c_j|t+1, S) \\ a_{t+1} - 1 & \text{else} \end{cases} . \tag{3.2}$$

[2]https://pytorch.org/audio/stable/tutorials/forced_alignment_tutorial.html

Given the established alignment, we can calculate the starting and ending times for each grapheme relative to the audio by collapsing consecutive identical graphemes in the alignment. This approach allows us to determine timestamps for each grapheme. To obtain timestamps for each word, we consider the timestamps of the first and last graphemes of the word. It is important to note, that these timestamps are initially computed with respect to the latent audio frames. To translate these timestamps to the original audio of length T', as opposed to its latent representation of length $T \leq T'$, we employ the following steps:

1. Calculate the ratio between the number of frames in the original waveform and the number of frames in the latent waveform.
2. Multiply the index of each latent audio frame by this ratio.
3. Round the result to obtain the indices of the word timestamps in the original audio.

By dividing these indices by the sample rate, we convert them into time measurements within the original part of an audio file.

Merging Partial Transcripts with Timestamps

After computing timestamps for the recognized text in each audio fragment, our next step involves merging these partial transcripts to reconstruct the recognized text with timestamps for the entire original audio file. Consider two consecutive fragments, f_i and f_{i+1}, with an overlap of a few seconds, and their corresponding recognized transcripts r_i and r_{i+1}. The objective is to merge these transcripts at the midpoint of the overlap, ensuring that the final words from r_i and the initial words from r_{i+1} are recognized with sufficient audio context from both segments.

In instances where the audio split occurs in the middle of a word, the last recognized word in r_i (denoted as $r_i[-1]$) and the first recognized word in r_{i+1} (denoted as $r_{i+1}[1]$) may not align. It is possible that this mismatch extends to k words, such that $r_i[-1 : -k] \neq r_{i+1}[1 : k]$. In such cases, the non-matching words are discarded, and in the worst-case scenario, we retain the last word of r_i before the overlap and the first word of r_{i+1} after the overlap. Data observations indicate that length of the problematic part is 3 or less. Complete discarding of the overlap is not typically necessary.

If the original audio was divided during a brief interword pause, the recognized transcripts r_i and r_{i+1} will exhibit minimal or no mismatched words. This scenario is handled similarly to the previous case.

In situations where the split occurs during a prolonged silence exceeding the overlap duration, there will be no overlapping similar words. However, we can be confident that the last word of r_i and the first word of r_{i+1} both have adequate audio context on either side. In such cases, the transcripts are simply concatenated, taking the recognized text with timestamps as they are.

Through this approach, we sequentially join the recognized transcripts with their corresponding timestamps, yielding the full transcript for the original audio file.

Aligning Recognized Transcripts with Official Stenographic Transcripts

At the current stage of processing, each original audio file is accompanied by a recognized transcript with timestamps and an official stenographic transcript. The recognized transcript is an unsegmented, lowercased text without punctuation but includes timestamps, in contrast to the official transcript, which contains both punctuation and casing. To effectively utilize both punctuation and casing labels along with the timestamps, it is necessary to align these two sequences.

We process and tokenize the official transcripts using the UDPipe 2.0 tool. Consequently, each word, punctuation mark, and number is treated as an individual token. In the original transcript, containing a wide range of punctuation marks, we replace less common punctuation marks (e.g., `[\]`, `[’]`, `[@]`, `[´]`, `[|]`, `[’]`) with whitespaces, and the substitute `[(]` and `[)]` with commas, and `[...]` with periods. Symbols representing words (e.g., `[§]` for “paragraph” or `[%]` for “procento”) and numbers are kept unchanged. Additionally, both lower and upper quotes are standardized to upper quotes. To make possible token classification for the punctuation and case restoration tasks, sequences with consecutive punctuation marks are simplified. For example, a sequence like $w_0\, w_1\, w_2\ :\ “\, w_3\, w_4\, w_5\ ,\ w_6\, w_7\ .\,”$ is reduced to $w_0\, w_1\, w_2\ :\ w_3\, w_4\, w_5\ ,\ w_6\, w_7\ .$, ensuring each word is followed by at most one punctuation mark. After the processing, each word w_i is assigned its punctuation class by appending the trailing punctuation mark. In our example, the labels for w_2, w_5 and w_7 are `[:]`, `[,]` and `[.]`, respectively, while the remaining w_i will be assigned `[blank]`.

The alignment method, inspired by the original ParCzech 3.0 strategy, employs a two-level edit distance procedure. The sequence level operates on words, ideally aligning identical words in both recognized and official texts. This implies a default exact similarity measure at the word level, assigning a score of 0 to non-identical word pairs and 1 to identical words. This strict match can result in aligning similar words to the blanks, specifically, say the word t_i from the official transcript and word r_j from the recognized transcript are not identical and have small edit distance, but instead of aligning the words to each other, (t_i, r_j), the current similarity measure will force alignment $(t_i, b), (b, r_j)$, with b representing blank symbol. To fix this, we introduce a character-level edit distance as a softened version of the exact match similarity.

For better understanding of the alignment procedure, Figure 3.1 demonstrates final alignment between recognized transcript and the official one. The official transcript is presented in its original form, complete with punctuation and uppercasing. For ease of visual comparison, both columns include an edit distance metric that disregards casing and punctuation.

A significant challenge in our data creation strategy and the data itself is the presence of numbers, special symbols, and abbreviations in the official stenographic transcript, or generally unnormalized official texts. This issue is evident in Figure 3.1, where the model correctly recognizes numbers and the `[/]` symbol, but proper alignment with the corresponding words in the official text is problematic. This challenge underscores the reason for extracting timestamps based on the recognized text rather than the official transcripts.

3.3.2 Dataset Statistics

Upon successfully aligning recognized texts with official transcripts, where each aligned official word is timestamped, we proceed to select segments with perfect alignment.

Audio 2017041211181132 (12.04.2017 11:18-11:32)			Audio 2014021816581712 (18.02.2014 16:58-17:12)		
Recognized word	Official word	Edit dist.	Recognized word	Official word	Edit dist.
který	který	0	na	na	0
vám	vám	0	zvyšování	zvyšování	0
byl	byl	0	důchodu	důchodů,	1
doručen	doručen	0	na	na	0
šestého	(blank)	7	slušné	slušné	0
(blank)	6.	1	přídavky	přídavky	0
března	března	0	pro	pro	0
tohoto	tohoto	0	děti	děti,	0
roku	roku.	0	na	na	0
usnesení	Usnesení	0	aktivní	aktivní	0
garančního	garančního	0	politiku	politiku	0
výboru	výboru	0	zaměstnanosti	zaměstnanosti,	0
pak	pak	0	investice	investice,	0
bylo	bylo	0	tak	tak	0
doručeno	doručeno	0	dávat	dávat	0
jako	jako	0	nyní	nyní	0
sněmovní	sněmovní	0	až	až	0
tis	tisk	1	neuvěřitelně	neuvěřitelně	0
devět	(blank)	5	velké	velké	0
set	(blank)	3	finanční	finanční	0
dvanáct	(blank)	7	náhrady	náhrady	0
lomeno	(blank)	6	církvím	církvím	0
třemi	(blank)	5	je	je	0
(blank)	912	3	mimořádný	mimořádný	0
(blank)	/	0	hazard	hazard	0
(blank)	3.	1	s	s	0
nyní	Nyní	0	veřejnými	veřejnými	0
se	se	0	financemi	financemi!	0
táži	táži	0	jistě	Jistě	0
navrhovatelky	navrhovatelky,	0	každý	každý	0
zda	zda	0	alespoň	alespoň	0
má	má	0	trochu	trochu	0
zájem	zájem	0	informovaný	informovaný	0
vystoupit	vystoupit	0	člověk	člověk	0
před	před	0	ví	ví,	0
otevřením	otevřením	0	že	že	0

Figure 3.1: Two examples of the resulting alignment for different audio files. The (blank) words in the verticals identify places where words were aligned to blanks. The third column in each example shows the edit distance between aligned words disregarding punctuation and special symbols.

Statistics	Before	After
Speakers	475	475
Words	22 067 968	19 370 266
Hours	3 071.6	2 817.2
Files	20 421	20 421

Table 3.3: Dataset properties before and after filtering for zero edit distance. Files represent the number of official transcripts.

This is done by discarding words with non-zero edit distances, yielding segments where every word is accurately recognized and timestamped. This strict criterion is essential as errors in alignment could stem from either inaccuracies in ASR recognition or modifications in the official transcript. A notable limitation of this approach is the exclusion of alignments between abbreviations and their expansions or between numbers and their verbal forms. This is a necessary compromise due to the unnormalized nature of the official transcript and the potential for misalignment.

Table 3.3 presents dataset characteristics before and after filtering for zero edit distance. It is important to note, that the word count here excludes punctuation, differing from the counts in Table 3.2. After the filtering, a significant portion of the data (92% in duration and 88% in word count) is preserved, despite the unnormalized and corrected nature of the official transcript. Recognition with our ASR system allowed us to preserve all speakers and all official transcripts, excluding parts with a positive edit distance. Also, our filtered dataset retains all original speakers and transcripts.

3.3.3 Construction of Training and Evaluation Items

We now further process the aligned data to create appropriate training and evaluation entries for the punctuation restoration and capitalization tasks. Specifically, the data must be segmented into varying lengths, thereby simulating a streaming scenario with a constrained audio window. Additionally, segments that are excessively brief, lacking sufficient textual and auditory context, should be excluded. Due to the absence of human supervision, we must filter the segments based on various statistical measures to ensure the cleanliness of the data. While this filtering process may eliminate some viable segments, the primary objective is to remove those segments that are potentially misleading for the model. Subsequently, the refined pool of segments will be split into training, testing, and validation sets following the original speaker-based partitioning in ParCzech 3.0.

The quantitative effects of our processing are summarized in Figure 3.2. The left column of the figure presents statistics calculated on the dataset with zero edit distance, while the right column depicts the data after the cleaning. The initial dataset, as shown in the left column, predominantly consists of shorter segments, with approximately 40% containing five words or fewer. Furthermore, about a quarter of the data has a duration of less than one second. The subsequent filtering process involves discarding segments with inadequate textual and auditory context. This includes eliminating words with an average character duration below 0.015 or above 0.25, indicative of incorrect timestamp assignments, where character duration is measured by summing all word lengths in a segment a dividing this sum by the segment duration. For the test and validation sets, segments containing multiple speakers are excluded to ensure unique speakers not present in the training set. Conversely, for the training set, segments with multiple speakers are retained. Segments exceeding 15 seconds in duration are divided into smaller parts. This segmentation is governed by a binomial distribution with parameters $n = 9$ and $p = 0.6$, determining the length of each segment. The process involves sampling a new segment length and retaining the corresponding prefix as a new, shorter segment. This procedure is repeated until the remaining segment is less than 15 seconds long. Segments lacking punctuation or uppercase characters, except the initial position, are subsequently filtered out. Finally, all segments comprising fewer than five words or shorter than 1.5 seconds in duration are discarded.

The right column of Figure 3.2 displays the statistics of the dataset following this filtering process. The filtering effectively eliminates outliers in terms of average word and character durations. It is important to note that the average character duration depicted in the figure slightly deviates from the metric used in the filtering process, as the former includes gaps between words and is calculated at the segment level. Additionally, some discrepancies may arise due to the binomial distribution-based splitting of longer segments into shorter ones. The primary aim of this filtering approach is to remove edge cases arising from the data and speech recognition procedure while maintaining a data distribution that closely resembles the original.

Moving forward, we present an analysis of the dataset's distribution across training, testing, and validation sets, as detailed in Table 3.4. The comparison between the data volume before any processing, as shown in the first column of Table 3.3, and the final dataset reveals that the latter preserves 56.7% of the original audio's duration and 64.3% of the words. This reduction underscores the significant impact of employing a more advanced speech recognition model in the data processing pipeline, compared to the ASR model used in the original ParCzech 3.0. Specifically, the new speech recognition model contributes to a dataset that is not only larger in terms of audio duration and word count but also of a higher reliability, because we required the perfect alignment with the official transcript. It is important to note, however, that this comparison may not fully account for the constraints of the original ParCzech 3.0 dataset, which was limited to segments aligning with sentence boundaries. On the other hand, the current version of the dataset discards the segments without any casing or punctuation marks.

Table 3.4 further shows the distribution of data across the train, validation, and test sets, with 1711 hours for the training part and 15 and 13 hours for testing and validation, respectively. As indicated in Figure 3.2, approximately 35% of the dataset, by segment count, consists of segments shorter than 4 seconds, enabling models trained on this data to effectively process brief inputs, suitable for scenarios requiring rapid response. The inclusion of 30 unique and unseen speakers in each of the test and validation sets provides a robust basis for evaluating model performance on previously unheard speakers, ensuring a comprehensive assessment of the model's generalization capabilities.

We now examine the distribution of labels across the subsets, focusing on punctuation and case classes for each segment at the word level, as detailed in Table 3.5. The dataset exhibits a significant imbalance in both tasks, with a predominant share of labels falling under the `[blank]` class, indicating the absence of punctuation or capitalization. Specifically, for the train set 84.1% of labels are `[blank]`, with dots and

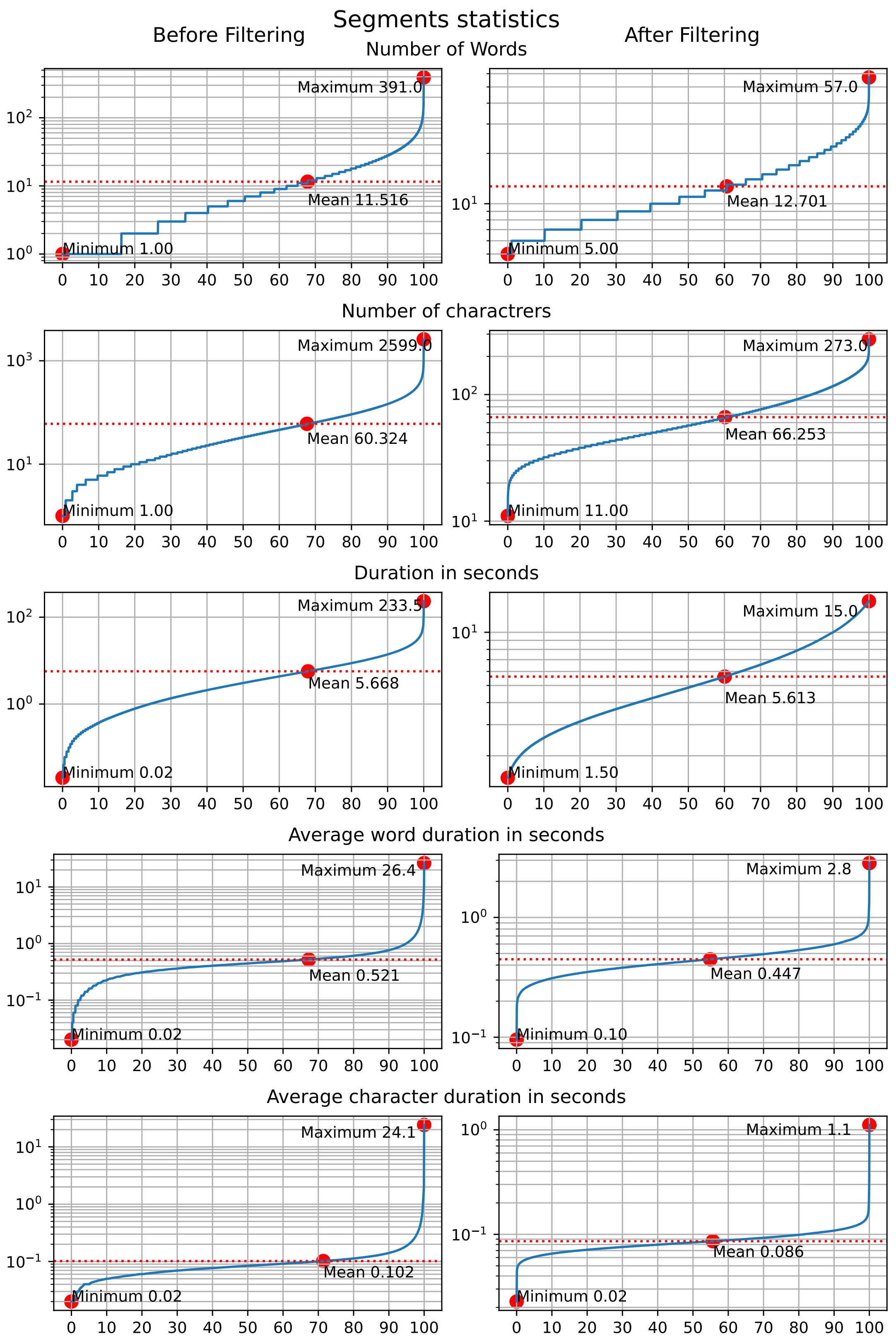


Figure 3.2: Statistical analysis of the dataset before and after the filtering process. All the statistics are collected on the set of segments. For each plot the corpus is sorted according to the selected statistic and the x axis presents percents of the segments. The statistics are displayed in log scale.

Set	Segments	Words	Duration (hours)	Speakers
Train	1 098 721	13 949 207	1711.43	415
Val.	9 657	126 874	15.62	30
Test	8 083	103 670	13.63	30
Total	1 116 461	14 179 751	1740.68	475

Table 3.4: Distribution of data across training, validation, and testing sets. The last row aggregates statistics for all the subsets.

Set	[blank]	[,]	[.]	[!]	[?]	[:]	[-]	[”]	[;]	[blank]	[cap]
Train	84.1	10.6	4.6	0.03	0.17	0.09	0.34	0.06	0.008	91.77	8.23
Val.	85.0	9.8	4.3	0.02	0.29	0.10	0.29	0.07	0.019	92.43	7.57
Test	86.0	9.2	4.2	0.01	0.14	0.08	0.25	0.10	0.026	92.30	7.70

Table 3.5: Distribution of punctuation and case labels across the train, validation, and test sets. The left part of the table represents label distribution for the punctuation restoration task and the two rightmost columns give label distribution for the case restoration task. The blank class is task specific. The percents are counted on the word level.

commas collectively accounting for about 15.2% of labels—commas alone constitute 10.6%. Consequently, the remaining six punctuation classes, including exclamation marks, question marks, colons, semicolons, quotation marks, and dashes, comprise a mere 0.7% of the dataset. Semicolons and exclamation marks are notably the rarest classes. In absolute terms, within the train set, around 4200 words are labeled with `[!]`, and merely 1110 words with `[;]`. Given the smaller size of the test set compared to the train set, it contains only 10 exclamation marks and 27 semicolons. The case restoration task presents a slightly more balanced scenario, with the capitalization class representing over 7.5% of words on average.

4. Training Speech Recognition Model

This section outlines the development of a custom speech recognition model, which will be utilized for generating datasets for punctuation and case restoration tasks, as well as for constructing a speech-informed model to address these tasks. The basis of this model is the unsupervised pre-trained wav2vec 2.0 base model, which comprises approximately 95 million parameters. To enhance speech recognition capabilities, this ASR model is integrated with a KenLM [Heafield, 2011] statistical N-gram language model within a beam search decoding framework. For convenience, we employ pre-trained weights for the wav2vec 2.0 model and utilize ASR datasets available through HuggingFace Datasets[1] [Lhoest et al., 2021].

4.1 Speech Recognition Data

For the speech recognition task, we leverage two datasets accessible via HuggingFace: the Czech version of the VoxPopuli dataset [Wang et al., 2021] and the Czech version of the Common Voice dataset [Ardila et al., 2020].

The Common Voice corpus is a multilingual collection of transcribed speech designed for research and development in speech technology, with a primary focus on speech recognition. This dataset is generated by contributors recording their voices as they read sentences displayed on the Common Voice website or application. These recordings are subsequently verified by other contributors through a voting mechanism. The latest release encompasses 29 languages, with data collection ongoing in 38 languages as of November 2019.

For our purposes, we only utilize the Czech portion of the Common Voice dataset, which is pre-segmented into training, validation, and testing subsets. The dataset statistics are presented in Table 4.1, indicating that approximately half of the dataset is allocated for training, with the remainder nearly equally split between validation and testing. Overall, the dataset comprises almost 40 hours of transcribed Czech speech, distributed across approximately 35 000 segments.

VoxPopuli is a comprehensive multilingual speech corpus designed for research in representation learning, semi-supervised learning, and speech interpretation. This dataset, derived from recordings of European Parliament events between 2009 and 2020, encompasses a wide range of activities including plenary sessions, committee meetings, and other parliamentary events. VoxPopuli provides an extensive collection of 400K hours of unlabeled speech data across 23 languages, rendering it an invaluable resource for unsupervised and semi-supervised learning methodologies. In addition to the unlabeled data, the corpus includes 1.8K hours of transcribed speeches in 15 languages, along with their oral interpretations into 15 target languages, totaling 17.3K hours of data.

The segmentation of speech data was performed using an energy-based voice activity detection algorithm. The dataset's compilation involved speaker diarization to amend inaccuracies in official timestamps and the application of automatic speech recognition

[1]https://huggingface.co/docs/datasets/index

Set	Segments	Duration in hours	Words
Train	14 612	19.19	115 739
Val	7 543	9.59	54 612
Test	7 714	9.66	54 276

Table 4.1: Statistics of the Czech Common Voice dataset.

(ASR) systems for force-alignment of speech segments with their transcriptions.

The processed Czech part of the VoxPopuli dataset, as outlined in Table 4.2, demonstrates the volume of data earmarked for training our ASR model. In comparison to the Common Voice dataset, VoxPopuli provides a significantly larger volume of transcribed data, both in terms of hours and word count. Notably, the increased data volume does not correspond to a higher segment count, indicating that VoxPopuli segments are, on average, longer both in duration and word count compared to those in Common Voice.

A key observation regarding VoxPopuli concerns its approach to digit normalization. The dataset offers two transcript variants for each entry: the original and a normalized version. Occasionally, the original transcript features numbers in their written form, while the normalized version may inappropriately convert all numbers to English. Moreover, VoxPopuli includes segments exceeding 68 seconds in length, that are not always suitable for training. Table 4.2 reports the data volume affected by English digit normalization and excessive segment length, with numbers in parenthesis indicating the impacted data volume. For training purposes, segments exceeding 24 seconds in length or featuring incorrect digit transcriptions were excluded. However, the validation set retains longer segments, excluding only those with improper digit normalization. The test set encompasses all data, including segments with English digit normalization, allowing for separate evaluation of model performance on segments with digits.

Furthermore, VoxPopuli contributes an additional 18.7K hours of unlabeled Czech speech data without transcripts, suitable for unsupervised ASR pretraining.

The final datasets for training, validation, and testing were compiled by merging the respective parts of the Common Voice and the refined VoxPopuli datasets. Additionally, all punctuation was removed, and texts were converted to lowercase across all dataset splits.

Set	Segments	Duration in hours	Words
Train	17 787 (886/275)	48.27 (3.58/2.17)	374 017
Val	1 022 (66/15)	2.69 (0.25/0.13)	20 946
Test	1 068 (44/12)	2.84 (0.17/0.08)	21 696

Table 4.2: The VoxPopuli dataset statistics after cleaning the data. The first number in the parenthesis displays the amount of data with incorrect normalization and the second one shows the amount of the data with long duration.

4.2 ASR Pipeline Training

The VoxPopuli corpus comes with ASR baselines for the 14 languages that have more than 10 hours of transcribed speech. This includes the Czech portion of the corpus. These ASR baselines utilize wav2vec 2.0 models with the original hyperparameter settings and are trained utilizing both the unsupervised wav2vec 2.0 loss and the supervised Connectionist Temporal Classification (CTC) loss. For the Czech language, both the unsupervised pre-trained model[2] and the CTC fine-tuned model[3] are available on Hugging Face. Each model is a base variant of the wav2vec 2.0 model with approximately 95M parameters.

The unsupervised model was pre-trained exclusively on 18.7K hours of unlabeled Czech data from the VoxPopuli corpus, learning robust contextualized representations of spoken Czech without exposure to any transcriptions. Conversely, the CTC fine-tuned model underwent pre-training on a 10K unlabeled subset of the VoxPopuli corpus before being fine-tuned on transcribed Czech data. Experimentation revealed that the primary limitation of the fine-tuned model stems from the normalization of digits in the Czech portion of the VoxPopuli corpus. Namely, due to the fine-tuning process employing digit normalization as English words, the model inconsistently recognizes spoken numbers in either English or Czech. This limitation significantly constrains the model's applicability in use cases involving the recognition of unlabeled data. However, initiating our efforts with the unsupervised model and subsequently training it on accurate transcripts presents a viable pathway forward.

4.2.1 Fine-Tuning the Unsupervised Model

Motivated by the preliminary findings, we start with fine-tuning the unsupervised model, which was pre-trained on 18.7K hours of unlabeled Czech audio from the VoxPopuli corpus. Without exposure to transcriptions, the model has previously only acquired an internal representation of speech. Our objective is to fine-tune this model using the CTC loss, employing a dataset previously constructed for this purpose.

We ran the training for 30 epochs with a batch size of 20, equating to an average of over one minute of audio per batch. Throughout the training process, the convolutional feature encoder remained fixed. A linear decay learning rate schedule was employed, starting with a peak learning rate of 4×10^{-5} and incorporating a warm-up phase over the initial 3000 steps. The last 12th layer of the wav2vec 2.0 model was reinitialized with random weights, adhering to a truncated normal distribution with a mean of 0.0 and a standard deviation of 0.02. Additionally, a linear head was appended atop the wav2vec 2.0 model to map the latent frame representations to the CTC vocabulary.

Figure 4.1 details the model training progress, with the upper graph depicting the trajectory of CTC loss across both training and validation datasets. Notably, the training loss demonstrates a consistent decline over the course of the training, whereas the validation loss reaches its local minimum within the first 8,000 updates before diverging — an indicator of potential overfitting to the training data. However, the overfitting is not confirmed by the lower graph, which shows the model's performance on the validation set in terms of Word Error Rate (WER). Utilizing greedy decoding for faster evaluation, a continual decrease in WER is observed, signifying that the model

[2]`https://huggingface.co/facebook/wav2vec2-base-cs-voxpopuli-v2`

[3]`https://huggingface.co/facebook/wav2vec2-base-10k-voxpopuli-ft-cs`

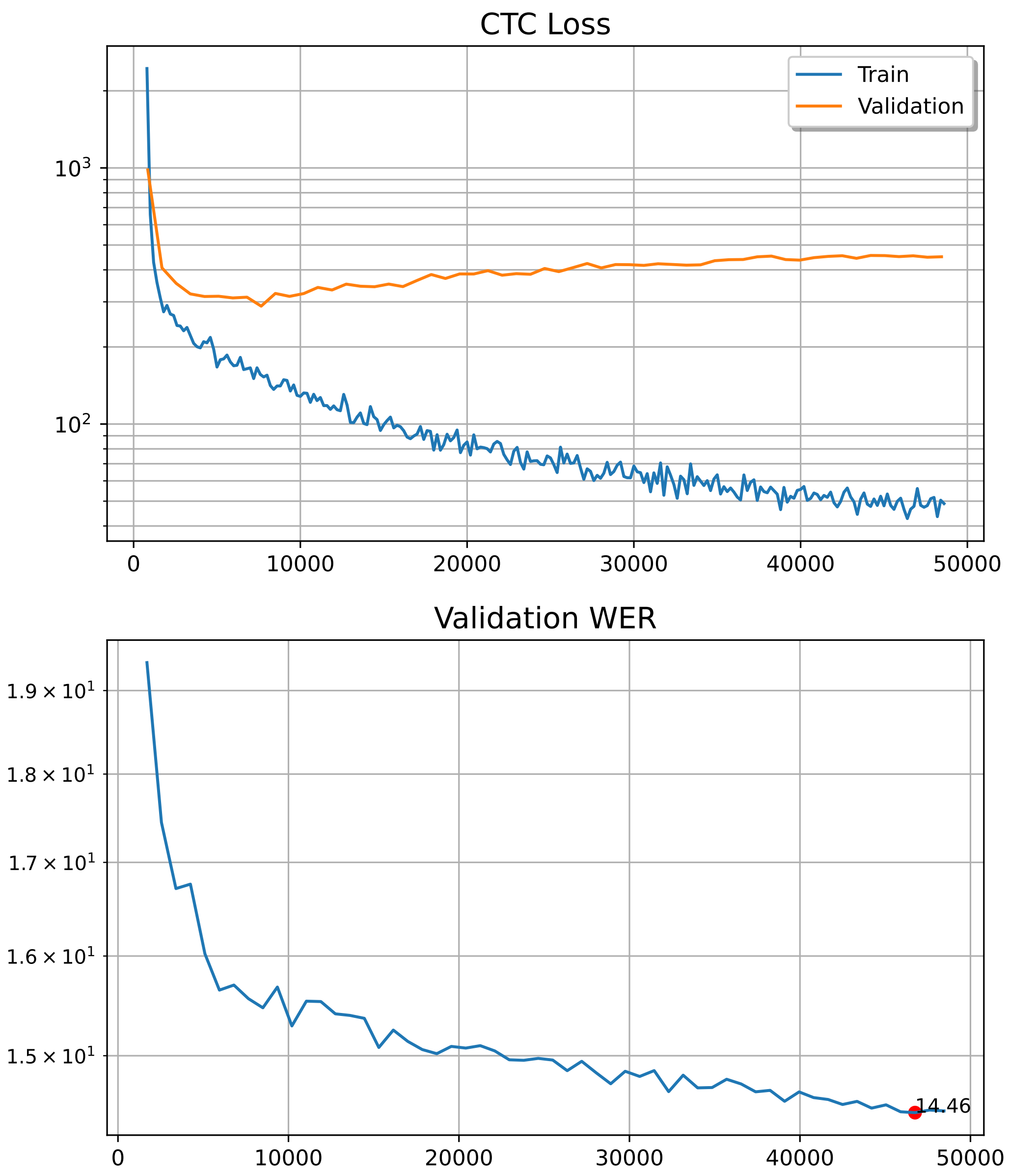


Figure 4.1: Dynamics of the ASR training metrics, illustrating the change of CTC loss and Word Error Rate (WER) over training and validation sets.

does not overfit and maintains generalizability across the validation set. Furthermore, the WER trajectory suggests that extended training could yield further enhancements. The minimal WER of 14.46 was achieved during the last updates, suggesting that longer training might lead to better model performance.

4.2.2 KenLM Training

Improving the performance of ASR models can be effectively achieved by employing beam search in conjunction with a language model for rescoring ASR hypotheses. In this context, we utilize open-source implementation[4] of KenLM, a word n-gram language model renowned for its efficiency. KenLM estimates the probability of an n-gram sequence and incorporates backoff-smoothed models, where the probability of an n-gram is determined based on the longest observed contiguous history.

For training the KenLM model, we utilize transcripts from the training subset of our dataset, augmented by the complete official transcripts of ParCzech 3.0. As detailed in the Data chapter, transcripts from ParCzech 3.0 were processed using UDPipe 2.0, facilitating sentence segmentation. Subsequently, all punctuation marks were removed from these sentences, reflecting the absence of punctuation in recognized transcripts. The integration of ParCzech data yields a total of 1,391,785 segments encompassing 22,104,659 words, resulting in a large training corpus for the KenLM model consisting of 1,409,572 segments and 22,478,676 words.

Beam search with KenLM is implemented using the `pyctcdecode` open-source package.[5] To refine model performance, we conduct a hyperparameter grid search aimed at minimizing the Word Error Rate (WER) on the validation set. The search encompasses four parameters: beam width $bw \in \{150, 200, 250\}$, language model weight $\alpha \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$, length score adjustment $\beta \in \{0.33, 0.66, 1, 1.33, 1.66, 2, 3\}$, and the order of n-grams for KenLM $n \in \{2, 3\}$. Our experiments identify the configuration $bw = 250$, $\alpha = 0.5$, $\beta = 0.66$, and $n = 3$, which effectively reduces the WER on the validation set to 11.7%.

4.3 Evaluation

In this section, we evaluate the performance of our custom fine-tuned model on the test set using both greedy and LM-based beam search decoding strategies. Additionally, we compare it to the original fine-tuned VoxPopuli model. The test subset created in the Data section serves as our evaluation dataset.

Table 4.3 presents the WER of the models on the test set, excluding the segments with English-normalized numbers. Our fine-tuned ASR model, which shares the wav2vec 2.0 architecture and decoding strategy with the VoxPopuli Czech model, demonstrates a significant performance improvement. Specifically, under greedy decoding, our model achieves nearly halves WER compared to the VoxPopuli model. The implementation of LM-based beam search decoding further enhances performance, reducing WER by 19.5% against the greedy decoding strategy. Notably, performance on the test set is marginally lower than on the validation set. This discrepancy can be attributed to the optimization of the model using the validation set, through selecting the best checkpoint and configuring beam search decoding, which led to slight overfitting to the validation data. The test set, having not been used in the model improvement process, provides a more reliable assessment.

Since we excluded segments with English-normalized numbers from our evaluation, we can now visually compare the performance of different models on test segments that contain numbers verbalized in English. Table 4.4 shows same parts of recognized

[4] `https://github.com/kpu/kenlm`
[5] `https://github.com/kensho-technologies/pyctcdecode`

Metric	VoxPopuli Greedy	Custom Greedy	Custom Beam LM
WER	30.98	15.52	**12.49**

Table 4.3: Comparison of ASR models performance on the previously created test set. VoxPopuli Greedy is CTC fine-tuned model from the VoxPopuli corpus, Custom Greedy is our trained model with greedy decoding strategy and Custom Beam LM is the same model with beam search and KenLM rescoring.

Models	Texts
Orig	ten je již vězněn fourteen měsíců a je obviněn z údajné podpory ...
VPGr	ten jejiž vězněn čtrnáct měsíců a je obviněn z údajné podpory ...
CusGr	ten je již vězněn čtrnáct měsíců a je obviněn z údajné podpory ...
CusB	ten je již vězněn čtrnáct měsíců a je obviněn z údajné podpory ...
Orig	... opětovné využití a recyklaci na seventy celkového ...
VPGr	... opětovné využití a recyklaci na sevenstyn celkového ...
CusGr	... opětovné využití a recyklaci na sedmdesát procent celkového ...
CusB	... opětovné využití a recyklaci na sedmdesát procent celkového ...
Orig	... vyplývá že víc než ninety respondentů chce aby byl původ ...
VPGr	... vyplývá že víclež ninty spondentů chce aby byl původ ...
CusGr	... vyplývá že více než devadesát respondentů chce aby byl původ ...
CusB	... vyplývá že více než devadesát respondentů chce aby byl původ ...
Orig	... text směrnice z roku two thousand and four je třeba modernizovat ...
VPGr	... text směrnice z roku two thousand and four je třeba modernizovat ...
CusGr	... text směrnice z roku dvatisíce čtyři je třeba modernizovat ...
CusB	... text směrnice z roku dvatisíce čtyři je třeba modernizovat ...

Table 4.4: Predictions of different models on the examples with detected English digit normalization. The first row in each example is the original normalized transcript from VoxPopuli corpus. VPGr is the Czech fine-tuned VoxPopuli model with greedy decoding, CusGr is our custom fine-tuned model with greedy decoging and CusB is the same model with LM-fused beam search decoding. The examples can be cut on both sides, to better fit the table.

transcripts for different models, with the original transcript displayed in the first row. Evaluating on the subset with identified incorrect digit verbalization, we noted that the fine-tuned VoxPopuli model tends to misinterpret Czech pronounced numbers as English words. In contrast, both our models, regardless of the decoding strategy employed, accurately recognize and transcribe numbers in Czech. Notably, the first and third examples highlight minor errors in the VoxPopuli model, whereas our models, employing both beam search and greedy decoding strategies, exhibit correct recognition across all provided examples. This observation underscores our models' capability in correct number recognition.

In conclusion, we have successfully trained an ASR model employing LM-fused beam search decoding that markedly surpasses the baseline provided by the VoxPopuli corpus in performance. This model holds potential for further application in speech recognition tasks and can serve as an effective sound encoder when integrating text and sound modalities.

5. Experiments

In this chapter, we dive into the process of utilizing audio data for punctuation prediction and capitalization. The first step involves the creation of a textual baseline using RobeCzech model. This baseline is then progressively transformed into a multimodal model that integrates both audio and textual data. During the construction of this multimodal neural network, a comparative analysis of various models is performed on the development set. Finally, the evaluation is performed on the test set.

To gain an understanding of the final model that will be developed through the forthcoming experiments, a visual representation of the resulting neural network is provided on Figure 5.1.

5.1 General Setup

As stated above the model will be sequentially developed from a textual baseline, however some of the model architectural and parameter choices will be fixed for all the experiments. All components were implemented in Python using PyTorch [Paszke et al., 2019] with PyTorch Lightning [Falcon and The PyTorch Lightning team, 2019] for multi-gpu scaling.

Each model that will occur in the experiments will implement a linear decay with linear learning rate warm-up. During the initial 20% of steps, the learning rate will progressively rise to its peak before reducing to zero. All models will be trained on two GPUs, each with a minimum of 16GB RAM, giving a batch size of 200 examples per GPU, thus achieving an effective batch size of 400. Models with this effective batch size will employ a peak learning rate of 6.0×10^{-5} and will be trained for 15 epochs.

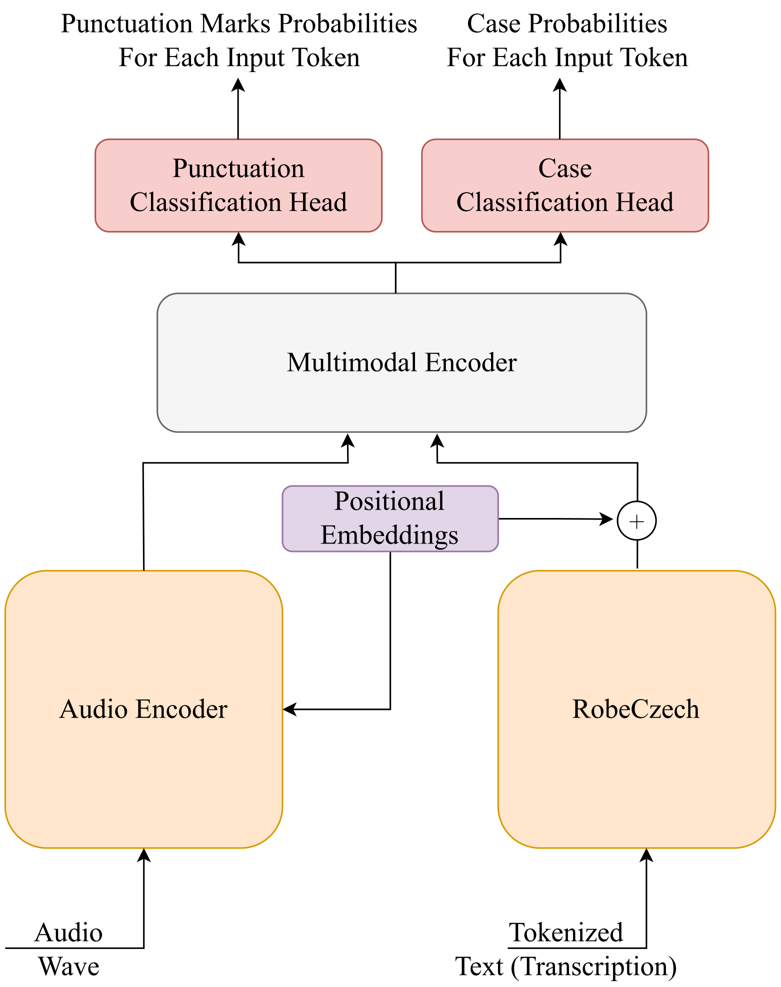
The training of all models will be performed with the sum of cross-entropy losses, denoted as $\mathcal{L} = 0.5\mathcal{L}_p + 0.5\mathcal{L}_c$, where $\mathcal{L}_p$ represents the punctuation loss averaged over the batch size and sequence length, and $\mathcal{L}_c$ signifies the case loss, averaged in the same manner.

Each model will feature two classification heads, one for punctuation classification and the other for case classification. The architecture of these heads will be identical, consisting of a dropout layer with a dropout probability of 0.2, a linear layer of size 768 followed by a tanh activation function, and a final linear layer of size determined by the number of classes for each task. With this architecture, each classification head has approximately 590,000 trainable parameters.

5.2 Metrics

In order to compare our models, we will use a range of different metrics. Firstly, the unweighted average F1 metric will be used independently for punctuation prediction and case prediction. This involves calculating the F1 metric for each class of a given task and then taking an unweighted average. An unweighted method is used to prevent metric boosting by most frequent classes.

Additionally, we will define the Punctuation Detection Precision (PDP) and Punctuation Detection Recall (PDR) metrics. These metrics simulate precision and recall performance on a binary classification task where the model must identify potential



Punctuation Marks Probabilities For Each Input Token
Case Probabilities For Each Input Token
Punctuation Classification Head
Case Classification Head
Multimodal Encoder
Positional Embeddings
+
Audio Encoder
RobeCzech
Audio Wave
Tokenized Text (Transcription)

Figure 5.1: Final structure of the multimodal model, which integrates both audio and textual inputs to generate outputs for punctuation and case classes.

punctuation placement locations. Specifically, in both of these tasks, any non-blank punctuation label is considered as the positive class resulting in binary labels. The task is to detect the punctuation locations without considering punctuation mismatches. Using the binary labels, the PDP reduces to the classical precision formulation, i.e., $\mathrm{PDP} = \frac{\mathrm{TP}}{\mathrm{TP+FP}}$, and for the PDR, we have $\mathrm{PDR} = \frac{\mathrm{TP}}{\mathrm{TP+FN}}$, where TP stands for true positives, FP for false positives, and FN for false negatives. Assuming that model performance is highly influenced by the available textual context, PDP and PDR will be also evaluated on the shortest 25% of the sequences and the longest 25% of the sequences.

In addition to these metrics, we will employ non-numeric methods such as plotting attention matrices and showing models' predictions in the textual form.

These methods will collectively provide a comprehensive evaluation of each model's performance. All metrics will be computed on the validation set every 500 training updates.

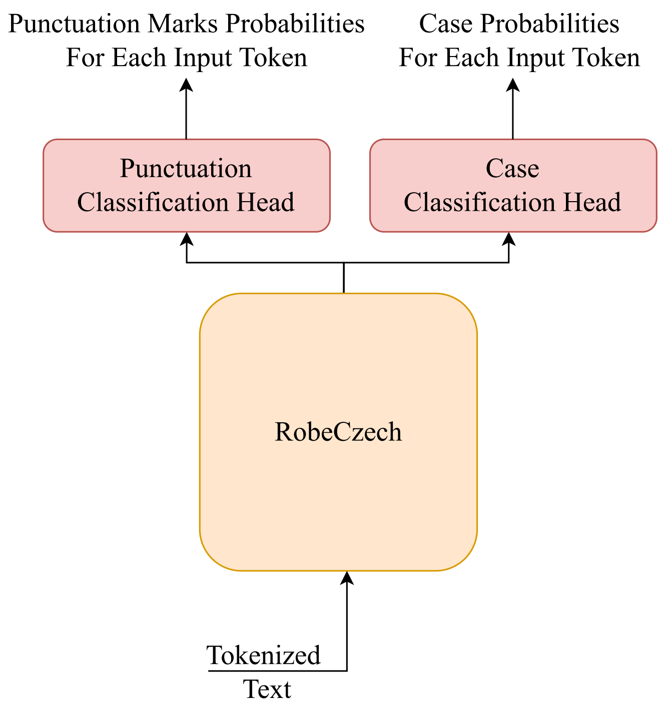


Figure 5.2: The text-only baseline architecture, which contains a pre-trained RobeCzech model along with two classification heads.

5.3 Text-Only Baseline

We create a text-only baseline using a method known as fine-tuning. In essence, fine-tuning is a process that begins with a model that has undergone pre-training on a broad task, such as next token prediction or masked language modeling. In natural language processing this pre-training phase enables the model to construct a robust internal representation of the written language.

Following this, the general-purpose language model is trained on a specific task. In our context, the specific task is punctuation and case restoration. The primary advantage of this approach lies in its efficiency. After the pre-training phase, the model requires fewer data and update steps to achieve satisfactory results when training on the specific task.

The structure of the text baseline, as illustrated in Figure 5.2, is based on the RobeCzech model, which is a monolingual RoBERTa language representation model that has been trained on Czech data. As previously mentioned, the model incorporates two classification heads, one for each task. The training process contains of 15 epochs, with a peak learning rate of 6.0×10^{-5}. Under this configuration, the text-only baseline has approximately 125×10^6 parameters for the RobeCzech model and around 1×10^6 parameters for both classification heads.

Figure 5.3 presents different losses throughout the training phase of the model. As previously indicated, the average loss is an average of punctuation and case losses. Additionally, it is observed that the overall training loss decreases faster than the validation loss, maintaining a steady gap throughout the training process. Typically, such a gap might imply the overfitting of the model, and we will revisit this discussion about overfitting later, once we present model performance on other metrics. This gap could potentially be attributed to the inadequate size of the text training corpus, resulting in

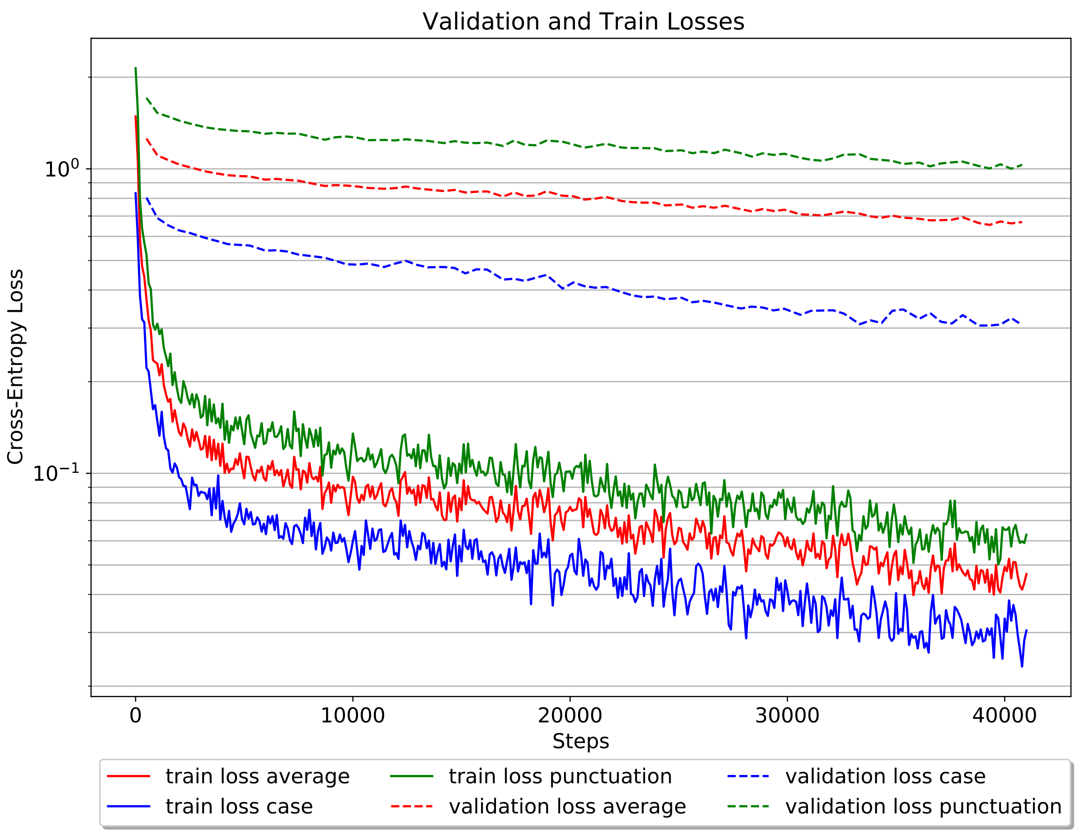


Figure 5.3: Plot with loss curves of the text-only baseline model.

certain lexical patterns present in the validation set not being sufficiently represented in the training data. Nonetheless, the uniform decline in both losses indicates that overfitting does not prevent overall successful training. Interestingly, this plot confirms the intuitive observation that restoring punctuation is more challenging than restoring the correct casing. Evidently, case restoration encounters difficult samples tied to Named Entity Recognition (NER), but punctuation restoration also faces challenging examples, such as exclamation marks, quotation marks, and semicolons. The analysis of this loss plot suggests that the text-only baseline continues to learn even beyond 15 epochs, with the potential to converge towards better results.

5.3.1 Punctuation Restoration

As illustrated in Figure 5.4, we evaluate the model's performance in terms of punctuation detection through three separate evaluations: on the shortest 25% of sequences, the longest 25% of sequences, and the entire validation set. Generally, a consistent pattern is observed where recall continually improves over the first $20,000$ steps. Once the recall peaks, precision begins a slight decline, with a one-point drop at the end of the training compared to its maximum value. It is also noticed that when precision reaches its highest points, recall often drops, and the reverse is also true. Both Punctuation Detection Precision (PDP) and Punctuation Detection Recall (PDR) metrics highlight the importance of context length in punctuation detection; the model demonstrates better performance with a longer context, a trend that is consistent throughout the training process. In general, shorter sequences pose a greater challenge for the

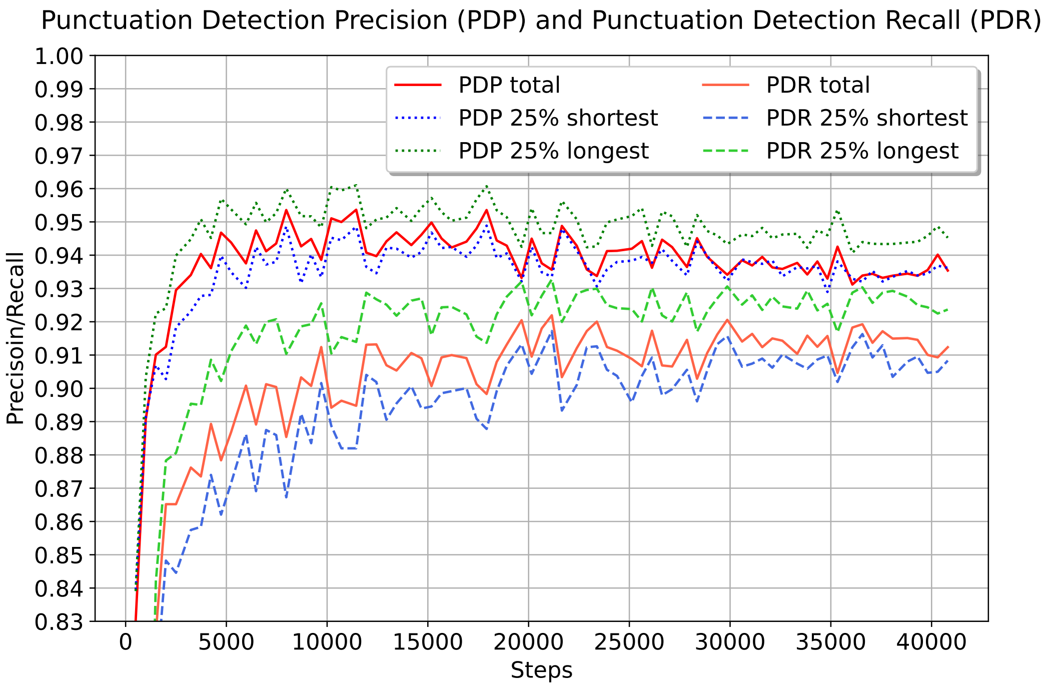


Figure 5.4: Performance of the text-only baseline on the punctuation detection task in terms of precision and recall.

model. However, when comparing precision for long and short sequences, the difference is small compared to recall, where performance on shorter sequences is significantly poorer. This pattern is easily explained for by the observation that, once the model detects punctuation placement, it is generally correct, with a higher error rate for shorter sequences. Nonetheless, punctuation detection can be challenging, and additional context can be beneficial. When we look at the numbers, the best combination of PDP and PDR, 0.91 and 0.945 respectively, was attained around the 19,000th step. Overall, it can be concluded that punctuation detection is not a very difficult task, as the model demonstrates good performance while being trained with different objectives.

The analysis of the punctuation restoration task continues by referring to Figure 5.5. This figure provides a detailed evaluation of the model's performance on each punctuation mark, in terms of the F1 score. The plot shows that the model learns different classes in sequence, starting with the most frequent and well-represented, and finishing with the most challenging ones. This can be easily explained by the punctuation class imbalance in the training data outlined in Table 3.5, specifically, model rapidly learns to detect commas and periods which are the most frequent classes, and only after approximately 8,000 steps starts to recognize classes with the lowest frequency, namely exclamation marks and semicolons. However, after approximately $25,000$ or $30,000$ training steps, the model's performance slightly declines for less exemplified punctuation marks, including quotations, dashes, semicolons, and even periods. However, the non-blank average remains steady, staying at a level of 0.45 after about $15,000$ steps. Considering the model's performance on various metrics, such as PDP, PDR and F1 scores, it is noted that after approximately $20,000$ steps, the performance either maintains or slightly decreases. A revisit to the punctuation task validation in Figure 5.3 reveals the consistent decline in loss throughout the training process. This difference

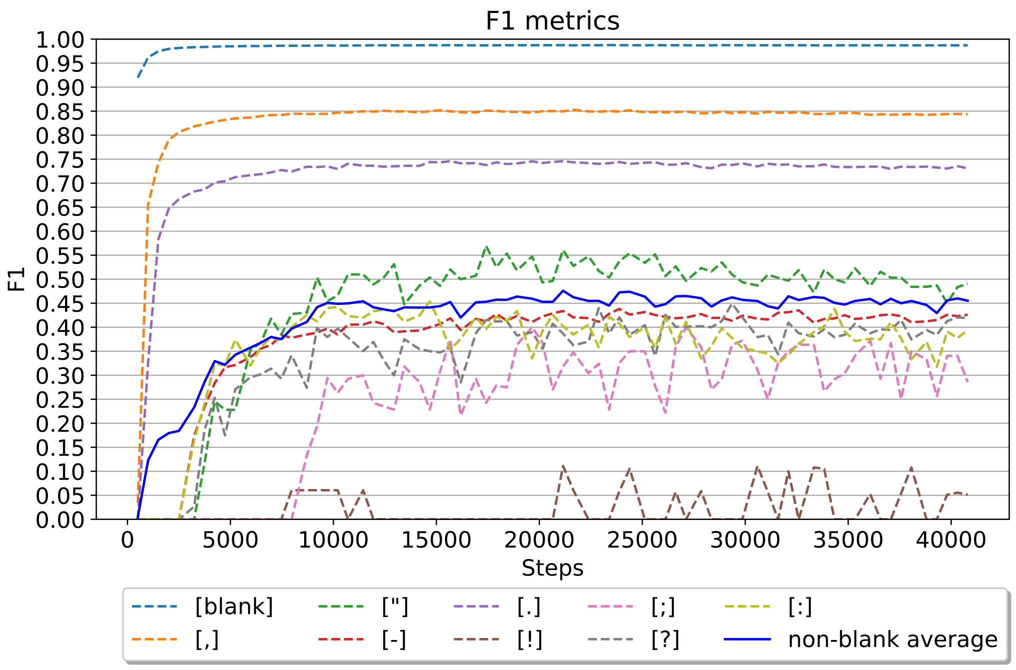


Figure 5.5: The performance of the text-only model, evaluated in terms of F1 scores for each punctuation mark.

suggests that the cross-entropy loss may not perfectly correspond with the actual model performance. For instance, in a scenario where the true label is `[A]` and the model predicts `[B]`, the model can still reduce its cross-entropy loss by slightly redistributing scores among classes, while retaining `[B]` as its most probable prediction. In this case, the metrics remain unchanged, but the loss decreases.

Figure 5.6 presents various confusion matrices for punctuation restoration during the training process. In these matrices, the rows sum to 100%, representing the distribution of model predictions given true punctuation labels. The steps were chosen to demonstrate the change of model performance during training. Initially, it is observed that during the first $10,000$ steps, the model progressively learns to distinguish between different punctuation marks, commencing with the most common and concluding with the most rare and challenging. Unsurprisingly, the most difficult punctuation mark to infer is the exclamation mark, which is essentially challenging to predict without the corresponding audio inputs and a sufficiently long textual context. As might be expected, exclamation marks are frequently misclassified as periods, which can be indistinguishable for some inputs without audio. Another challenging case for the text-only baseline involves question marks, which are also frequently misclassified as periods. This could be due to the limited context, particularly in situations where the beginning of the sentence is missing. Furthermore, the free word order in Czech does not indicate questions as clearly as it does in English. Throughout training, the model constructs robust representations for the most popular punctuation classes, such as periods and commas, and even in the early stages of training, the model can differentiate between these two classes. Notably, as commas are the second most common punctuation class in the data, the model continues to learn to distinguish commas from periods. However, for periods, the situation is the opposite, and as training progresses, the model increasingly misclassifies periods as commas. In general, it can be concluded that model performance stabilizes after $15,000$ to $20,000$ steps, with minor improvements thereafter. This generally aligns with the observations from Figure 5.4, where a similar stabilization in performance is seen after $20,000$ steps.

5.3.2 Capitalization

In regards to the case restoration task, we can perform an equivalent analysis using confusion matrices and F1 scores. Essentially the task corresponds to binary classification with highly imbalanced classes.

We start with the F1 scores as illustrated in Figure 5.7. It is no surprise that the model rapidly learns to detect uppercase words within the initial 5,000 steps and retains this local optimum for approximately the next 20,000 steps. Similar the punctuation restoration task, we observe a small decrease in metric performance after 25,000 steps.

To verify our assumptions on the model's performance in the case restoration task, we refer to Figure 5.8, which plots various confusion matrices for selected iterations. Notably, even after 2,000 updates, the model manages to learn a valuable internal representation of the capitalization class. As previously observed, starting from the 5,000th step, the performance does not alter significantly, with a minor decline after 22,000 iterations.

Given the two previous observations, we can conclude that the model reaches peak performance approximately at the 22,000th training step.

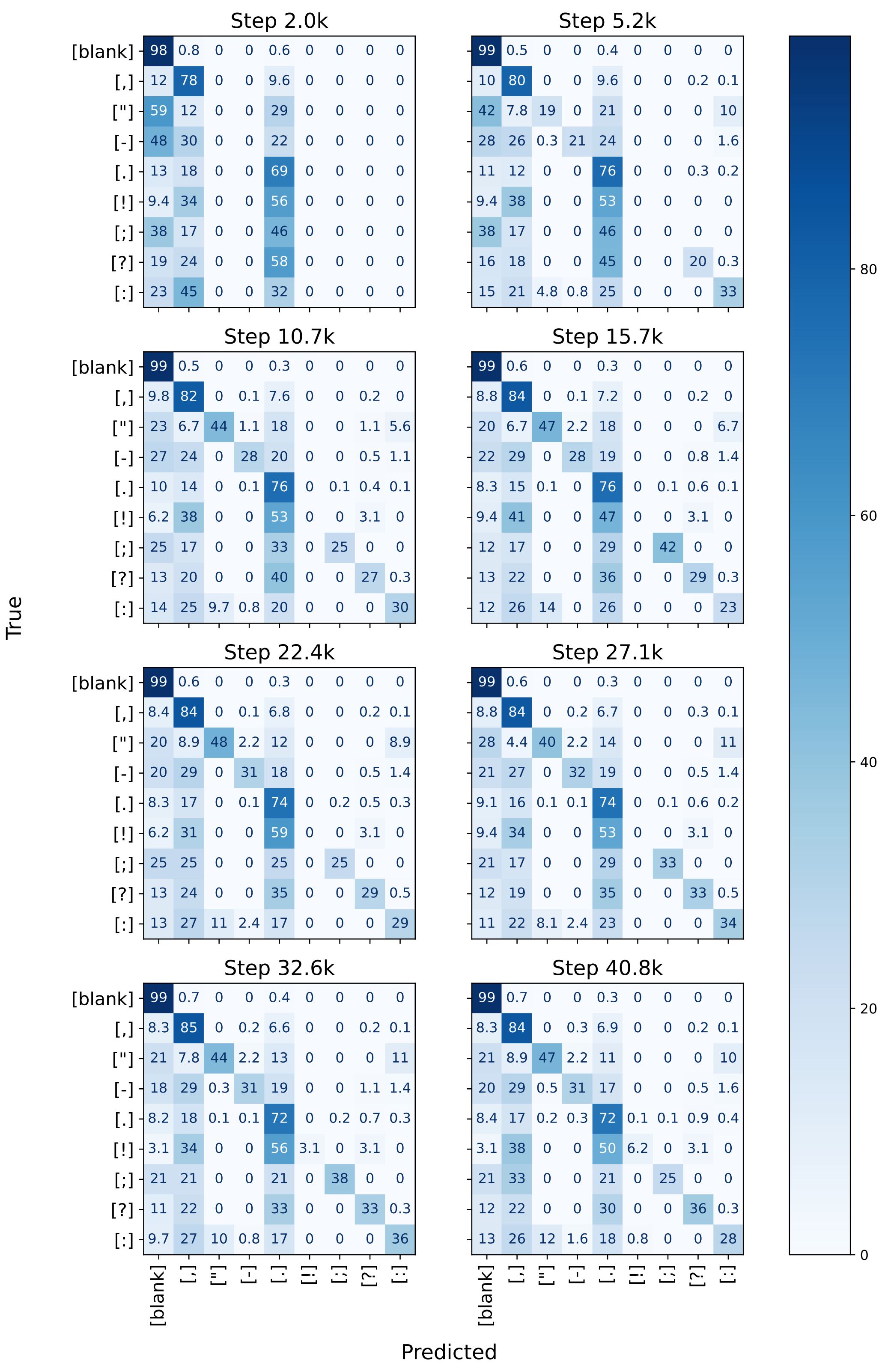
Revisiting Figure 5.3, we can now observe that even though the gap between the training loss and validation loss is visible from the initial training steps, slight overfitting only emerges after the 25,000th step. However, we observe that this overfitting arises more from the inherent task structure than from the model employed. The high imbalance towards the `[blank]` class complicates the learning of useful representations for other classes.

5.4 Multimodal Encoder

The most straightforward method of integrating audio information with textual input is to manually construct a set of features and append them to the corresponding words. This strategy, however, has several drawbacks. The primary limitation is that it demands expert knowledge in audio processing and laborious feature engineering. Moreover, given the alignment of words and their audio representations, we may risk losing global context in the audio; for instance, consider questions where the intonation gradually rises towards the end of the sentence. Finally, appending audio features to the textual input may harm pretraining approach.

A way to bypass feature engineering and allow the model to discern the pertinent factors is to employ an attention mechanism. This approach can be seen as a more generalized method of fusing different modalities, where the model is free to autonomously generate useful features without relying on manually extracted information. Furthermore, this setup enables the model to attend to every sound feature in the audio input, not just the one corresponding to the current word.

Our multimodal encoder block takes inspiration from the decoder block in the original Transformer architecture, particularly the cross-attention component of this block. In cross-attention, the keys and values stem from the encoder portion and queries are provided by the decoder block. However, in our context, we lack a decoding part;



Step 2.0k

	[blank]	[,]	["]	[-]	[.]	[!]	[;]	[?]	[:]
[blank]	98	0.8	0	0	0.6	0	0	0	0
[,]	12	78	0	0	9.6	0	0	0	0
["]	59	12	0	0	29	0	0	0	0
[-]	48	30	0	0	22	0	0	0	0
[.]	13	18	0	0	69	0	0	0	0
[!]	9.4	34	0	0	56	0	0	0	0
[;]	38	17	0	0	46	0	0	0	0
[?]	19	24	0	0	58	0	0	0	0
[:]	23	45	0	0	32	0	0	0	0

Step 5.2k

	[blank]	[,]	["]	[-]	[.]	[!]	[;]	[?]	[:]
[blank]	99	0.5	0	0	0.4	0	0	0	0
[,]	10	80	0	0	9.6	0	0	0.2	0.1
["]	42	7.8	19	0	21	0	0	0	10
[-]	28	26	0.3	21	24	0	0	0	1.6
[.]	11	12	0	0	76	0	0	0.3	0.2
[!]	9.4	38	0	0	53	0	0	0	0
[;]	38	17	0	0	46	0	0	0	0
[?]	16	18	0	0	45	0	0	20	0.3
[:]	15	21	4.8	0.8	25	0	0	0	33

Step 10.7k

	[blank]	[,]	["]	[-]	[.]	[!]	[;]	[?]	[:]
[blank]	99	0.5	0	0	0.3	0	0	0	0
[,]	9.8	82	0	0.1	7.6	0	0	0.2	0
["]	23	6.7	44	1.1	18	0	0	1.1	5.6
[-]	27	24	0	28	20	0	0	0.5	1.1
[.]	10	14	0	0.1	76	0	0.1	0.4	0.1
[!]	6.2	38	0	0	53	0	0	3.1	0
[;]	25	17	0	0	33	0	25	0	0
[?]	13	20	0	0	40	0	0	27	0.3
[:]	14	25	9.7	0.8	20	0	0	0	30

Step 15.7k

	[blank]	[,]	["]	[-]	[.]	[!]	[;]	[?]	[:]
[blank]	99	0.6	0	0	0.3	0	0	0	0
[,]	8.8	84	0	0.1	7.2	0	0	0.2	0
["]	20	6.7	47	2.2	18	0	0	0	6.7
[-]	22	29	0	28	19	0	0	0.8	1.4
[.]	8.3	15	0.1	0	76	0	0.1	0.6	0.1
[!]	9.4	41	0	0	47	0	0	3.1	0
[;]	12	17	0	0	29	0	42	0	0
[?]	13	22	0	0	36	0	0	29	0.3
[:]	12	26	14	0	26	0	0	0	23

Step 22.4k

	[blank]	[,]	["]	[-]	[.]	[!]	[;]	[?]	[:]
[blank]	99	0.6	0	0	0.3	0	0	0	0
[,]	8.4	84	0	0.1	6.8	0	0	0.2	0.1
["]	20	8.9	48	2.2	12	0	0	0	8.9
[-]	20	29	0	31	18	0	0	0.5	1.4
[.]	8.3	17	0	0.1	74	0	0.2	0.5	0.3
[!]	6.2	31	0	0	59	0	0	3.1	0
[;]	25	25	0	0	25	0	25	0	0
[?]	13	24	0	0	35	0	0	29	0.5
[:]	13	27	11	2.4	17	0	0	0	29

Step 27.1k

	[blank]	[,]	["]	[-]	[.]	[!]	[;]	[?]	[:]
[blank]	99	0.6	0	0	0.3	0	0	0	0
[,]	8.8	84	0	0.2	6.7	0	0	0.3	0.1
["]	28	4.4	40	2.2	14	0	0	0	11
[-]	21	27	0	32	19	0	0	0.5	1.4
[.]	9.1	16	0.1	0.1	74	0	0.1	0.6	0.2
[!]	9.4	34	0	0	53	0	0	3.1	0
[;]	21	17	0	0	29	0	33	0	0
[?]	12	19	0	0	35	0	0	33	0.5
[:]	11	22	8.1	2.4	23	0	0	0	34

Step 32.6k

	[blank]	[,]	["]	[-]	[.]	[!]	[;]	[?]	[:]
[blank]	99	0.7	0	0	0.4	0	0	0	0
[,]	8.3	85	0	0.2	6.6	0	0	0.2	0.1
["]	21	7.8	44	2.2	13	0	0	0	11
[-]	18	29	0.3	31	19	0	0	1.1	1.4
[.]	8.2	18	0.1	0.1	72	0	0.2	0.7	0.3
[!]	3.1	34	0	0	56	3.1	0	3.1	0
[;]	21	21	0	0	21	0	38	0	0
[?]	11	22	0	0	33	0	0	33	0.3
[:]	9.7	27	10	0.8	17	0	0	0	36

Step 40.8k

	[blank]	[,]	["]	[-]	[.]	[!]	[;]	[?]	[:]
[blank]	99	0.7	0	0	0.3	0	0	0	0
[,]	8.3	84	0	0.3	6.9	0	0	0.2	0.1
["]	21	8.9	47	2.2	11	0	0	0	10
[-]	20	29	0.5	31	17	0	0	0.5	1.6
[.]	8.4	17	0.2	0.3	72	0.1	0.1	0.9	0.4
[!]	3.1	38	0	0	50	6.2	0	3.1	0
[;]	21	33	0	0	21	0	25	0	0
[?]	12	22	0	0	30	0	0	36	0.3
[:]	13	26	12	1.6	18	0.8	0	0	28

Figure 5.6: Confusion matrices for text-only model on the punctuation classes. The chosen steps serve to better depict the different phases of model training.

instead, we have two encoders one for audio and one for text. Given that punctuation and case labels are naturally tied to the text rather than the sound, queries in the

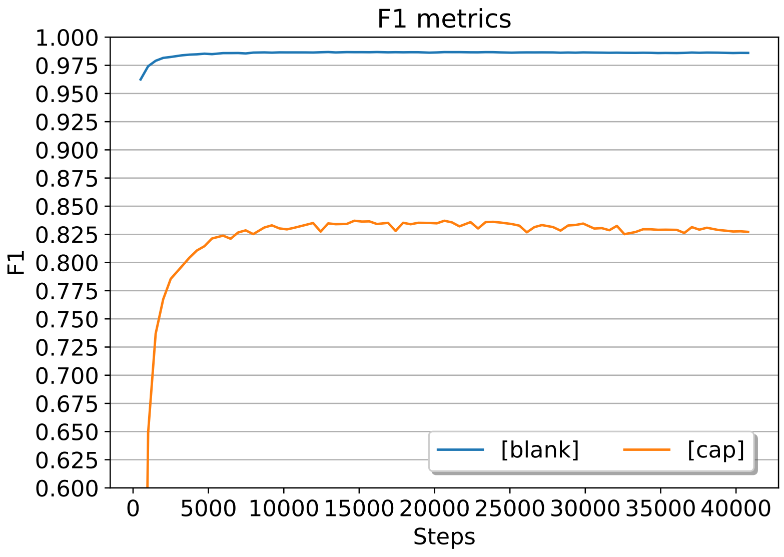


Figure 5.7: The performance of the text-only model, evaluated in terms of F1 scores for case restoration task.

multimodal cross-attention will come from the text part, while keys and values will be supplied by the sound. The intention is that for certain ambiguous cases in text, the model can seek additional information in the audio input. Also, since we do not perform decoding, we can avoid causal attention masking in both attention parts. This architectural decision imposes limitations on the online application of the multimodal model, when the input is still growing. In practice, we can simulate the online scenario by using a small buffer to balance performance and latency. The architecture of the multimodal encoder block is illustrated on Figure 5.9.

Before starting experiments that combine audio and text, we need to set the parameters for the multimodal encoder. We start with setting the number of heads for both the multi-head self-attention and multi-head multimodal cross-attention at 8. Furthermore, we need to specify the dimensions of the key, value, and query projection matrices in the multi-head self-attention layer, which will be identical to the dimensions of RobeCzech, specifically 768. For the sake of simplicity, the dimensions for the multi-head multimodal cross-attention will be kept consistent with those of the multi-head attention layer.

For data normalization, we have opted for Layer Normalization with $\epsilon = 6.0 \times 10^{-5}$, mirroring the ϵ value used in RobeCzech. The feed forward layer in the encoder is a compact fully connected network that includes a linear layer that projects the incoming tensors from 768 to 2048 dimensions. This is followed by dropout [Srivastava et al., 2014] with a probability of 0.2 and the application of the GELU [Hendrycks and Gimpel, 2023] activation function. Finally, a reverse linear projection is performed to revert the tensors to 768 dimensions.

All add-norm blocks in the encoder utilize skip connections. The previous state is initially passed through a dropout layer with a probability of 0.2 and then added to the current state. The result of this addition is subsequently normalized using Layer Normalization with the same ϵ.

The multimodal encoder will consist of two stacked blocks. In total, the multimodal encoder comprises two stacked blocks, which yields approximately 20×10^6 parameters.

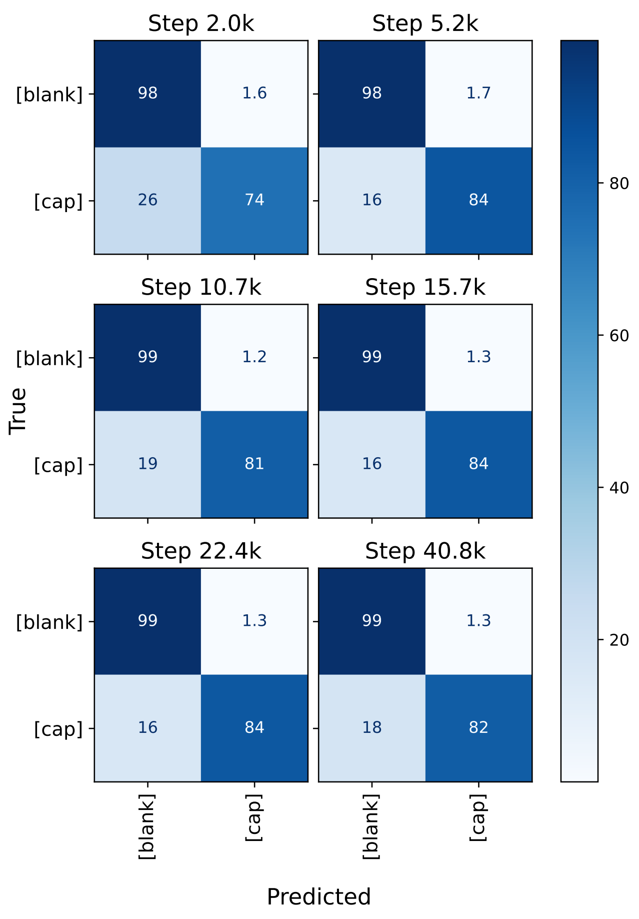


Figure 5.8: Case restoration confusion matrices for text-only model. The chosen steps serve to visually depict the different phases of model training.

It is crucial to note, that the architecture and parameters of the multimodal encoder will remain constant throughout the forthcoming experiments.

5.5 Audio Encoder Baseline

The first attempt of integration of audio into the multimodal model is achieved by employing a trained Automatic Speech Recognition (ASR) model as an audio encoder. For the creation of our corpus, we had previously trained a wav2vec 2.0 speech recognition model in Chapter 4, which we now incorporate into our first multimodal neural network. At this stage both of the input encoders have been pretrained on previous tasks and are now fine-tuned for the task of punctuation and case prediction. This multimodal baseline solution has 126×10^6 parameters for the RobeCzech model with two classification heads, 20×10^6 parameters for the multimodal encoder, and approximately 95×10^6 parameters for the wav2vec 2.0 model, giving a total of 243×10^6 parameters. Consequently, this model is approximately twice the size of the preceding text-only baseline. The hyper-parameters for the audio and text encoders remain unchanged.

We also note that, given the kernel sizes and strides of the convolutional part of wav2vec 2.0, we are able to compute the receptive field of each frame in the audio encoder, which is approximately 25 ms per frame. The window size of the audio

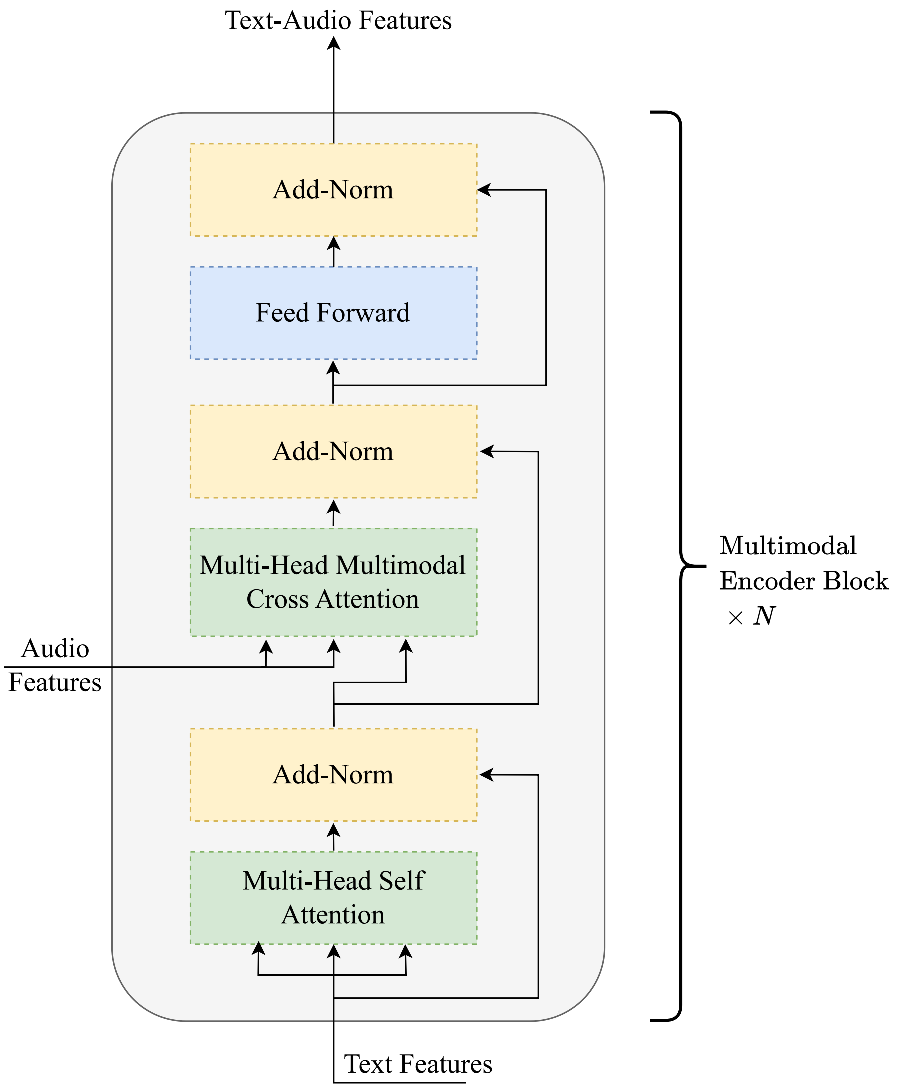

Text-Audio Features
Add-Norm
Feed Forward
Add-Norm
Multi-Head Multimodal Cross Attention
Audio Features
Add-Norm
Multi-Head Self Attention
Text Features
Multimodal Encoder Block × N

Figure 5.9: Architecture of the multimodal cross-attention block

encoder is determined by the original task for which it was designed, namely speech recognition. Such a small window size is common in speech recognition tasks, where the goal is for one frame to capture only one specific sound without any overlap with neighboring sounds. Later in this section, we will estimate how well this window size suits the punctuation and case restoration task.

However, due to the model's size of 243×10^6 parameters and the quadratic memory complexity of wav2vec 2.0 with respect to the processed audio input, which is usually longer than the textual input, we had to dramatically reduce the batch size from 200 to 4 per GPU, given the 16GB of GPU RAM. As a consequence of the smaller batch size, the learning rate was decreased from 6×10^{-5} to 1×10^{-5}. With these settings, training for just 4 epochs was roughly 7 times longer than the training time of the text-only baseline.

Observe that during training, the text-only baseline model undergoes one update for every 200×2 examples. In contrast, the multimodal model updates once for every 4×2 examples. This translates to the multimodal model undergoing 50 updates for every single update of the text-only baseline. For example, after 20,000 updates, the multimodal model has seen only 160,000 examples, whereas the text-only baseline has encountered 8,000,000 examples for the same number of updates. However, if we invert our perspective, the multimodal model executes 12,500 updates for every 100,000 examples, while the text-only model performs a mere 250 updates. Summing up, during their respective training periods, the text-only model processes approximately 16M examples, whereas the multimodal model processes around 4.4M examples. In an ideal scenario, comparing these two models would require training the text-only baseline with identical batch sizes and learning rates. However, limiting the text-only baseline artificially might unfairly lead to worse results. A better comparison would be between the best text-only model and the multimodal counterpart. However, different batch sizes make this comparison tricky, forcing us to compare the absolute performance values of models using various metrics.

5.5.1 Punctuation Restoration

In Figure 5.10, we observe the performance of the audio model in terms of punctuation detection. Interestingly, the difference in performance between short and long sequences is not very pronounced and diminishes as training progresses. Notably, precision plateaus after the initial 200,000 steps, staying in the fixed interval, while recall steadily increases. It is intriguing that upgoing trend in recall does not compromise precision, suggesting the model progressively identifies new punctuation places while maintaining detection quality.

When we compare the performance of the text-only model PDP-PDR (depicted on Figure 5.4) with our audio baseline (Figure 5.10) — which, although trained over a longer duration, encountered less data — the differences are evident. Primarily, the performance distinction between short and long sequences is nearly eliminated, implying the audio model's improved capability in punctuating even short sequences. There are multiple explanations to rationalize this behavior. The optimistic perspective corresponds to the model's utilization of audio features to improve punctuation detection. In contrast, a more skeptical view infers that the larger size of the multi-modal model enables better punctuation detection independent of audio. To dive deeper into this, we will perform experiments incorporating random audio input. Another visible trend is the improvement in recall over the text-only baseline. In quantifiable terms, the text-only model registers the best PDR and PDP combination of 0.91 and 0.945 respectively at the 20,000th step. Conversely, around the 450,000th step for the audio-fused model, we record a PDR of 0.92 and a PDP of 0.945, showing a recall improvement with same precision.

We extend our discussion to the punctuation performance of the multimodal model, focusing on F1 scores as depicted in Figure 5.11. Similar to what we saw with PDP and PDR, we notice steady improvements in the metrics throughout the training process. The model gradually learns to discern among different punctuation classes, starting the most represented classes(Table 3.5), namely commas and periods. The subsequent learning phases involve quotation marks, colons, dashes, and question marks, culminating in the learning semicolons. Regrettably, the model does not show proficiency with exclamation marks. A common trend observed across all punctuation types is the gradual improvement of the non-blank average F1 score over the entire training duration.

When comparing the performance of the multimodal model (Figure 5.10) with that of the text-only model, as depicted in Figure 5.5, we see distinct differences. Foremost,

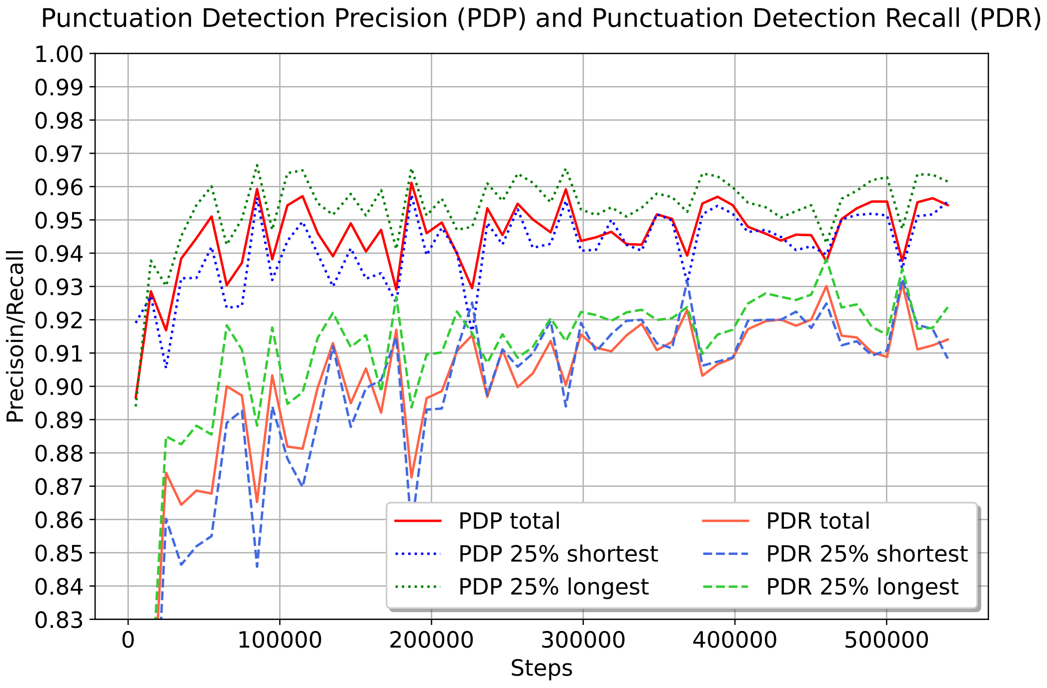


Figure 5.10: Performance of the audio model with wav2vec 2.0 encoder in punctuation detection during training.

the multimodal model demonstrates a superior overall average F1 score in comparison to the text-only baseline. Specifically, a score of approximately 0.482 is achieved at the 500,000th step, contrasted with 0.473 at the 30,000th step for the baseline model. Evaluating each class separately reveals the multimodal model's advantage over the text-only baseline, especially in the period class; it attains an F1 score of 0.806 compared to 0.735 for the baseline. Both commas and semicolons register higher scores for the multimodal model, whereas other classes maintain scores relatively similar to those of the text-only counterpart. Intriguingly, neither the question marks nor the exclamation marks show any significant improvements. This is counterintuitive, given the intuition that audio would be particularly beneficial for these classes. In conclusion, mirroring the trends observed with PDP and PDR, the multimodal model surpasses the text-only baseline in the punctuation restoration task.

5.5.2 Capitalization

Now, we turn our attention to the model's performance on capitalization restoration, starting from F1 scores, depicted in Figure 5.12. We observe that throughout the training process, the model manages to improve its performance on the capitalization task without getting stuck at local optima. Further, it is likely that if we continued to train this multimodal model for a few additional epochs, the score may increase even further.

A comparison of the F1 scores for the capitalization class of the multimodal model and the text-only baseline reveals that the multimodal model also outperforms the baseline in this area. In absolute terms, the highest F1 score for the capitalization class for the text-only model is about 0.837, as can be seen on Figure 5.7, whereas the

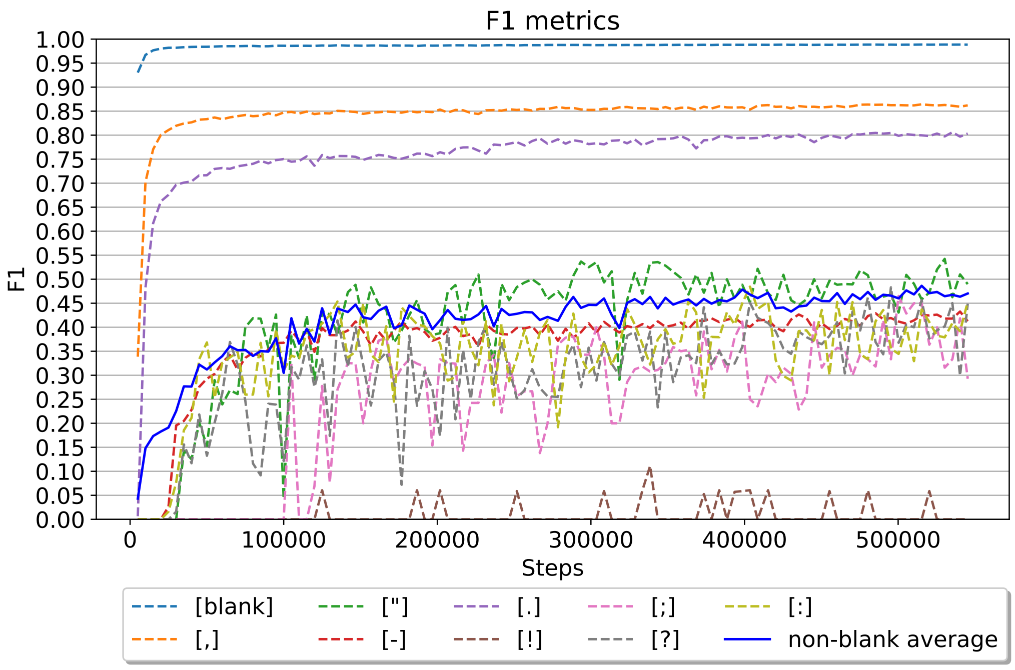


Figure 5.11: F1 scores of the multimodal model with wav2vec 2.0 encoder for all punctuation classes over the course of training.

multimodal model shows a solid performance of 0.863. A possible explanation could be that, as previously observed, the multimodal model improved its F1 score on periods compared to the textual baseline. Given the natural correlation between capitalization and periods, this could have led to an improvement in the capitalization task as well. However, such a large difference between the audio model and the text-only baseline is somewhat surprising with this explanation alone. As previously suggested, it would be interesting to investigate whether such an improvement could be achieved solely by increasing the model size without utilizing sound.

5.5.3 Multimodal Attention Analysis

To understand the utilization of audio in this multimodal setup, we can examine the attention weights of the cross-attention layer. Figure 5.13 plots the average attention weights of the cross-attention layer within the final multimodal block. Having extracted timestamps for every word in the audio, it becomes feasible to find the approximate position of a word within the audio's latent space. The illustrated examples were obtained by segmenting the validation set into quartiles based on text length, subsequently selecting the instance with the highest loss in each quartile. This stratification was motivated by the model's different performance across short and long sequences.

The text dimension corresponds to the y-axis and sound to the x-axis, forming a rectangular matrix where the element at position (i, j) indicates the degree of attention text token j pays to sound frame i. The texts and sounds are presented in their complete forms, including all punctuation and casing, allowing for an assessment of whether a word that should be assigned a particular punctuation or casing class is actually focusing on the correct word in the sound space. It is important to note that in these plots, timestamps for words in the latent sound space indicate the start but

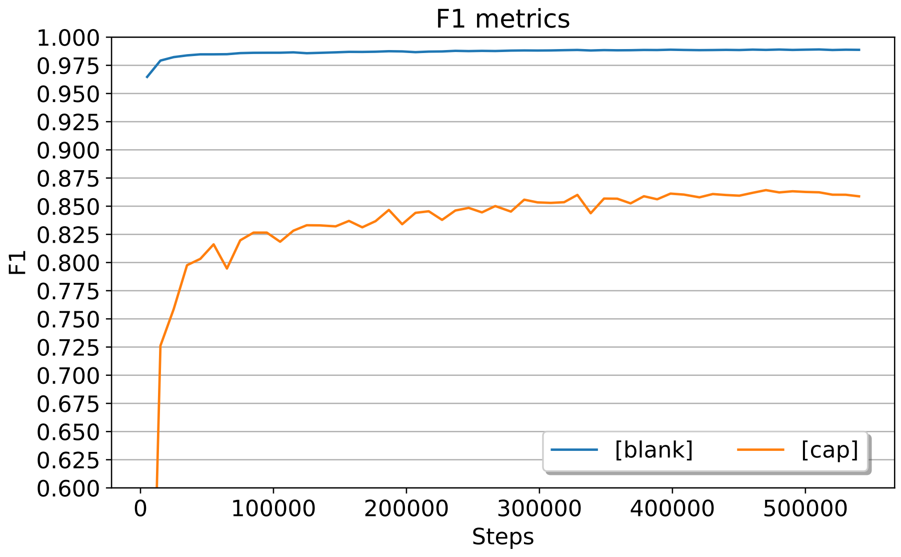


Figure 5.12: The performance of the multimodal model with wav2vec 2.0 encoder, evaluated in terms of F1 scores for case restoration task.

not the end of a word. This can imply a silence or the end of a word if the subsequent frame lacks an assigned word. In the text representation, some rows do not contain any word due to the splitting of words into multiple tokens, with word labels indicating the end of a word, unlike the sound representation that marks word starts. Due to the large receptive field for each frame in the audio hidden space, a single frame may contain multiple words, contrasting with speech recognition approaches where each grapheme is represented in multiple frames to prevent overlap.

Primarily, an evident trend across the samples is the striking similarity observed in most rows, telling that each text token consistently examines audio features in an almost identical manner. This could be a marker indicating that audio features might not be particularly representative, potentially lacking any information needed for determining punctuation or casing. Another observation is low magnitude of the attention weights, further showing the possibility that audio features may not provide any significant relative information. On closer inspection, while certain parts of the plot appear dark or light, the actual difference between these regions is small since the attention values belong to a small interval.

In summary, the attention plot demonstrates that, for this multimodal technique, audio features might not provide any meaningful information. Thus, it seems that the primary gains of the model are likely attributable to the added parameters.

In conclusion, the multimodal model exhibited notable enhancements over the text-only baseline in terms of F1 scores for both punctuation and capitalization tasks. Specifically, the model demonstrated significant improvements in identifying periods and correctly capitalizing words. However, it also highlighted certain limitations presented in the current approach. The audio encoder, originally designed as an ASR model, operates on small audio segments. This characteristic may not be ideal for punctuation or capitalization restoration tasks, where longer frames may be necessary to capture temporal changes such as pauses and variations in intonation. Additionally, the audio encoder, based on the BERT architecture, has quadratic memory complexity with respect to the length of the audio input, forcing to use a very small batch size.

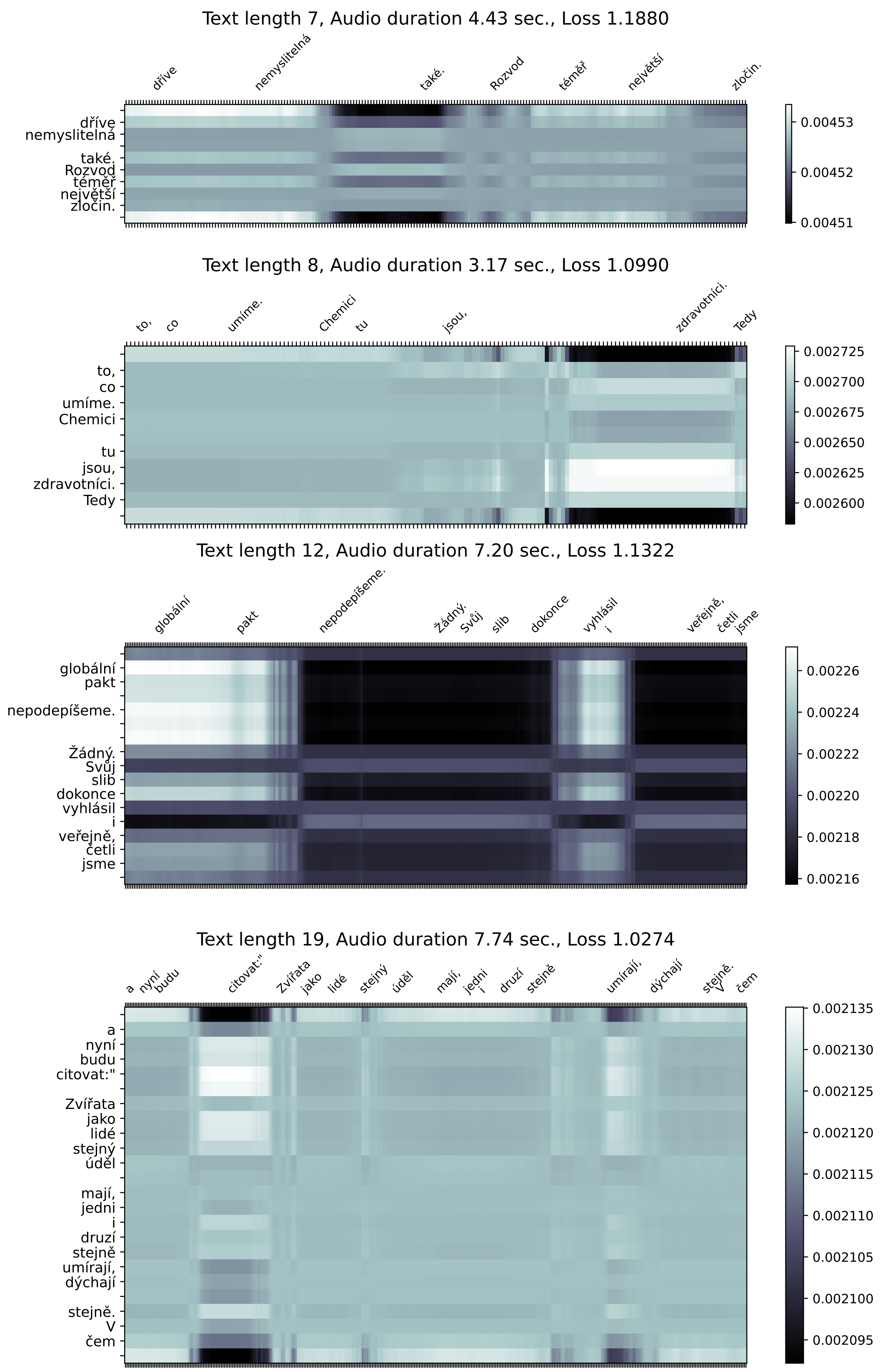


Figure 5.13: Attention weights averaged across heads of the last cross-attention layer in multimodal model with wav2vec 2.0 encoder. Note that, although the differences in the attention scores are generally very small, the colour visualization highlights them.

Overall, these limitations may hinder the multimodal model's ability to effectively utilize the audio information.

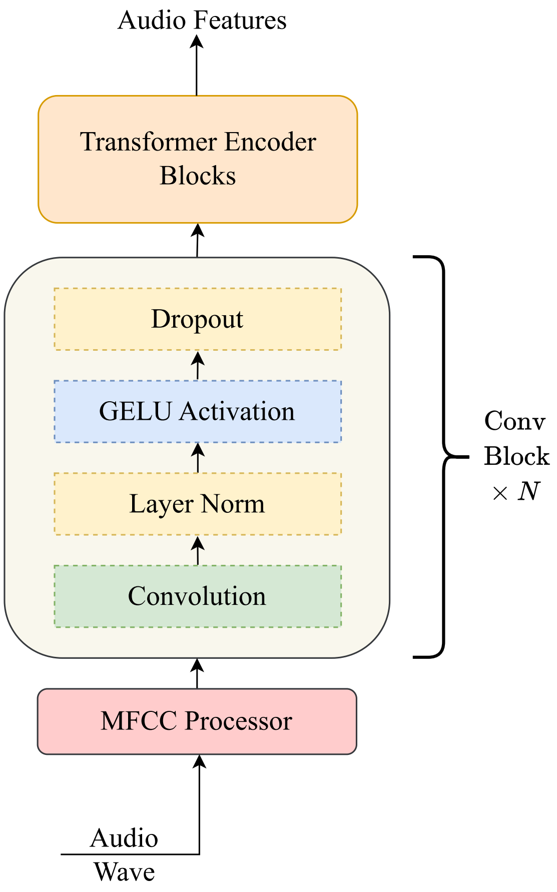


Figure 5.14: Architecture of the MFCC-based convolutional sound encoder.

5.6 MFCC Convolutional Sound Encoder

In the preceding section, we observed the potential enhancement in model performance by integrating audio. Yet, due to the architecture of the audio encoder, we confronted challenges, particularly in drastically reducing the batch size from 200 to 4 in comparison with the text-only baseline. Also the utility of the sound input, given the architecture of the audio encoder, was arguable. Our focus now shifts to supplanting the prior wav2vec 2.0 audio encoder with a lighter version grounded in mel spectrograms complemented by convolutional processing layers followed by two Transformer encoder blocks. The structure of the new MFCC convolutional sound encoder is illustrated in Figure 5.14.

The mel spectrogram input preprocessor diminishes the frequency to 61 Hz, appending a new feature dimension consisting of 128 Mel filter-banks. We employ torchaudio's [Yang et al., 2021] existing implementation to derive the mel spectrogram from raw audio, facilitating the extraction of Mel features on GPUs. The preprocessor is set with a window length of 512 and a hop length of 256. Subsequently, to augment the receptive field of each latent audio frame and learn novel features, we incorporate a compact convolutional network. This design draws inspiration from wav2vec 2.0 audio processing, encompassing several identical basic convolutional blocks. Every block comprises a convolution, Layer Normalization layer, GELU activation function, and a dropout. With a dropout rate of 0.1 and ϵ set at 1×10^{-5} for the Layer Normalization, there are seven blocks in total. The strides for convolutions are as follows: [2, 2, 2, 2, 1, 2, 1], with all kernels sized at 2, and the outgoing features from each convolution are [128, 256, 256, 512, 512, 768, 768]. Neither mel spectrogram processing nor convolution blocks employ padding, which curtails the audio sequence length while amplifying the hidden dimension. Audio preprocessing through mel spectrogram and convolutional blocks amplifies the receptive field of each latent audio frame to roughly 1.3 seconds, a marked improvement from the 0.025 seconds via the wav2vec 2.0 audio encoder.

As earlier mentioned, following convolutional block processing, the refined audio traverses two Transformer encoder blocks. While convolutional processing should distill pertinent details from audio waves, the attention mechanism within Transformer encoder blocks primarily addresses interactions amongst latent audio frames. Each block is uniformly structured, comprising 8 attention heads, a linear projection layer that upscales latent audio frames from 768 to 2048, a dropout rate of 0.1, and layer normalization with $\epsilon = 1 \times 10^{-5}$. The final Transformer encoder block's output is channeled into the multimodal attention block, akin to the prior wav2vec 2.0 framework.

The driving principle of this revised audio encoder is improving the wav2vec 2.0 encoder to better suit punctuation and casing restoration tasks. Where wav2vec 2.0 integrated a full-sized Transformer encoder model with 12 blocks and a small convolutional preprocessor to represent input audio, the new encoder applies more convolutional layers to further compress audio, simultaneously reducing the Transformer encoder block count. Given that Transformer encoder blocks present quadratic memory demands with respect to the sequence length, compressing audio allows to apply larger batch size. This sequence length reduction reinstated our batch size to 200 per GPU. Furthermore, we reverted to the original learning rate of 6×10^{-5} and the original number of epochs of 15.

Contrasting the two, the novel audio encoder has only 14×10^6 parameters comparing to the 95×10^6 in wav2vec 2.0. Also, in total the new multimodal model integrates just 34×10^6 (from the audio encoder and multimodal encoder) additional parameters compared to the text-only baseline, approximating 26.5% of the text-only model's parameters.

The loss of the multimodal model equipped with the MFCC Convolutional Sound Encoder is illustrated in Figure 5.15. Initially, both the validation and training losses diminish throughout the training, suggesting that further training for several more epochs could potentially enhance the model's performance, particularly in terms of the loss metric. Contrasting punctuation restoration with the casing task in terms of cross-entropy loss, the model exhibits superior performance in capitalization relative to punctuation restoration. This discrepancy is manifested across both training and validation sets. Intriguingly, as training progresses, the disparity between training losses narrows. This could imply that beyond a certain point, there is minimal room for improvement in the casing task. Furthermore, a discernible gap exists between the training and validation losses for both tasks. As previously noted, such a gap could be indicative of model overfitting, yet it should not be solely relied upon for decision-making. When juxtaposed with the loss trajectory of the text-only baseline, as depicted in Figure 5.3, the novel multimodal methodology registers analogous loss values for both tasks.

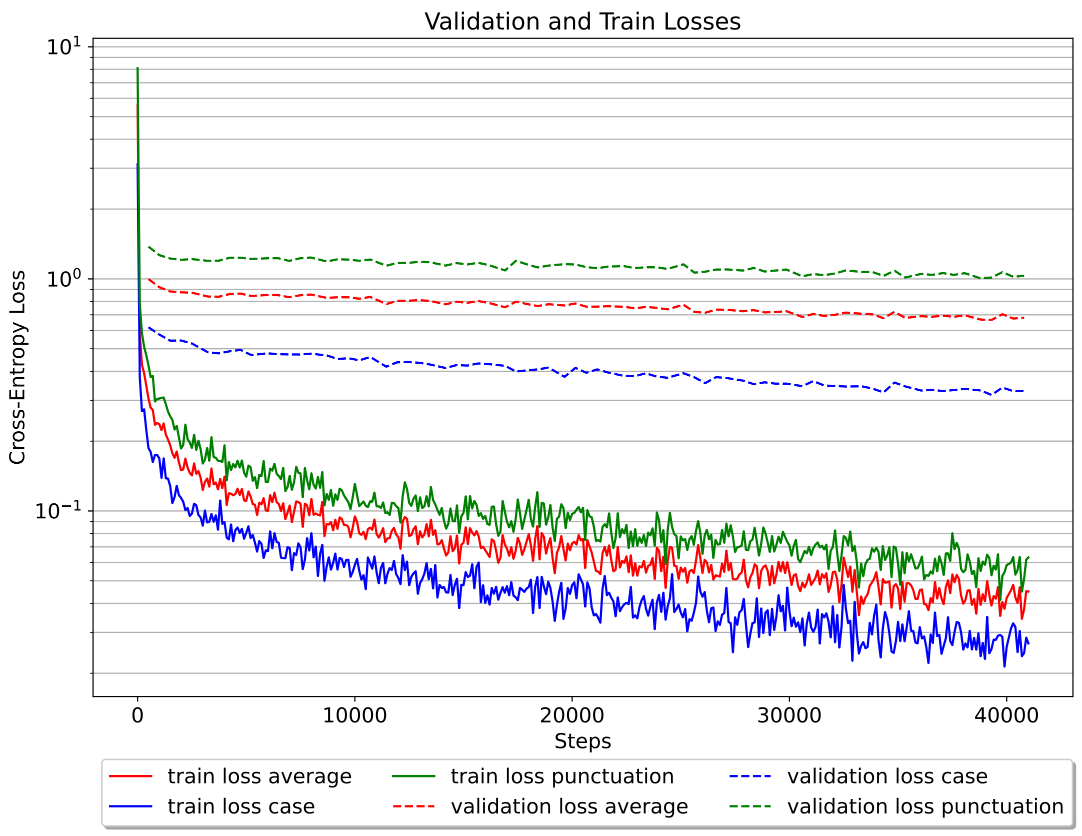


Figure 5.15: Loss dynamics of the multimodal model with MFCC-based convolutional sound encoder.

5.6.1 Punctuation Restoration

We now turn our attention to the model's efficacy on the punctuation detection task, as illustrated in Figure 5.16. As observed previously, the model's performance varies based on sequence length. Nevertheless, during training, the disparity between long and short sequences diminishes for both precision and recall metrics. Another discernible trend is the stabilization of precision and recall after approximately 20,000 training steps, both metrics fluctuating within a narrow band centered around 0.95 and 0.93 respectively. Potential marginal advancements are noticeable after the 35,000th step. The model's peak in terms of PDP and PDR is attained around the 38,000th step, recording values of 0.945 and 0.93 respectively. Contrasting these metrics with the peak values from the wav2vec 2.0 based variant, plotted in Figure 5.10—specifically 0.945 for PDP and 0.92 for PDR—it becomes evident that the novel audio encoder empowers the model to pinpoint a greater number of punctuation locations, while maintaining the consistent precision. Broadly speaking, when solely appraising the PDP and PDR metrics, it is clear that despite a reduction in the parameter count of the audio encoder, the new strategy outperforms its predecessor.

In this study, we extend our analysis of the model's performance in the punctuation prediction task, focusing on F1 scores as illustrated in Figure 5.17. The model demonstrates commendable performance in the initial 28,000 steps, achieving a peak non-blank average F1 score of 0.496 at the 27,400th step, followed by a marginal decline towards the end of the training period. A detailed examination of performance across individual punctuation classes reveals that the model's best performance, particularly

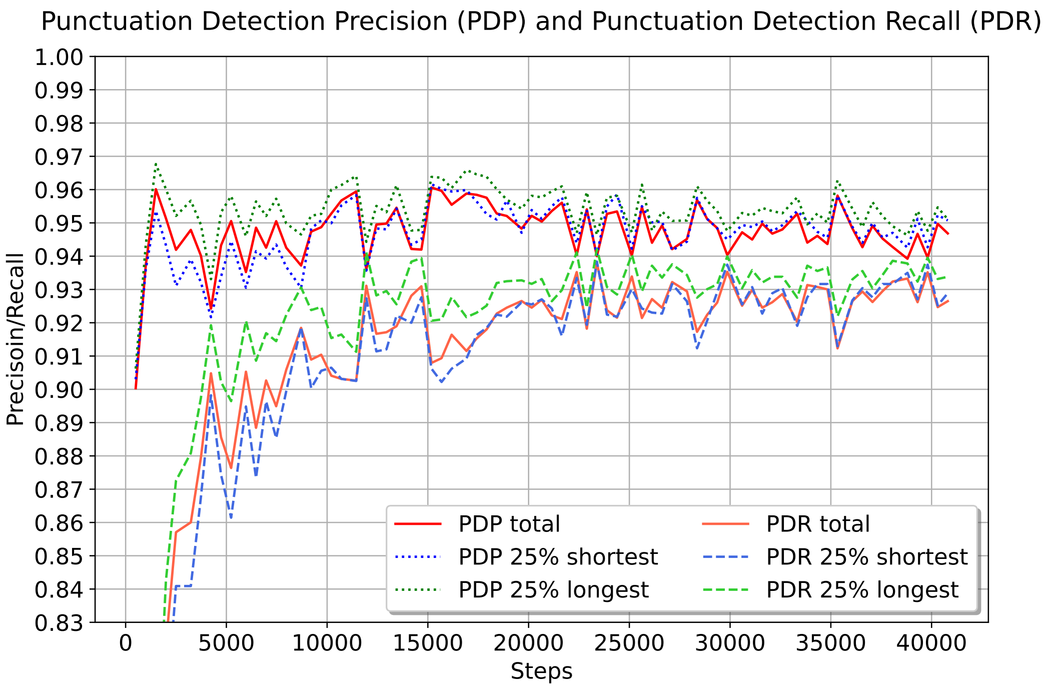


Figure 5.16: Precision and recall performance of the MFCC-based multimodal model on the punctuation detection task.

in recognizing exclamation marks and achieving peak performance for semicolons and quotation marks, occurs between the 18,000th and 28,000th steps. After the 28,000th step, a downward trend is observed for semicolons and quotation marks, while the performance on exclamation marks remains stable throughout the training. Consequently, it appears improbable that continued training in the same manner would yield significant improvements.

Comparing the performance of the multimodal model, which uses an MFCC-based audio encoder, with our prior model that utilized a wav2vec 2.0 sound encoder, notable enhancements are evident. The previous multimodal approach's F1 scores for punctuation marks are depicted in Figure 5.11 for reference. The new model attains a non-blank average F1 score of 0.496, surpassing the previous model's peak score of 0.482. Analysis of the two plots indicates that the primary improvements are in the recognition of exclamation marks and quotation marks, while the F1 scores for other punctuation classes are comparable to those achieved with the wav2vec 2.0 sound encoder. In summary, the new model with a more compact sound encoder outperforms the previous approach.

Further discussion on the model's performance in punctuation restoration is facilitated by Figure 5.18, which presents confusion matrices for each punctuation class, computed on the entire validation set at selected training steps. As observed in Figure 5.17, the model progressively learns to detect each punctuation mark type, influenced by its frequency in the training data, starting with the most common (commas and periods) and concluding with the least represented (exclamation mark). Notably, despite the higher frequency of commas in the data, the model more frequently misclassifies other classes as periods rather than commas. The possible explanation is that, other classes like exclamation marks, question marks and semicolons are usually

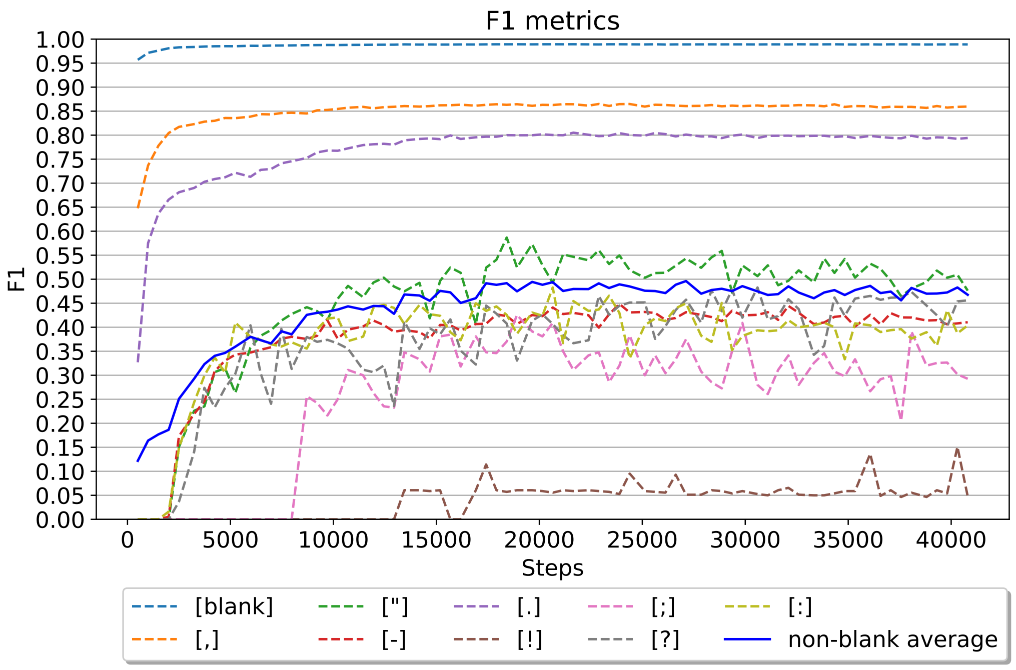


Figure 5.17: F1 score performance of the MFCC-based multimodal model on punctuation prediction.

correspond to the end of the sentence, unlike commas that usually follow strict grammar rules in Czech. Contrary to the observations in Figure 5.17, where no significant improvement was noted after the 25,000th step, Figure 5.18 indicates continued optimization in performance for all classes towards the end of training, with the model achieving its best performance at the 36,100th step.

5.6.2 Capitalization

Turning to the model's performance in the capitalization restoration task, Figure 5.19 displays the F1 scores for each capitalization class. The primary focus is on the class representing capitalization. The model commences with a relatively high F1 score of 0.625, which then incrementally increases to 0.868 around the 24,500th step, maintaining this level for most of the training duration with a slight decline towards the end. Intriguingly, the F1 score for the capitalization class correlates with the model's performance on the period class (as shown in Figure 5.17), where the model also achieves its best results around the 23,000th step. Comparing these results with the previous multimodal approach (Figure 5.12), the highest F1 score attained by the previous model was 0.863 at approximately the 470,000th step, which is lower by 0.005 points compared to the current solution.

Our analysis of the model's performance in the case restoration task ends with an examination of confusion matrices at various training stages, as depicted in Figure 5.20. Initially, the model achieves a 77% accuracy in detecting missing casing after the first 2,000 steps. This accuracy stabilizes at around 85% after 10,000 steps, with a minor improvement observed at the 22,000th step. These results exhibit slight discrepancies with the F1 score for the capitalization class, where a progressive improvement was noted until the 23,000th step. The confusion matrices, however, show only marginal

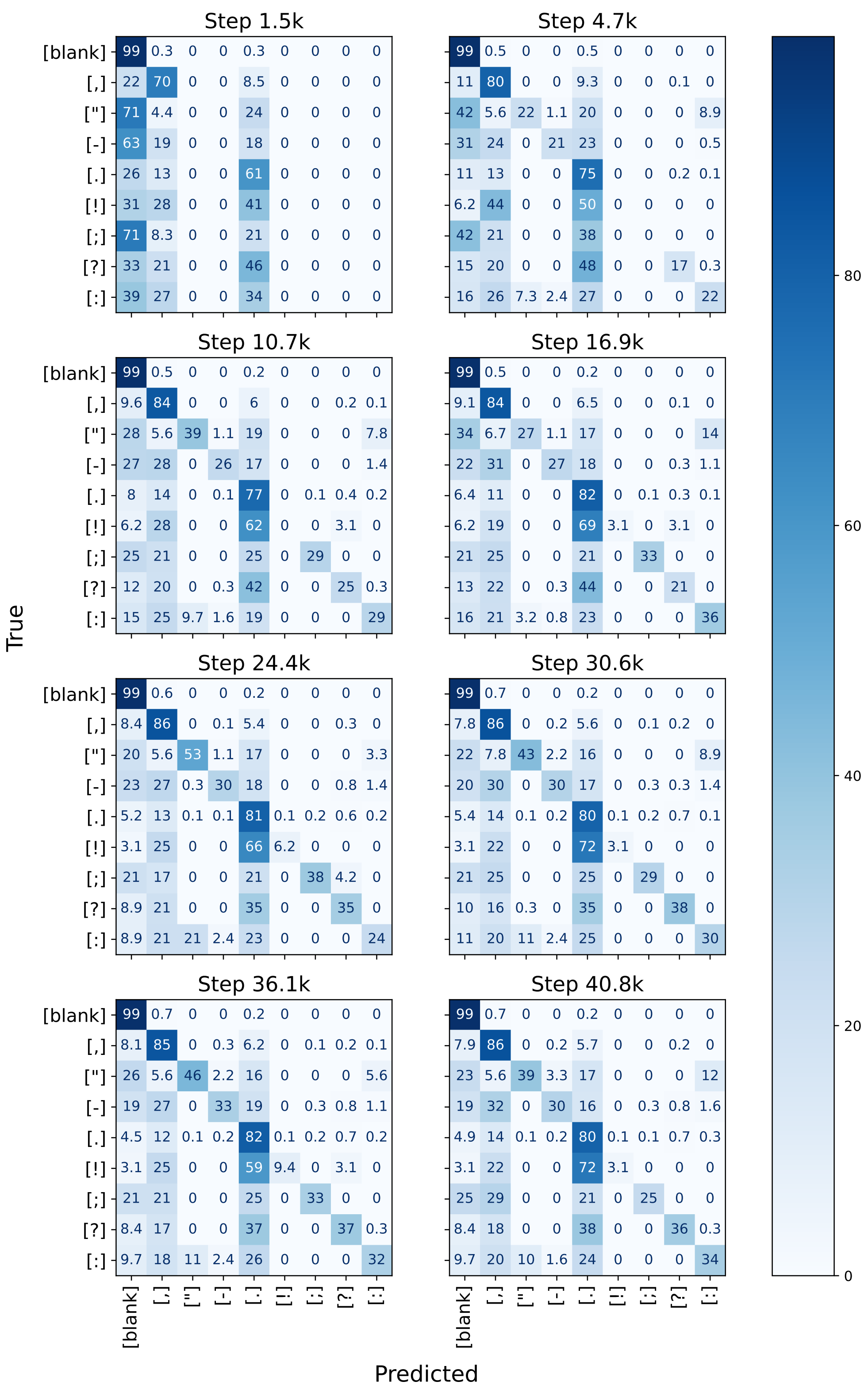


Figure 5.18: Visualization of confusion matrices for each punctuation class at selected training steps for the MFCC-based approach.

improvement after the 10,000th step. This discrepancy may be attributed to the limited number of steps visualized in the confusion matrices plot, suggesting that the results might not be as straightforward as they appear.

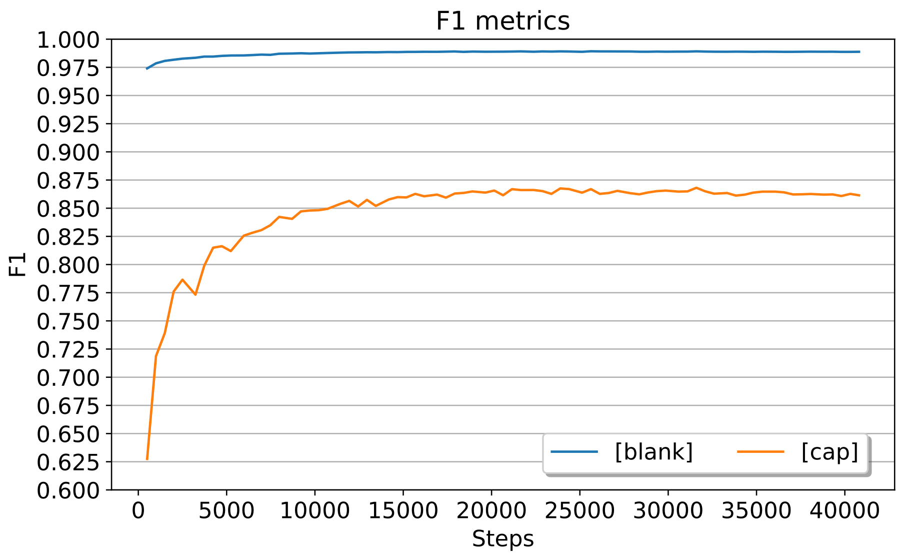


Figure 5.19: F1 scores of the MFCC-based multimodal model for the capitalization and blank classes in the capitalization restoration task.

5.6.3 Multimodal Attention Analysis

In our final analysis of the new multimodal model featuring an MFCC-based audio encoder, we turn to the attention plots in Figure 5.21. As we can see, some sound frames exhibit relatively high attention scores, suggesting their potential utility in addressing punctuation and case restoration challenges. While certain text tokens focus on specific audio frames, others disregard the same frames, indicating a refined ability of text tokens to discern the relevance of a given frame. This results in a shift from vertical patterns in the attention matrices to more diagonal or lower triangular patterns. Attention clustering at the beginning of a segment is also noted, possibly due to a lack of positional information in the sound features, caused by the large number of convolutional blocks, making the audio latent representation to act as a bag of audio tokens. Alternatively, this clustering could indicate a less versatile audio latent space, incapable of effectively conveying sound events, leading text to focus on segments with the most representative audio frames.

Comparing this with the attention plot for the previous wav2vec 2.0 based approach (Figure 5.13), we observe significant improvements. In the new model, attention scores reach up to 0.2 for selected frames, whereas in the previous model, attention scores were tightly clustered with a maximum value of 0.0045. This suggests that in the new approach, text can identify useful information in sound while disregarding irrelevant segments. Additionally, unlike the previous model, we do not observe pronounced vertical patterns in the attention plots, indicating that each text token can independently navigate the audio latent space and select token-specific information, although this information is apparently not corresponding between sound and text quite monotonically. Interestingly, both models exhibit higher attention scores for shorter sequences, suggesting that with limited textual context, models utilize more sound information. However, as textual context increases, the reliance on sound diminishes, as inferred from lower attention scores for longer sequences.

In summary, the new multimodal approach has led to multiple improvements. Most notably, the text representation now effectively utilizes the audio latent representation,

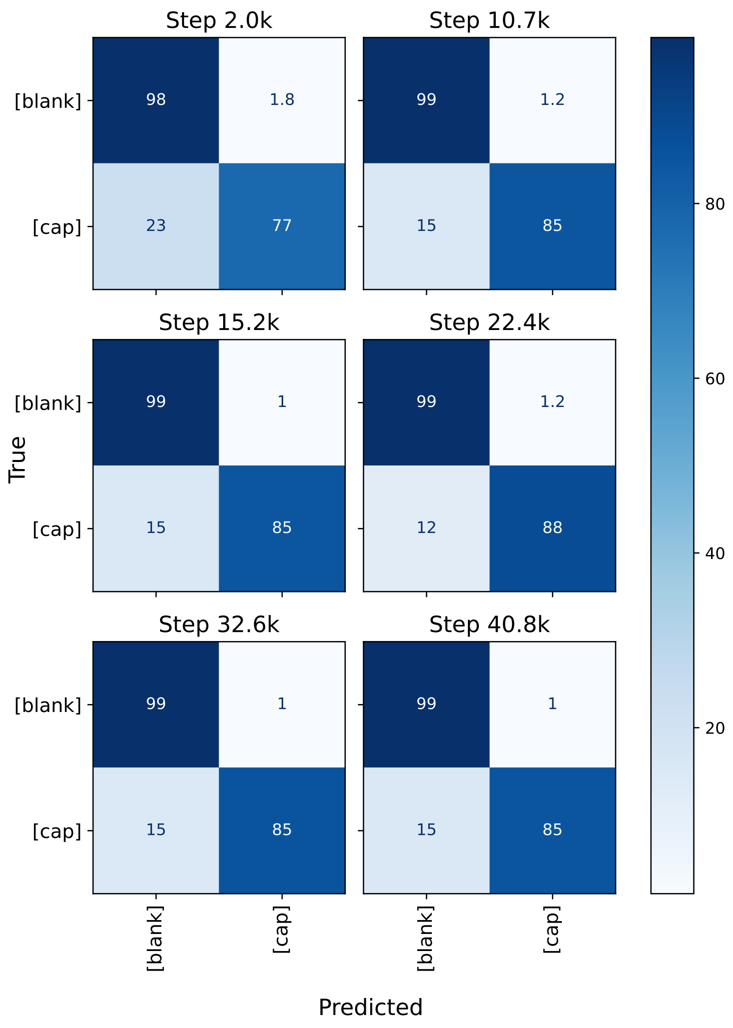


Figure 5.20: Confusion matrices of the MFCC-based model for case restoration at various training stages.

as demonstrated in the attention plots (Figure 5.21). Additionally, the MFCC-based approach outperforms the previous model in various metrics, including F1 scores for both punctuation and capitalization restoration classes, and shows increased recall in punctuation detection while maintaining precision comparable to the wav2vec 2.0 scenario. Another technical advancement is the model's ability to operate on larger batches, with fewer parameters and reduced training time compared to the first multimodal approach. The larger batch size also aligns well with our text-only baseline, which now shares the same batch size as the MFCC-based multimodal approach.

However, potential areas for improvement remain. The attention plots (Figure 5.21) reveal attention clusters at the beginning of some sequences, suggesting that the audio features may be somewhat unordered, making it challenging for text features to follow words in the audio space. This issue is more pronounced in longer sequences, where high attention scores are not consistently observed near the diagonal. Our next approach will aim to address these challenges.

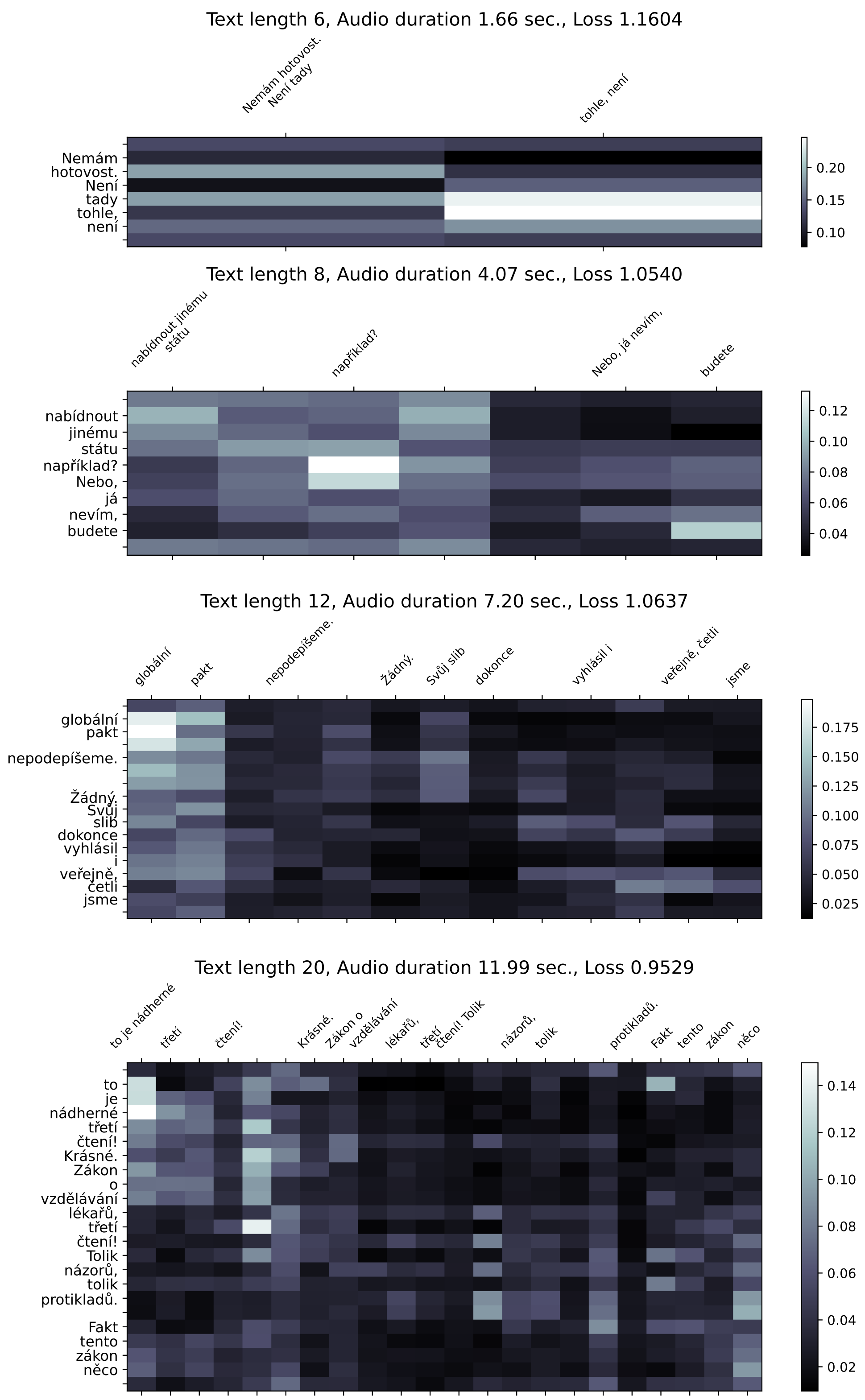


Figure 5.21: Attention plots for the MFCC-based multimodal model. The plot structure is described in Section 5.5.3.

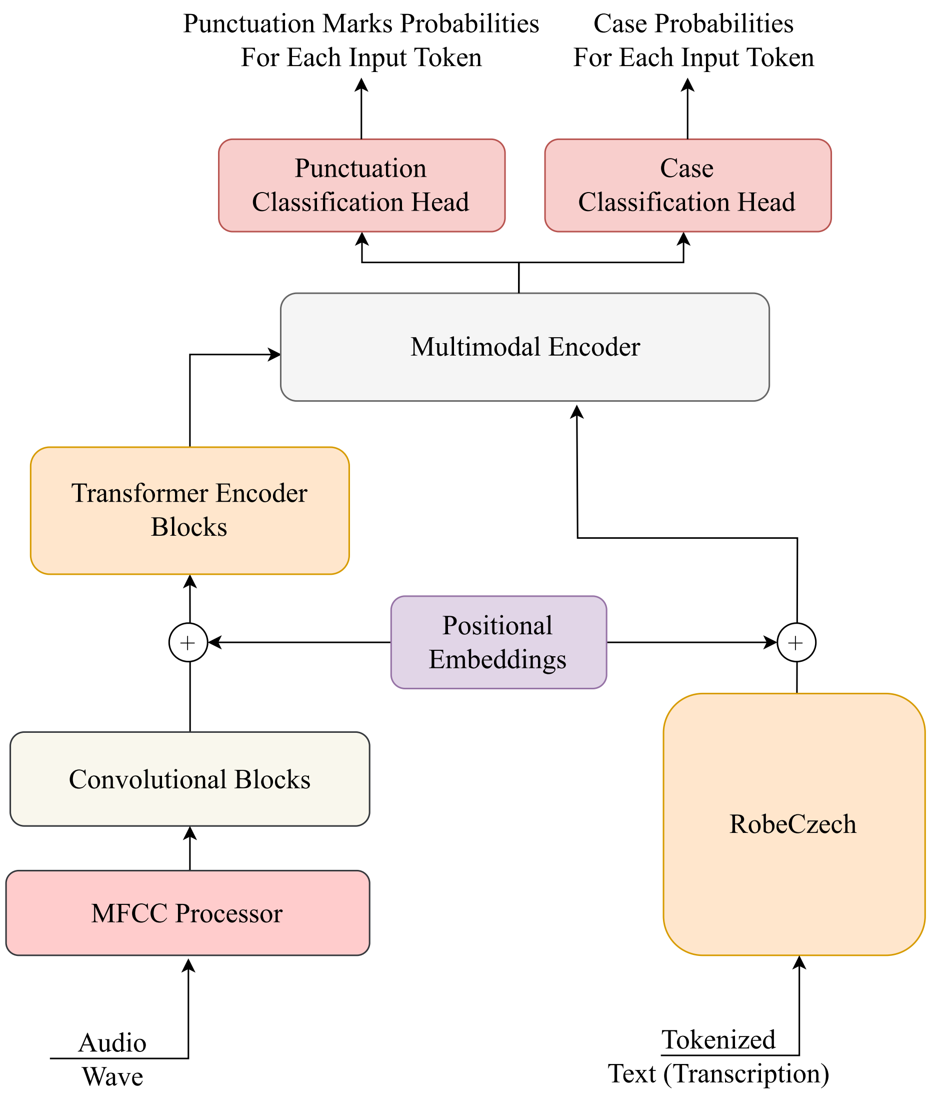

Punctuation Marks Probabilities For Each Input Token
Case Probabilities For Each Input Token
Punctuation Classification Head
Case Classification Head
Multimodal Encoder
Transformer Encoder Blocks
Positional Embeddings
Convolutional Blocks
RobeCzech
MFCC Processor
Audio Wave
Tokenized Text (Transcription)

Figure 5.22: Schematic of the new model incorporating positional embeddings.

5.7 Adding Positional Embeddings

In this section, we aim to augment the previous multimodal model, which utilized an MFCC-based audio encoder, by integrating positional information into the audio encoder. This initiative is motivated by the challenge in weak correlation of text tokens with their corresponding audio representations in the previous model, a difficulty that exacerbated with an increasing number of words in the input. Our objective is to establish a more robust connection between text and sound representations of words. This will be achieved by introducing a positional embedding layer, shared between the sound and text encoders. The schematic of the new model with positional embeddings is illustrated in Figure 5.22.

The dataset's construction inherently provides starting and ending times for all recognized words, determined by the wav2vec 2.0 ASR model. Utilizing these timestamps, we extracted segments with a zero word error rate, ensuring alignment with the words in the original transcript. Consequently, we have segments with known starting and ending times. Given the parameters of the MFCC transformation and each convolution layer, we can approximate timestamps for each word in the latent space. Since convolutional blocks aim to reduce audio sequence length and extract positionally independent information about speech segments, the insertion of learnable positional information is most effective before the Transformer encoder blocks. The self-attention mechanism in the Transformer encoder blocks, which processes the entire sequence, benefits from positional information, unlike the convolutional blocks that focus on local features. In the text encoder, same positional embeddings are added to the outputs of the final layer, thereby facilitating a more coherent connection between sound and text features for the multimodal encoder.

Learnable positional embeddings assign each frame a textual ordinal position, implicitly creating a linkage between text and sound. For longer words, multiple frames will share the same positional embedding, while very short words may correspond to one or even zero frames, particularly in cases where multiple words overlap in a single frame in the latent space. Consider a word w_i appearing in frames $f_l, \dots, f_{l+m}$; we add the positional embedding $e(i)$ to each frame and then pass this sum to the Transformer encoder blocks of the sound encoder. On the text side, the number of positional embeddings for a given word depends on its token count. For a word w_i represented by tokens $t_j, \dots, t_{j+k}$, we add the positional embedding $e(i)$ to each token. Ideally, this embedding strategy should reinforce a diagonal pattern in the multimodal cross-attention layer, particularly when no words are omitted in the hidden audio representation.

Our model analysis commences with an examination of the loss curves, as depicted in Figure 5.23. As observed in previous iterations, a significant gap exists between each task and between the training and validation phases. While this could typically indicate overfitting, it is not a concern in our context, based on prior experiments. More importantly, there is a consistent decay pattern for each task in both training and validation, suggesting continuous performance improvement throughout the training. Furthermore, the absence of plateaus in the plots suggests that further training could yield further improvements.

Comparing the loss performance of the current model with the previous model without positional embedding, as shown in Figure 5.15, reveals few differences. Both models exhibit large gaps between different tasks, with superior performance in case restoration and notable gaps between training and validation sets. Additionally, both models demonstrate continual improvement in their loss scores throughout the training. The only minor distinction is that the embedding-enhanced model shows slightly better performance in the capitalization task, though the improvement is not substantial. Based solely on the loss curves, it is challenging to ascertain whether the extension of the MFCC-based model offers significant advantages for the tasks.

5.7.1 Punctuation Restoration

We proceed with the analysis of the model's performance on the punctuation restoration task by examining the Punctuation Detection Precision (PDP) and Punctuation Detection Recall (PDR) curves, plotted in Figure 5.24. The model appears to converge to a local maximum within the first 25,000 steps, after which the performance plateaus, with PDP oscillating around 0.95 and PDR around 0.93. Notably, there is no discernible gap between overall performance and performance on the shortest sequences, while performance on the longest sequences is comparatively better. This indicates that while predictions on the longest sequences still benefit from a large context, sequences with smaller and medium context lengths perform similarly. The best

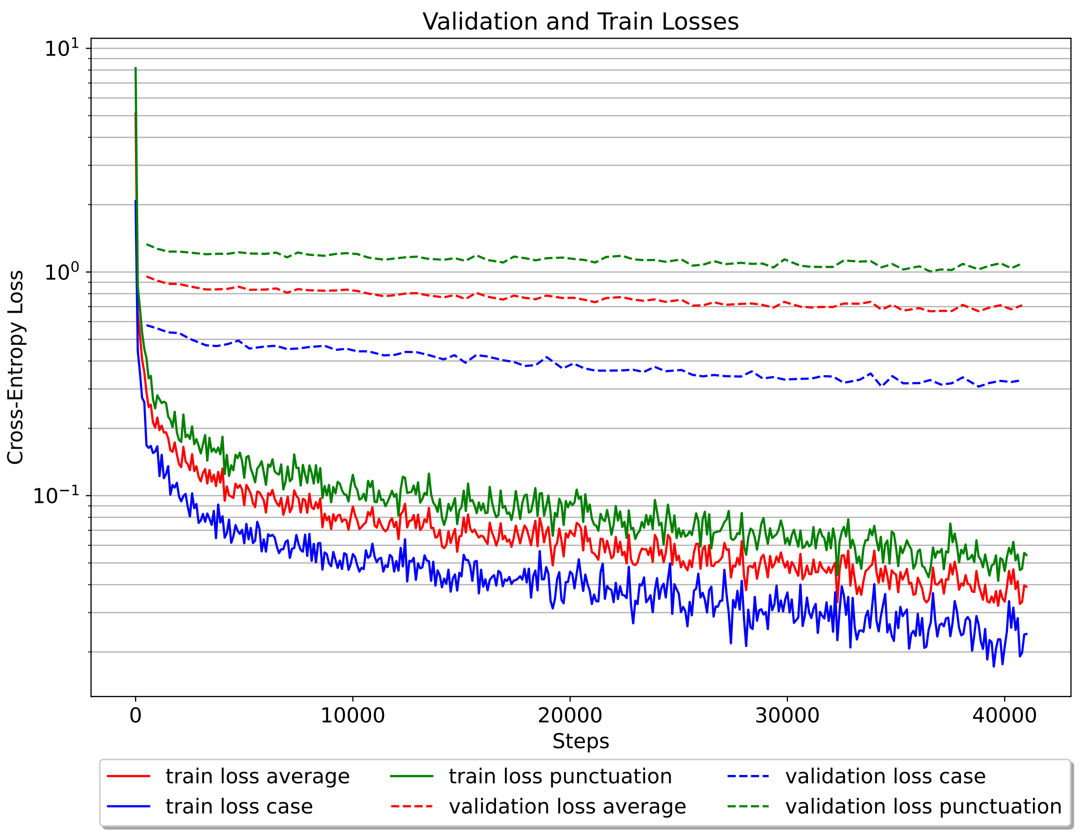


Figure 5.23: Loss curves for the multimodal model with positional embeddings.

pair of PDP and PDR values, specifically 0.945 and 0.935, was achieved at the 38,800th step, suggesting potential benefits from prolonged training.

To assess whether positional embeddings enhance the performance of the MFCC-based multimodal model on the punctuation detection task, we refer to the results of the previous model depicted in Figure 5.16. The overall behavior of the models appears quite similar, stabilizing in the second half of the training. However, it is noteworthy that the gap between long and short sequences is smaller for the model without embeddings, primarily because the new model exhibits improved results for longer sequences. Fixing the precision on long sequences at 0.95, we observe that the best PDR on long sequences of the current model is 0.943 at the 39,000th step, whereas the previous model only reaches 0.937 of PDR at the 40,000th step, having same precision. In general, the previous best total PDP-PDR value pair is 0.945 and 0.93 at the 38,000th step, which is marginally inferior to the current best pair of 0.945 and 0.935. Consequently, we infer that positional embeddings have enabled the current model to enhance its results on the punctuation detection task, likely due to improved performance on longer sequences.

To gain a deeper understanding of the model's behavior in punctuation restoration task, we now examine the model's performance in terms of F1 scores for each punctuation type. The corresponding curves are depicted in Figure 5.25. Initially, the non-blank average F1 score shows improvement over the first 27,000 steps, peaking at 0.502 at the 26,000th step, before oscillating around 0.47. Notably, after approximately the 28,000th step, we observe a decline in performance for periods and quotation marks. However, the model consistently improves its F1 score for the less represented types,

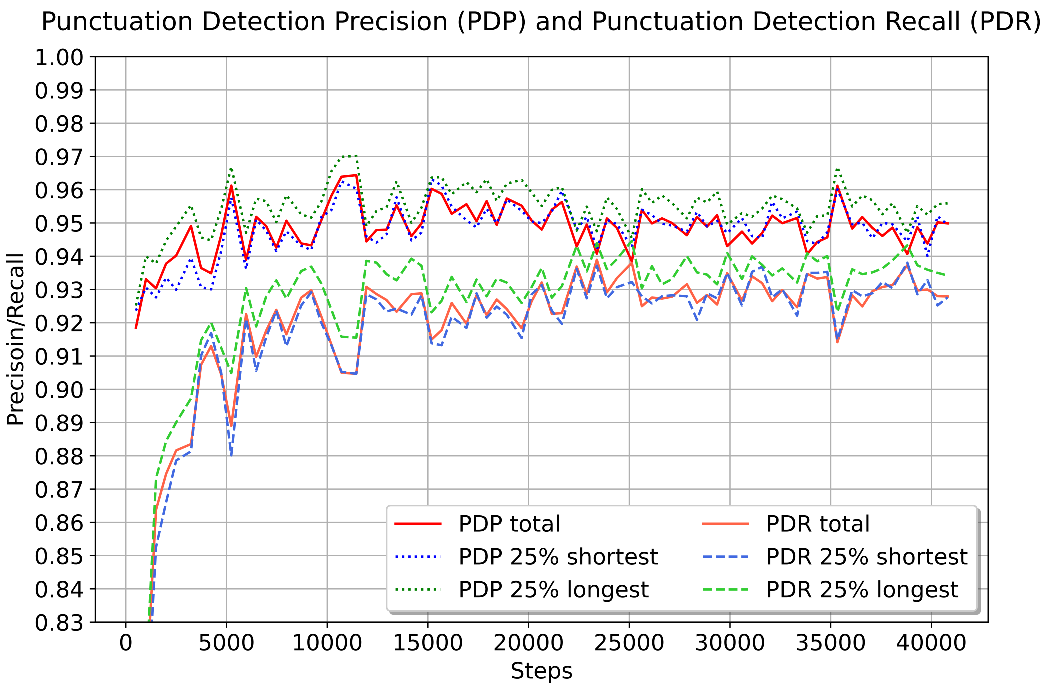


Figure 5.24: PDP and PDR curves for the multimodal model with positional embeddings.

namely exclamation marks and semicolons, suggesting potential benefits from further training.

In contrast to the PDP and PDR curves, we can now assess which punctuation types benefit most from the introduction of positional embeddings. The F1 curves of the preceding model, which lacked positional embeddings, are shown in Figure 5.17. A significant difference is observed in the classification of exclamation marks. The new model achieves a peak F1 score of 0.24 at the 40,000th step, compared to 0.15 in the previous model. This improvement, though seemingly counterintuitive, can be attributed to the natural similarity between exclamation marks and periods, differing primarily in emotional coloring and intonation. Given the rarity of exclamation marks in the dataset, even minor improvements can lead to substantial changes in their F1 score. The new model with positional embeddings also demonstrates enhanced performance in the period class, achieving an F1 score of 0.82 at the 24,500th step, compared to 0.803 at the 21,500th step in the previous model. While the improvement is not as marked as in the case of exclamation marks, it is notable given the high representation of the period class in the dataset. The performance of the new model in other classes, such as question marks, commas, and semicolons, remains similar to the previous approach. However, we observe a decline in F1 scores for quotation marks, colons, and dashes, the reasons for which are not immediately apparent. One hypothesis is that these classes, unlike exclamation marks, are not typically accompanied by distinctive audio features that could aid in differentiation from periods and commas. Alternatively, having positional information of potential punctuation places, the model may default to predicting more frequently represented classes when faced with uncertainty.

Comparing the overall non-blank F1 averages of the two approaches, the model with positional embeddings achieves a best average of 0.502 at the 28,000th step,

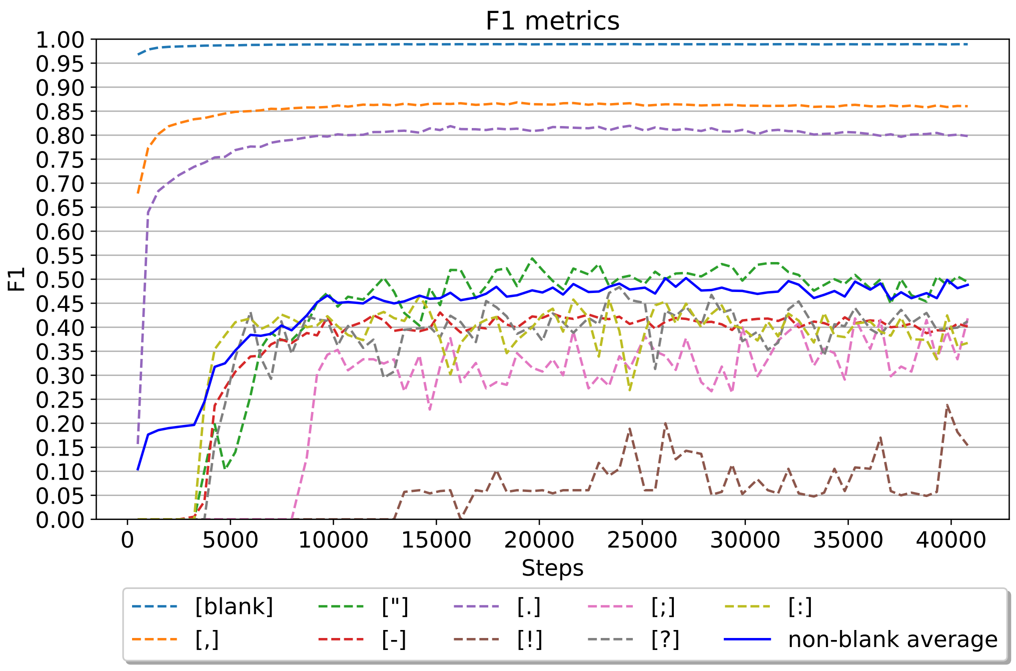


Figure 5.25: F1 score trends for punctuation classes for the model with positional embeddings.

outperforming the previous model's 0.496 at the 27,400th step.

Finally, we conclude our analysis of the new multimodal model with positional embeddings by examining the confusion matrices presented in Figure 5.26. The model demonstrates its best performance at the 24,400th step, with subsequent slight decreases in most classes, except for dashes, semicolons, and colons. This training dynamic aligns with previous experiments, where at first the model learns well-represented classes before gradually improving in less represented ones. Consequently, the highest accuracy scores of 83 and 87 are observed for periods and commas, respectively. The exclamation mark class, being the least represented, shows the lowest accuracy at 16 points. Notably, at the 24,400th step, exclamation marks and periods exhibit the lowest misclassification rates with the blanks. Moreover, since exclamation marks are expected to be followed by richer audio signal, e.g. higher intonation, they are never misclassified with blank class. However, this rationale does not fully explain the high misclassification rate of question marks with the blank class, where we would expect similarly salient audio information. The discrepancy may arise from the structural differences in questions compared to statements, coupled with the model's limited exposure to complete questions.

When comparing the current approach with its predecessor without positional embeddings in terms of confusion matrices (Figure 5.18), we observe higher accuracy scores for exclamation marks in the new model, consistent with the F1 score analysis. However, the new model also shows higher accuracy for quotation marks, suggesting a higher false negative rate for this class. The overall pattern indicates that the new approach is less prone to misclassifying punctuation marks as blanks, particularly for colons, semicolons, and periods. The notable exception is the question mark, which the current model often confuses with periods and commas, a tendency less pronounced in the previous model.

In conclusion, the results from the confusion matrices corroborate our earlier findings: the introduction of positional embeddings enhances the model's ability to accurately identify punctuation marks, with a few notable exceptions, and generally improves differentiation from blank spaces.

5.7.2 Capitalization

With the punctuation restoration analysis complete, we delve into the model's performance on the casing task, with a specific focus on the F1 scores depicted in Figure 5.27. Similar to the punctuation restoration task, particularly in terms of F1 scores, the model exhibits peak performance between the 18,000th and 28,000th steps, achieving the best F1 score of 0.875 at the 24,400th step. After the 28,000th step, a decline in performance on the capitalization class is observed, stabilizing at an F1 score of approximately 0.875. Recalling the model's results on the punctuation restoration task, as illustrated in Figure 5.25, a parallel can be drawn with the performance on the period class, which also demonstrates a performance decay after the 28,000th step.

After examining Figure 5.27 for the results of the multimodal model with positional embeddings, we proceed to contrast this with the previous model's outcomes, which are displayed in Figure 5.19. The previous model, employing an MFCC-based audio encoder devoid of additional positional information, achieved its best F1 score of 0.868 on the capitalization class at the 23,000th step. The integration of positional information in the current model yielded an incremental improvement of 0.007 F1 points on this task. Furthermore, a comparison of the two models' performance trajectories reveals a more rapid convergence in the enhanced model, reaching an F1 score of 0.85 within the initial 7,000 steps, as opposed to approximately 12,000 steps in the preceding model. Both models exhibit a similar trend of slight F1 score reduction after reaching their respective peaks, akin to the trend observed in the punctuation restoration task.

Transitioning to the analysis of the model's performance on the case restoration task via confusion matrices, as depicted in Figure 5.28, the model with positional embeddings demonstrates a swift proficiency in differentiating between blank and capitalization classes, attaining its highest performance around the 22,000th step. However, a plateau is observed post this point, accompanied by a marginal performance decline indicative of overfitting. In contrast, the confusion matrices of the previous model, shown in Figure 5.20, underscore the benefits of positional information. Notably, the model with positional embeddings exhibits higher accuracy and a reduced misclassification rate at each evaluated step. The faster convergence facilitated by positional information is evident, with the new model achieving an accuracy of 85 points in the capitalization class within the first 2,000 steps, a milestone that took approximately 10,000 steps for the previous model to reach.

In summary, the integration of positional embeddings has enhanced the model's performance on the case restoration task, paralleling the improvements observed in the punctuation restoration task. It appears that these advancements can be predominantly attributed to the model's improved handling of the period class, which exhibits a strong correlation with uppercase words.

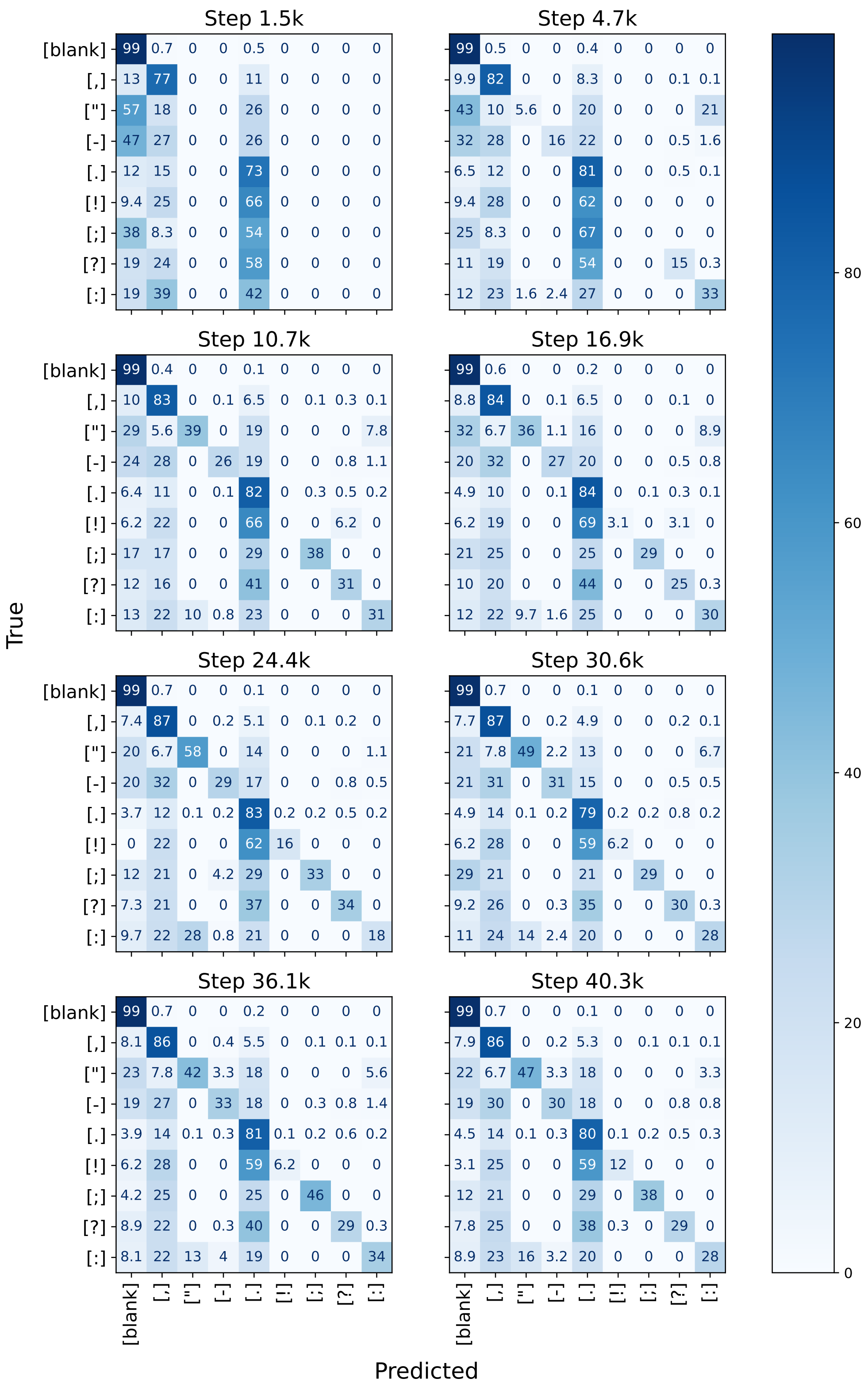


Figure 5.26: Confusion matrices for punctuation classes for the model with positional embeddings.

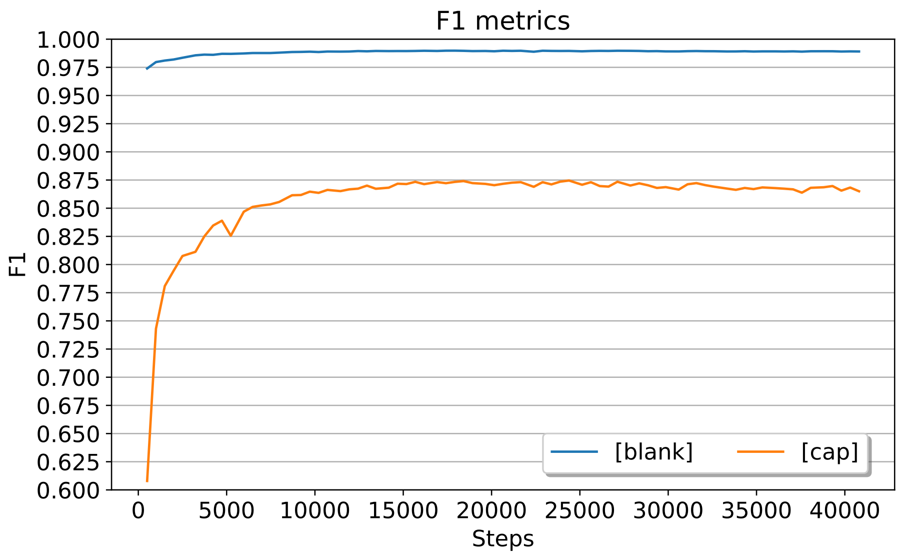


Figure 5.27: F1 score trends for the casing task for the model with positional embeddings.

5.7.3 Multimodal Attention Analysis

Concluding the analysis of the model's performance on punctuation and case restoration tasks, we now turn our attention to the model's attention matrices, as presented in Figure 5.29. The attention plots affirm the model's effective utilization of audio information, evidenced by the absence of degenerate patterns, such as uniform vertical columns, which would suggest a lack of informative value in the audio frames. The attention scores span a considerable range, from 0.02 to 0.25, and maintain consistency across longer sequences. However, the absence of pronounced diagonal patterns in the attention matrices, despite the introduction of positional embeddings, suggests that our initial hypothesis regarding text-audio alignment may require refinement. This observation does not necessarily imply a deficiency in the latent audio representation, as indicated by the absence of vertical attention patterns. Additionally, the upper right corners of the attention matrices generally exhibit lower values, mitigating the risk of early text tokens disproportionately influencing the latter audio frames.

To facilitate cross-comparison between the current model and the previous approach, we revisit Figure 5.21 for the attention matrices associated with the previous approach without positional embeddings. Overall, we observe same large ranges of attention scores, with similar attention patterns. To mitigate selection bias in attention comparison, we can refer to Figure 5.30. This plot displays the maximum attention score at each audio frame for each validation example, followed by averaging the results across the validation dataset. Considering the varying durations of audio inputs, we compute the average score for frame i using only audios with $m \geq i$ latent audio frames. This implies that every example contributes to the statistic at the first frame, while only the longest ones contribute to the average score at the last frame. The light bands around each line represent an interval of $[-\sigma_i, +\sigma_i]$, where σ denotes the standard deviation of the maximum attention scores for the ith frame.

Upon inspection, we note that in most audio frames, the average attention score of the new model with positional embeddings surpasses that of the previous approach without positional information. This aligns with our rationale for integrating shared

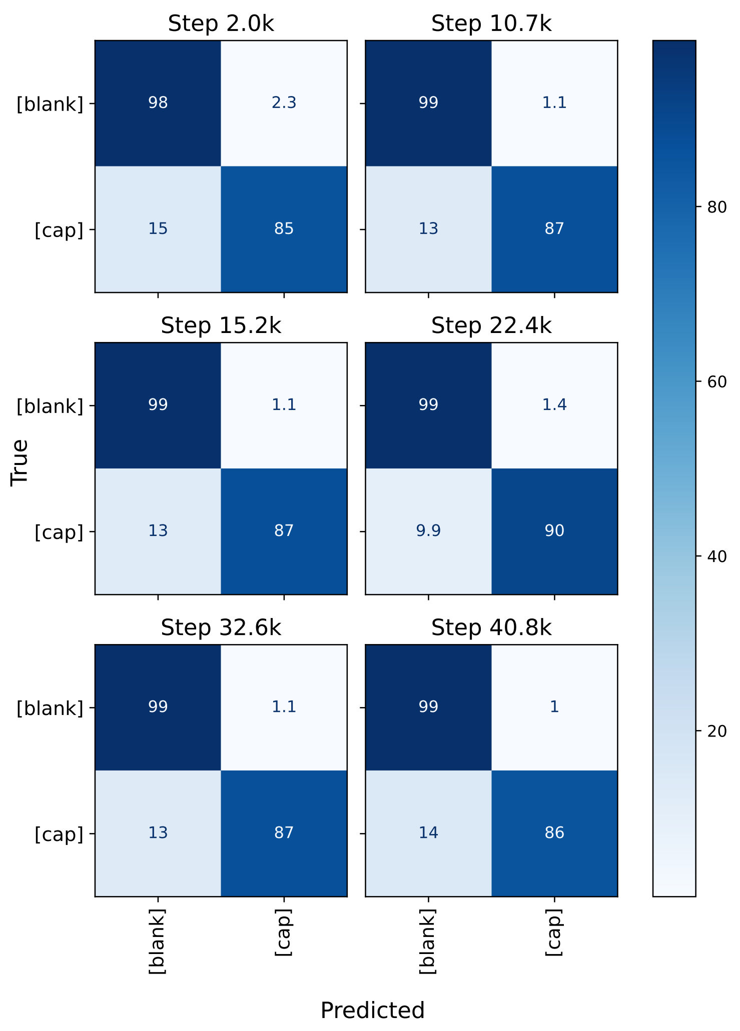


Figure 5.28: Confusion matrices illustrating the performance of the model with positional embeddings in the case restoration task.

positional information, allowing the model to navigate more effectively in the latent audio space and better leverage audio information. Interestingly, both models exhibit a descending trend in attention scores, with higher scores for the initial frames and approximately half the scores for the final ones. This pattern matches with the notion that audio information holds greater relevance for the initial tokens due to the absence of textual context.

In summary, the new model benefits from the shared positional information between audio and text encoders, particularly evident in longer sequences, as indicated by the observations on confusion matrices and PDP-PDR curves.

5.8 Training with Random Audio

In retrospect, the experiments demonstrated that each subsequent approach led to some improvements in model performance over its predecessors. However, as new experiments were conducted, the model architecture became increasingly complex, raising the possibility that gains might be attributable to additional model parameters. To discern the actual benefits of the added sound information, we juxtapose the best

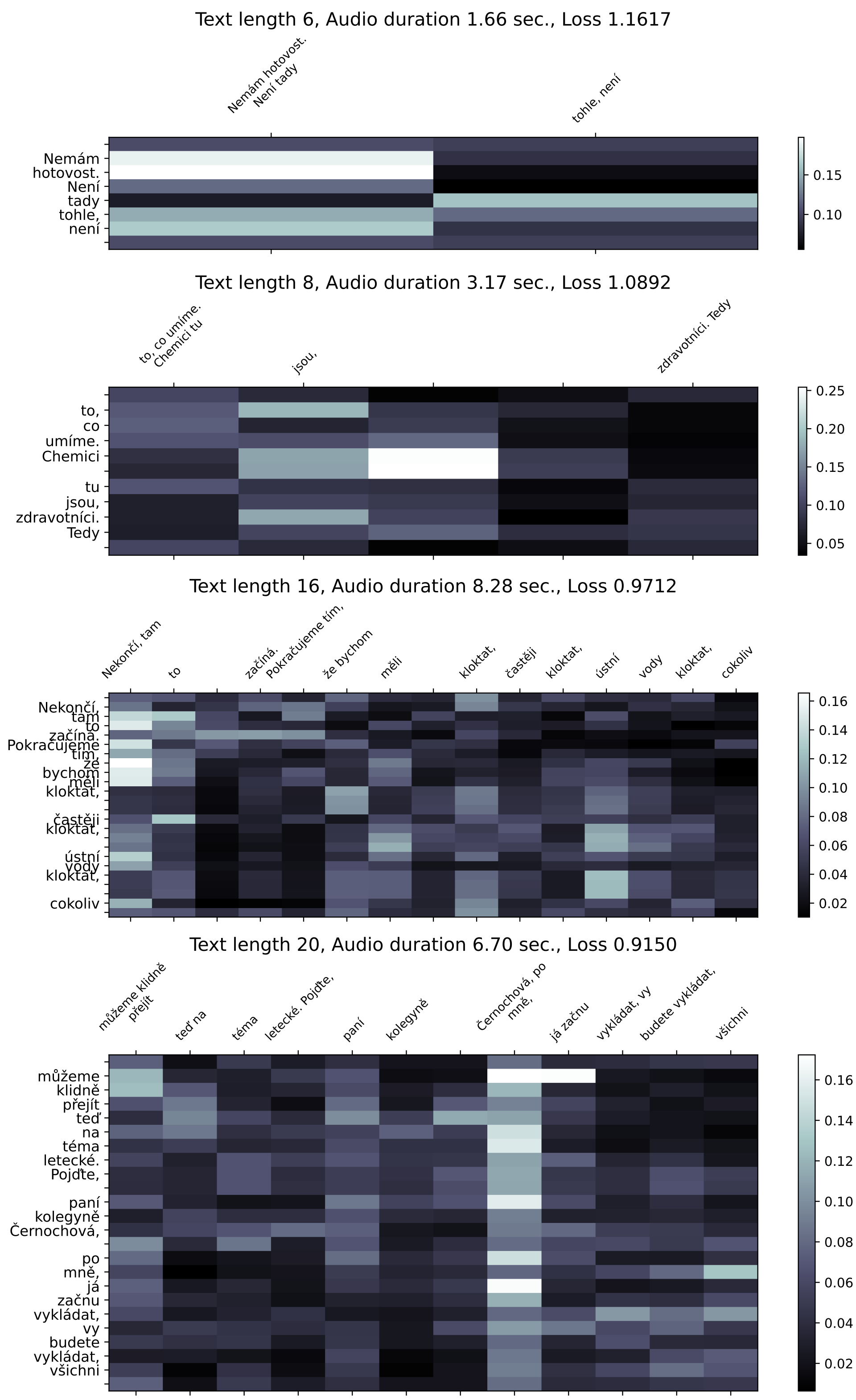


Figure 5.29: Attention matrices for the multimodal model with positional embeddings. The plot structure is described in Section 5.5.3.

model—specifically, the multimodal model with an MFCC-based audio encoder and positional embeddings—against an equivalent model in terms of parameters and architecture but with corrupted audio on the input.

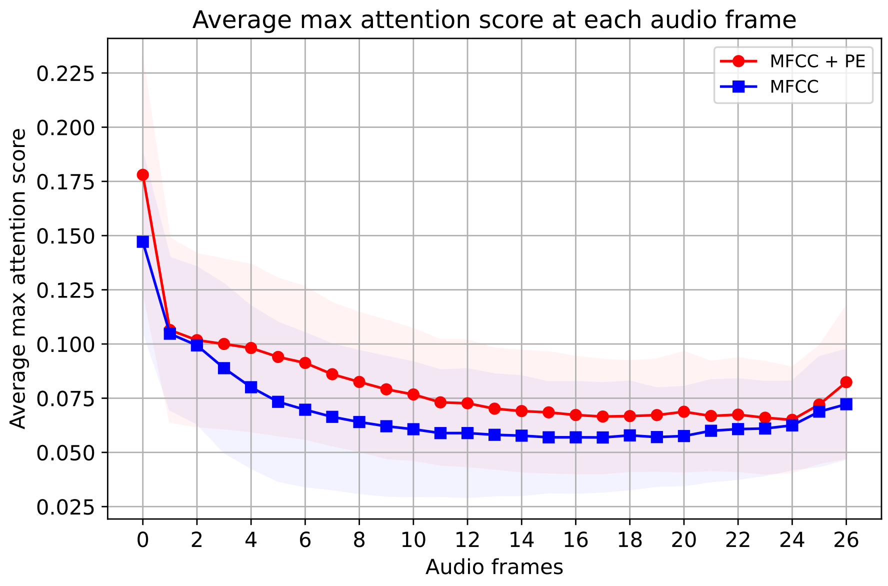


Figure 5.30: Comparative analysis of multimodal cross-attention patterns after integrating positional embeddings. The colored background represents an interval of plus or minus one standard deviation.

One viable method for corrupting the sound involves permutating audio frames prior to their input into the model, thereby disrupting all temporal connections between the frames. The effects of this frame permutation on two distinct audio sequences are visualized in Figure 5.31. The plot demonstrates the impact of frame reordering by displaying both the original and mixed audio signals alongside their spectrograms. After the permutation, the resulting sound closely resembles white noise, indicating successful disruption of temporal frame connections. Additionally, it is worth noting that the final spectrograms and waveforms slightly differ from each other, as they were generated from different audio inputs. This variation can be attributed to the assignment of different samples of white noise to the original audio sequences. However, these different samples of white noise do not convey any meaningful information to the model.

Following the examination of frame permutation effects, we turn our attention to a comparative analysis of previous models against the model with distorted audio sequences. This examination will utilize various F1 scores, specifically the average F1 score on non-blank classes for the punctuation restoration task and the F1 score for the capitalization class, denoted as `[cap]`, for the casing task. Accordingly, four models will be compared: a text-only model, a multimodal model with an MFCC-based audio encoder, and a multimodal model with an MFCC-based audio encoder and positional embeddings for both normal and permutated sound. The corresponding F1 score curves for all mentioned models are depicted in Figure 5.32. These results, measured during training on the validation set, align with previous analyses.

We start the discussion with an analysis of the punctuation restoration task results, as shown in the upper plot of Figure 5.32. The data indicates that the model with corrupted audio input marginally outperforms the text-only baseline, yet falls short of the multimodal models. For instance, the highest non-blank average F1 score for the model with shuffled audio is 0.477, achieved at the 28,500th step, compared to 0.474

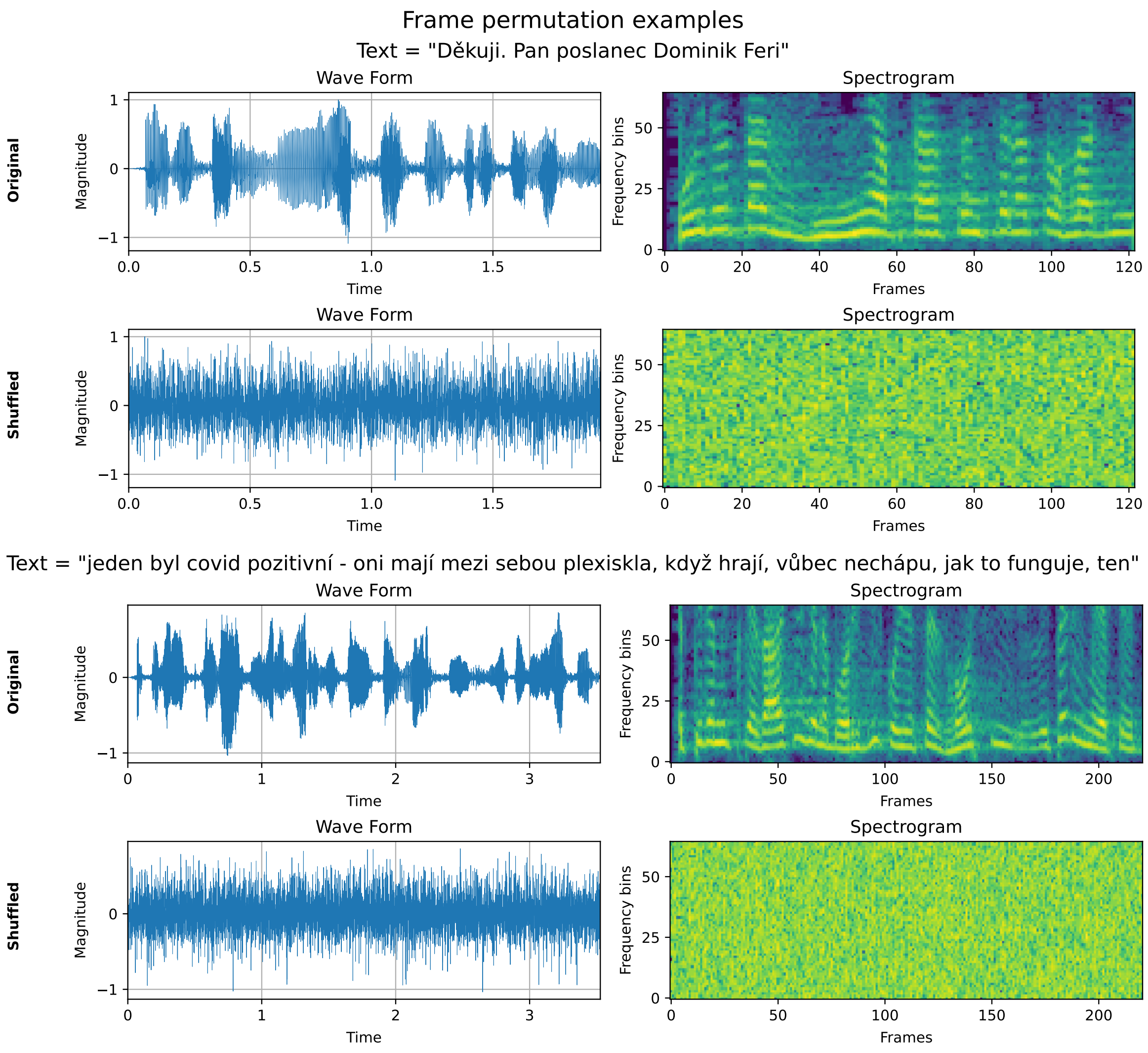
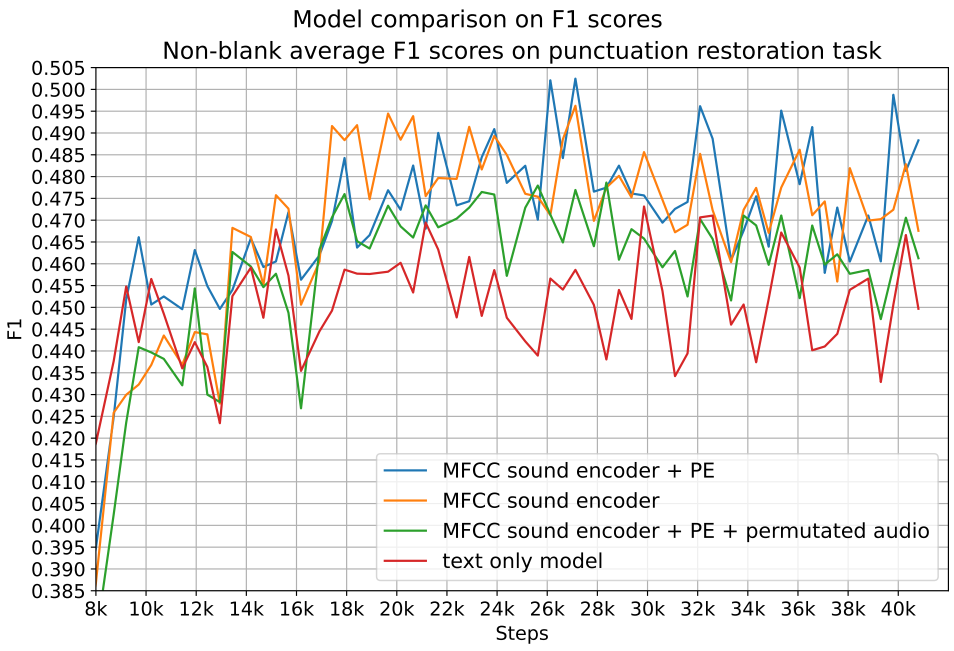


Figure 5.31: Impact of audio frame permutation on signal characteristics.

at the 29,500th step for the text-only model. Furthermore, at most training steps, the multimodal model with positional embeddings exhibits higher scores than the text-only baseline, despite its inability to extract anything useful from the sound. Notably, the peak F1 score for the multimodal model with positional embeddings is 0.502, utilizing the same architectural framework but with meaningful audio input. Therefore, in the punctuation restoration task, the model evidently benefits from a larger architecture with more trainable parameters, although this alone is insufficient to match the results of the sound-informed model.

Turning to the casing task, we examine the F1 scores for different models on the `[cap]` class, as depicted in the lower plot of Figure 5.32. Similarly to the punctuation restoration task, the data suggests that a larger model architecture can be advantageous. Specifically, the model with permutated audio input consistently achieves a higher F1 score on the `[cap]` class compared to the text-only baseline. The peak F1 score for the model with shuffled audio is 0.841, recorded at the 32,000th step, versus 0.837 for the text-only model at the 21,000th step. The F1 curves further illustrate that, for most of the training duration, the larger model with corrupted sound surpasses its smaller text-only counterpart by a significant margin. Nonetheless, all sound-informed models demonstrate superior results, with the best multimodal model with positional



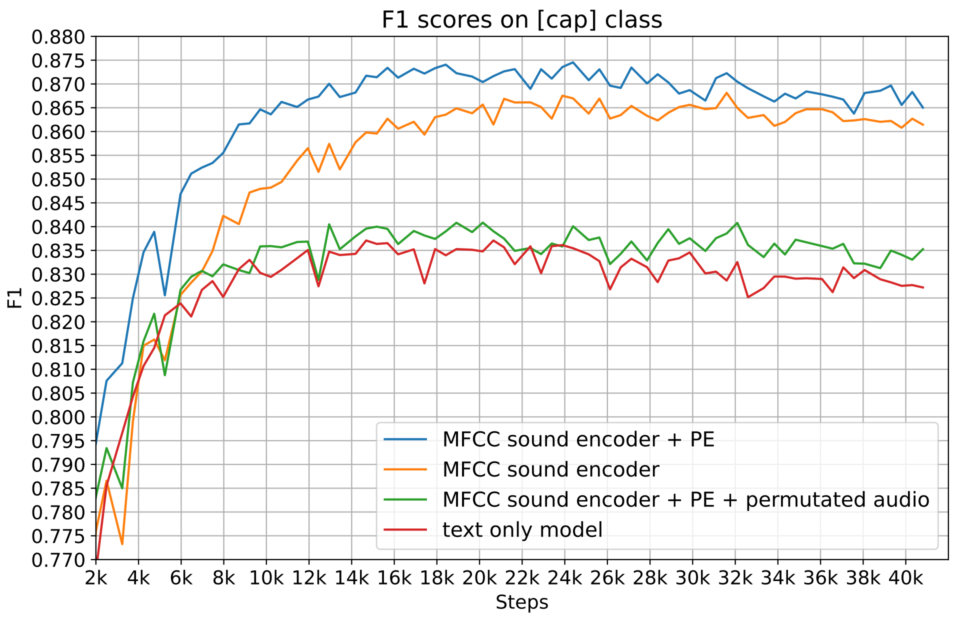


Figure 5.32: Comparative F1 score analysis of different model configurations.

embeddings and true sound input achieving a 0.874 F1 score on the `[cap]` class. Although the influence of sound on casing task results may seem counterintuitive, a plausible explanation is that enhanced performance in punctuation restoration enables the model to better comprehend the input text, thereby achieving higher scores in the casing task.

As confirmed by the experimental data, we can further enhance the performance of the text-only model by employing a more complex architecture with additional trainable parameters. Nevertheless, with the same architecture, multimodal models consistently outperform text-only models by a significant margin on both punctuation restoration and capitalization tasks.

5.9 Evaluation on the Test Set

In the final section, the discussion will be dedicated to comparing the models on the test set, as all contrastive analyses thus far were performed on the validation set. Since all improvements were made using the validation set, the objective is to verify if the same trends hold for an independent test set, which was not used in model training or selection.

We start the comparison of the models with the punctuation restoration task, namely the F1 scores for each punctuation class. The F1 scores for each class and model are presented in Table 5.1, with the highest scores highlighted in bold. Additionally, the table includes the 95% confidence intervals for each score, with intervals computed using bootstrap with 1000 resamplings. Beginning with the comma class, we observe that the multimodal MFCC model with positional embeddings (MM MFCC PE) achieves the highest score, significantly surpassing the text-only baseline and the multimodal model with wav2vec audio encoder. However, compared to the multimodal MFCC model (MM MFCC) without positional information, the improvement is not statistically significant. Shifting to the period class, we note that the MM MFCC PE model significantly outperforms all other models, as their scores fall below its 95% confidence interval. Another punctuation class where we observe significant results is the question mark. Here, all multimodal models outperform the text-only baseline, underscoring the importance of audio features in question mark detection. Interestingly, the text-only model excels only in the quotation mark performance, significantly outperforming the multimodal model with wav2vec 2.0 audio encoder. For the remaining punctuation types, the differences in the results are not statistically significant. It is worth noting that, as commas and periods are the most well-represented punctuation types, each model has relatively tight confidence intervals. However, as the frequency of punctuation classes decreases, the confidence intervals become wider, potentially masking significant results. Another important factor is that remaining punctuation types such as hyphens, semicolons, colons, and possibly quotation marks, are highly style-specific and often interchangeable.

Following the comparison of results on the punctuation restoration task, the subsequent discussion will focus on the comparative evaluation of models on the casing task. As before, due to data imbalance, the F1 metric was chosen for comparison, specifically the F1 score on the capitalization class denoted as `[cap]`. The resulting F1 values are presented in Table 5.1 in the last row. The text-only model has the lowest F1 score, namely 85.25, compared to the other models. Notably, the inclusion of sound modality helps multimodal models achieve significantly better results. Similarly to the experiments on the validation set, the MM MFCC PE model demonstrates the best performance on the capitalization task, although the results are not significantly better compared to the other multimodal approaches. A possible explanation for how sound information can enhance capitalization performance lies in the improved detection of sentence boundaries by multimodal models, i.e., better recognition of periods, question marks, and exclamation marks.

To better understand when our models actually rely on sound information, we compare the performance of the MM MFCC PE model with a text-only baseline across various input lengths. Figure 5.33 shows the F1 scores for the positive class between these two models in the punctuation detection task, where the presence of a punctuation mark is considered a positive label and its absence a negative one. The plot illustrates

Classes	Text Only	MM w2v	MM MFCC	MM MFCC PE
[,]	85.84 (85.3, 86.4)	86.84 (86.3, 87.4)	87.38 (86.9, 87.9)	**87.71** (87.2, 88.2)
[.]	75.26 (74.3, 76.3)	81.47 (80.6, 82.4)	82.12 (81.2, 83.1)	**83.17** (82.3, 84.0)
[-]	41.30 (34.2, 47.8)	43.31 (37.3, 49.9)	**44.31** (37.8, 50.7)	43.47 (36.2, 48.9)
[?]	35.94 (28.0, 44.3)	48.62 (41.1, 55.8)	46.09 (38.0, 54.1)	**50.42** (42.7, 58.7)
[:]	34.48 (23.8, 45.0)	38.46 (28.4, 48.0)	**39.73** (29.6, 49.7)	36.12 (26.2, 45.5)
["]	**45.50** (33.5, 56.2)	32.43 (18.8, 44.8)	40.00 (25.5, 52.1)	40.90 (26.2, 52.6)
[!]	5.41 (0.0, 27.2)	8.07 (1.1, 36.3)	13.33 (4.2, 42.9)	**14.28** (4.7, 43.1)
[;]	45.45 (28.1, 59.4)	39.13 (20.5, 56.6)	50.00 (34.4, 63.2)	**54.28** (40.0, 66.7)
[cap]	85.25 (84.6, 85.9)	87.54 (87.0, 88.2)	87.83 (87.3, 88.4)	**88.18** (87.5, 88.6)

Table 5.1: F1 scores of the models on the capitalization class [cap] and punctuation classes. The best results are shown in bold. Numbers in parenthesis give 95% confidence interval of the F1 score. The classes are sorted according to their frequencies in the training set.

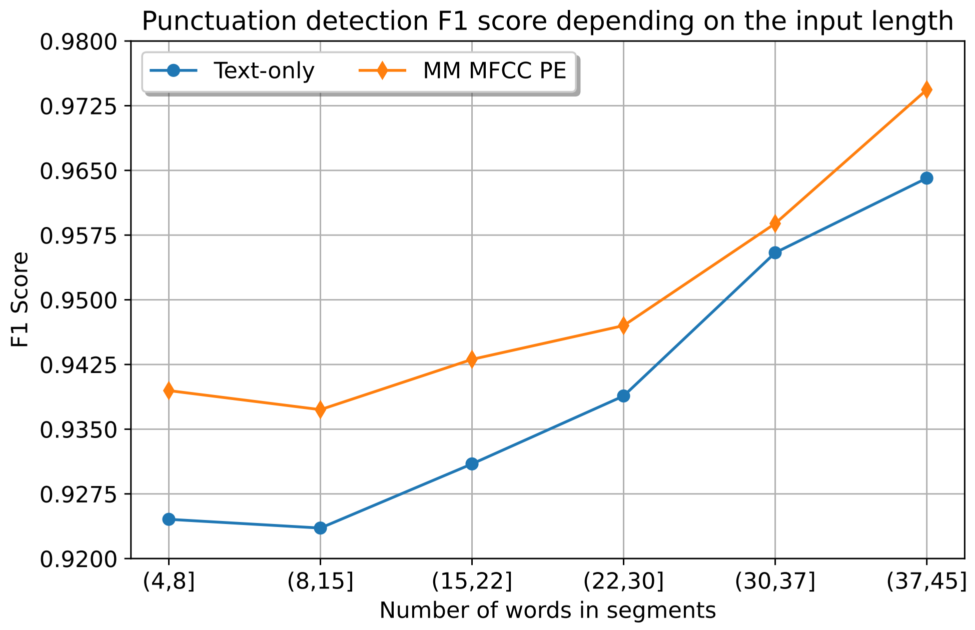


Figure 5.33: Comparison of F1 scores between multimodal and text-only models at varying input lengths, highlighting the impact of audio information on punctuation detection.

the models' performance as a depending on input length, measured by the number of input words. It is clear that our best multimodal model outperforms the text-only baseline across all input length groups. Both models demonstrate better performance with longer input context. Interestingly, as the input length increases, the differences between the models tend to diminish. This trend may be explained by the decreasing value of audio information with larger textual context. This observation shows the importance of additional audio information in scenarios where the input is limited, such as in online cases where the lowest possible latency is desired.

Transitioning to the analysis of the outputs, we present the predictions of the previously mentioned models in Table 5.34. Punctuation predictions of the models are emphasized by square brackets. Note that if a punctuation class appears after the first word in predictions, it is not emphasized, as the models were not trained to predict punctuation or casing after the first word; hence, the true punctuation is simply copied from the data due to the lack of context. The first row always displays the true text with punctuation marks and casings.

Moving forward, the general picture presents some examples where all or the majority of the models make the same mistakes. Starting from the first example, all models managed to detect the end of the sentence and uppercased the next word. However, the text-only model likely missed the end of the sentence and did not uppercase the second word. This sentence illustrates the challenge of predicting the correct punctuation mark at the end of a sentence, as it can be interpreted as both a question and an affirmative proposition, hence the sound input can provide the model with additional information. Consider the second example, where all models predicted a period instead of a dash. This example perfectly demonstrates the importance of left context for correct predictions and the ambiguity of the punctuation restoration task; without left context, a period naturally fits the data. As evidenced by the predictions in the third example, all models were misled by the word “kolegu” and uppercased the word “čunětem”; additionally, only models with better sound processing detected the exclamation mark at the end of the sentence. This example illustrates that models do not fully comprehend Czech grammar and sentence semantics, as the word “čunětem” following “kolegu” likely indicates an insult but not a name. Moving to the fourth example, we observe that models can reliably identify frequent patterns, such as placing a comma after “aby” and uppercasing the word “Poslaneckou” in the phrase “Poslaneckou sněmovnu”. However, they still struggle with semicolons. Specifically, only the multimodal model with positional embeddings managed to place a semicolon and did not uppercase the subsequent word, while other models uppercased the word “za”. A possible reason is that the forced diagonal attention pattern provided by the positional embeddings enabled the model to better extract the meaning of the punctuation mark before the word “za” and thus did not uppercase it.

Concluding our evaluation on the test set, the results observed on the validation set during experiments align well with the performance on the test set. Specifically, the multimodal model with positional embeddings demonstrated the best performance compared to previous experiments. That said, as the analysis of the predictions indicates, there is still room for further improvement.

Models	Texts
True	negativně? Zvýší se riziko daňových úniků? Ministerstvo
Text-only	negativně? zvýší se riziko daňových úniků[.] Ministerstvo
MM wav2vec	negativně? Zvýší se riziko daňových úniků[?] Ministerstvo
MM MFCC	negativně? Zvýší se riziko daňových úniků[.] Ministerstvo
MM MFCC PE	negativně? Zvýší se riziko daňových úniků[?] Ministerstvo
True	zase neodpustím - když jsme tady měli vodní zákon, tak
Text-only	zase neodpustím[.] Když jsme tady měli vodní zákon[,] tak
MM wav2vec	zase neodpustím[.] Když jsme tady měli vodní zákon[,] tak
MM MFCC	zase neodpustím[.] Když jsme tady měli vodní zákon[,] tak
MM MFCC PE	zase neodpustím[.] Když jsme tady měli vodní zákon[,] tak
True	lidí - vymytými mozky a našeho kolegu čunětem! Pokud
Text-only	lidí - vymytými mozky a našeho kolegu Čunětem[.] pokud
MM wav2vec	lidí - vymytými mozky a našeho kolegu Čunětem[.] Pokud
MM MFCC	lidí - vymytými mozky a našeho kolegu Čunětem[!] Pokud
MM MFCC PE	lidí - vymytými mozky a našeho kolegu Čunětem[!] Pokud
True	schválit v předloženém znění; za třetí zmocňuje zpravodajku výboru, aby s usnesením seznámila Poslaneckou sněmovnu
Text-only	schválit v předloženém znění[;] Za třetí zmocňuje zpravodajku výboru[,] aby s usnesením seznámila Poslaneckou sněmovnu
MM wav2vec	schválit v předloženém znění[.] za třetí zmocňuje zpravodajku výboru[,] aby s usnesením seznámila Poslaneckou sněmovnu
MM MFCC	schválit v předloženém znění[;] Za třetí zmocňuje zpravodajku výboru[,] aby s usnesením seznámila Poslaneckou sněmovnu
MM MFCC PE	schválit v předloženém znění[;] za třetí zmocňuje zpravodajku výboru[,] aby s usnesením seznámila Poslaneckou sněmovnu
True	nejmíň subjektů. Protože nedokážu si představit tu kritiku, kdyby náhodou se něco nepovedlo, tak co asi sklidíme. Tak to
Text-only	nejmíň subjektů[,] protože nedokážu si představit tu kritiku[,] kdyby náhodou se něco nepovedlo[,] tak co asi sklidíme[.] Tak to
MM wav2vec	nejmíň subjektů[.] Protože nedokážu si představit tu kritiku[.] Kdyby náhodou se něco nepovedlo[,] tak co asi sklidíme[?] Tak to
MM MFCC	nejmíň subjektů[,] protože nedokážu si představit tu kritiku[,] kdyby náhodou se něco nepovedlo[.] Tak co asi sklidíme[?] Tak to
MM MFCC PE	nejmíň subjektů[.] Protože nedokážu si představit tu kritiku[,] kdyby náhodou se něco nepovedlo[,] tak co asi sklidíme[?] Tak to

Figure 5.34: Predictions of the models on the test set. Predictions on the punctuation classes are presented with brackets. The first punctuation marks or capitalization were not used in training and is copied from the data.

6. Conclusion

In this thesis, we explored combining audio input with the corresponding automatic text transcripts to improve performance on the inverse text normalization task, focusing specifically on punctuation restoration and capitalization. To compare text-only and multimodal approaches, we adapted the existing ParCzech 3.0 dataset. Particularly, to address certain limitations of ParCzech 3.0, such as the constrainedness of transcripts to sentence boundaries and imprecise alignment with the official transcript, we trained an ASR model and used it to recognize all the original audio files available in the dataset. Following the ParCzech 3.0 methodology, we aligned our recognized version of the transcripts with the official ones. This recognition and alignment process allowed us to extract more data from the official transcripts in comparison to ParCzech 3.0, even though we required a perfect match between the recognized and official transcripts. Also, the created segments were no longer aligned to sentence boundaries, offering a more realistic scenario for ITN use cases.

To enable integration of the sound information into the ITN neural model, we developed a multimodal cross-attention module inspired by the original Transformer architecture. This module utilizes the attention mechanism to establish a soft alignment between text input and corresponding audio input. We experimented with various methods of integrating audio input into the ITN model. The simplest method was motivated by the pretraining framework, where we used an already trained ASR model as a trainable sound encoder. However, attention plots indicated that this integration of audio information was not particularly beneficial for the ITN task. To enhance audio utilization, we created a custom audio encoder inspired by the wav2vec 2.0 architecture, adding more convolutional blocks and reducing the number of Transformer encoder blocks. This architecture showed promising results, improving the utility of the corresponding sound input. To further assist the multimodal model in navigating the audio latent space, we introduced multimodal positional embeddings that enhanced audio utilization by the multimodal model.

Evaluation on the test set showed that the speech-informed models achieved statistically significant improvements in the detection of capitalization class and detection of the most exemplified punctuation classes, specifically periods, commas, and question marks. These punctuation marks are generally more deterministic compared to others like dashes, colons, quotation marks, and exclamation marks, which are heavily influenced by stylistic choices. Furthermore, the differences in performance on less represented classes were not statistically significant. Our experiments also demonstrated that the value of audio information varies with input length. For instance, in shorter inputs (fewer than 10 words), the multimodal model relies heavily on sound information due to limited textual context. As input length increases, the relevance of sound information diminishes.

For future work, we plan to address certain limitations we encountered in the data and model the ITN task as a sequence-to-sequence generation problem. Currently, our approach requires perfect alignment between the recognized and official transcripts, allowing us to treat the ITN problem in a sequence labeling framework and address punctuation restoration and capitalization tasks as token classification tasks. However, this approach excludes certain ITN subtasks, such as inverse number normalization (converting verbalized numbers to digit form), inverse abbreviation normalization (converting verbalized abbreviations to their shorter written forms), removal of disfluencies (like self-corrections or repetitions), and potentially correcting ASR errors. To incorporate all these aspects into our training and evaluation dataset, we would need a deeper analysis of ParCzech 3.0 with a more sophisticated alignment procedure and detailed benchmarks on the ASR system to ensure accurate recognition of digits and abbreviations and precise transcription of disfluencies. With such a dataset, we would be able to model the ITN problem within machine translation framework, where the goal would be to modify the recognized transcript so that it better corresponds to the provided official transcript.

Bibliography

R. Ardila, M. Branson, K. Davis, M. Henretty, M. Kohler, J. Meyer, R. Morais, L. Saunders, F. M. Tyers, and G. Weber. Common voice: A massively-multilingual speech corpus. In *Proceedings of the 12th Conference on Language Resources and Evaluation (LREC 2020)*, pages 4211–4215, 2020.

Mikhail Arkhipov, Maria Trofimova, Yuri Kuratov, and Alexey Sorokin. Tuning multilingual transformers for language-specific named entity recognition. In Tomaž Erjavec, Michał Marcińczuk, Preslav Nakov, Jakub Piskorski, Lidia Pivovarova, Jan Šnajder, Josef Steinberger, and Roman Yangarber, editors, *Proceedings of the 7th Workshop on Balto-Slavic Natural Language Processing*, pages 89–93, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-3712. URL `https://aclanthology.org/W19-3712`.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 12449–12460. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper_files/paper/2020/file/92d1e1eb1cd6f9fba3227870bb6d7f07-Paper.pdf`.

Varnith Chordia. PunKtuator: A multilingual punctuation restoration system for spoken and written text. In Dimitra Gkatzia and Djamé Seddah, editors, *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 312–320, Online, April 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.eacl-demos.37. URL `https://aclanthology.org/2021.eacl-demos.37`.

Christopher Cieri, David Miller, and Kevin Walker. The fisher corpus: a resource for the next generations of speech-to-text. In Maria Teresa Lino, Maria Francisca Xavier, Fátima Ferreira, Rute Costa, and Raquel Silva, editors, *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC'04)*, Lisbon, Portugal, May 2004. European Language Resources Association (ELRA). URL `http://www.lrec-conf.org/proceedings/lrec2004/pdf/767.pdf`.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.747. URL `https://aclanthology.org/2020.acl-main.747`.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill

Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019a. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL `https://aclanthology.org/N19-1423`.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019b. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL `https://aclanthology.org/N19-1423`.

Trinh–Minh–Tri Do and Thierry Artieres. Neural conditional random fields. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 177–184, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL `https://proceedings.mlr.press/v9/do10a.html`.

William Falcon and The PyTorch Lightning team. PyTorch Lightning, March 2019. URL `https://github.com/Lightning-AI/lightning`.

A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional lstm networks. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 4, pages 2047–2052 vol. 4, 2005. doi: 10.1109/IJCNN.2005.1556215.

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, page 369–376, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595933832. doi: 10.1145/1143844.1143891. URL `https://doi.org/10.1145/1143844.1143891`.

Kenneth Heafield. KenLM: Faster and smaller language model queries. In Chris Callison-Burch, Philipp Koehn, Christof Monz, and Omar F. Zaidan, editors, *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 187–197, Edinburgh, Scotland, July 2011. Association for Computational Linguistics. URL `https://aclanthology.org/W11-2123`.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2023.

Qiushi Huang, Tom Ko, H. Lilian Tang, Xubo Liu, and Bo Wu. Token-Level Supervised Contrastive Learning for Punctuation Restoration. In *Proc. Interspeech 2021*, pages 2012–2016, 2021. doi: 10.21437/Interspeech.2021-661.

Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33:117–28, 01 2011. doi: 10.1109/TPAMI.2010.57.

Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Matyáš Kopp, Vladislav Stankov, Jan Oldřich Krůza, Pavel Straňák, and Ondřej Bojar. Parczech 3.0: A large czech speech corpus with rich metadata. In Kamil Ekštein, František Pártl, and Miloslav Konopík, editors, *Text, Speech, and Dialogue*, pages 293–304, Cham, 2021. Springer International Publishing. ISBN 978-3-030-83527-9.

Jan Oldřich Krůza. Czech parliament meeting recordings as ASR training data. In *Proceedings of the 2020 FCCSIS*, volume 21 of *Annals of Computer Science and Information Systems*, pages 185–188. IEEE, 2020. URL `http://dx.doi.org/10.15439/2020F119`.

Ludwig Kürzinger, Dominik Winkelbauer, Lujun Li, Tobias Watzel, and Gerhard Rigoll. Ctc-segmentation of large corpora for german end-to-end speech recognition. In Alexey Karpov and Rodmonga Potapova, editors, *Speech and Computer*, pages 267–278, Cham, 2020. Springer International Publishing. ISBN 978-3-030-60276-5.

Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, R. Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989. URL `https://proceedings.neurips.cc/paper_files/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf`.

Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander Rush, and Thomas Wolf. Datasets: A community library for natural language processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 175–184, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. URL `https://aclanthology.org/2021.emnlp-demo.21`.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692, 2019. URL `https://api.semanticscholar.org/CorpusID:198953378`.

Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970. ISSN 0022-2836. doi: https://doi.org/10.1016/0022-2836(70)90057-4. URL `https://www.sciencedirect.com/science/article/pii/0022283670900574`.

Alp Öktem, Mireia Farrús, and Leo Wanner. Attentional parallel rnns for generating punctuation in transcribed speech. In Nathalie Camelin, Yannick Estève, and Carlos Martín-Vide, editors, *Statistical Language and Speech Processing*, pages 131–142, Cham, 2017. Springer International Publishing. ISBN 978-3-319-68456-7.

Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: An asr corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210, 2015. doi: 10.1109/ICASSP.2015.7178964.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. *PyTorch: an imperative style, high-performance deep learning library.* Curran Associates Inc., Red Hook, NY, USA, 2019.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In David E. Rumelhart and James L. Mcclelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, pages 318–362. MIT Press, Cambridge, MA, 1986.

Jakub Sido, Ondřej Pražák, Pavel Přibáň, Jan Pašek, Michal Seják, and Miloslav Konopík. Czert – Czech BERT-like model for language representation. In Ruslan Mitkov and Galia Angelova, editors, *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2021)*, pages 1326–1338, Held Online, September 2021. INCOMA Ltd. URL `https://aclanthology.org/2021.ranlp-1.149`.

Andrew Silva, Barry-John Theobald, and Nicholas Apostoloff. Multimodal punctuation prediction with contextual dropout. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3980–3984, 2021. doi: 10.1109/ICASSP39728.2021.9414979.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, jan 2014. ISSN 1532-4435.

Milan Straka. UDPipe 2.0 Prototype at CoNLL 2018 UD Shared Task. In *Proceedings of the CoNLL 2018 ST: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 197–207. ACL, 2018. URL `http://dx.doi.org/10.18653/v1/K18-2020`.

Milan Straka, Jakub Náplava, Jana Straková, and David Samuel. Robeczech: Czech roberta, a monolingual contextualized language representation model. In Kamil Ekštein, František Pártl, and Miloslav Konopík, editors, *Text, Speech, and Dialogue*, pages 197–209, Cham, 2021. Springer International Publishing. ISBN 978-3-030-83527-9.

Monica Sunkara, Srikanth Ronanki, Dhanush Bekal, Sravan Bodapati, and Katrin Kirchhoff. Multimodal semi-supervised learning framework for punctuation prediction in conversational speech, 2020a.

Monica Sunkara, Srikanth Ronanki, Kalpit Dixit, Sravan Bodapati, and Katrin Kirchhoff. Robust prediction of punctuation and truecasing for medical ASR. In Parminder Bhatia, Steven Lin, Rashmi Gangadharaiah, Byron Wallace, Izhak Shafran, Chaitanya Shivade, Nan Du, and Mona Diab, editors, *Proceedings of the First Workshop on Natural Language Processing for Medical Conversations*, pages 53–62, Online, July 2020b. Association for Computational Linguistics. doi: 10.18653/v1/2020.nlpmc-1.8. URL `https://aclanthology.org/2020.nlpmc-1.8`.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 3104–3112, Cambridge, MA, USA, 2014. MIT Press.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

Changhan Wang, Morgane Riviere, Ann Lee, Anne Wu, Chaitanya Talnikar, Daniel Haziza, Mary Williamson, Juan Pino, and Emmanuel Dupoux. VoxPopuli: A large-scale multilingual speech corpus for representation learning, semi-supervised learning and interpretation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 993–1003, Online, August 2021. Association for Computational Linguistics. URL `https://aclanthology.org/2021.acl-long.80`.

Yao-Yuan Yang, Moto Hira, Zhaoheng Ni, Anjali Chourdia, Artyom Astafurov, Caroline Chen, Ching-Feng Yeh, Christian Puhrsch, David Pollack, Dmitriy Genzel, Donny Greenberg, Edward Z. Yang, Jason Lian, Jay Mahadeokar, Jeff Hwang, Ji Chen, Peter Goldsborough, Prabhat Roy, Sean Narenthiran, Shinji Watanabe, Soumith Chintala, Vincent Quenneville-Bélair, and Yangyang Shi. Torchaudio: Building blocks for audio and speech processing. *arXiv preprint arXiv:2110.15018*, 2021.

List of Figures

List of Tables

List of Abbreviations

ASR	Automatic Speech Recognition.
CTC	Connectionist Temporal Classification.
CNN	Convolutional Neural Network.
GMM	Gaussian Mixture Model.
HMM	Hidden Markov Model.
ITN	Inverse Text Normalization.
LM	Language Model.
MFCC	Mel-Frequency Cepstral Coefficients.
MLM	Masked Language Modeling.
MM	Multimodal.
NLP	Natural Language Processing.
NSP	Next Sentence Prediction.
PDP	Punctuation Detection Precision.
PDR	Punctuation Detection Recall.
PE	Positional Embeddings.
RNN	Recurrent Neural Network.
SCL	Supervised Contrastive Loss.
WER	Word Error Rate.

A. Attachments

The URL `https://github.com/Stanvla/ThesisITN` contains the source codes used to create the dataset and perform the experiments. The scripts are organized into the following folders.

wav2vec_ctc

This folder contains scripts for generating the specialized ITN dataset from ParCzech 3.0.

- `wav2vec_ctc/wav2vec_inference.py`: Runs inference using a trained wav2vec 2.0 model to produce recognized transcripts and alignment.
- `wav2vec_ctc/dataset-review.py`: Filters and modifies aligned segments. Finally, generates training, validation, and testing sets from clean segments.

punc_restoration

This folder includes scripts for training and evaluating the ITN models.

- `punc_restoration/sequence_labeling.py`: Contains code for training and running inference on text-only and multimodal models.
- `punc_restoration/multi_modal_attn.py`: Implements the multimodal cross-attention module.
- `punc_restoration/load_data.py`: Defines the structure for multimodal and text-only datasets.