FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

**MASTER THESIS**

Borek Požár

# Robust language understanding for a voice-based dialogue system

Institute of Formal and Applied Linguistics

|  |  |
|---|---|
| Supervisor of the master thesis: | Mgr. et Mgr. Ondřej Dušek, Ph.D. |
| Study programme: | Language Technologies and Computational Linguistics |

Prague 2024

I want to thank my supervisor, Mgr. et Mgr. Ondřej Dušek, Ph.D., for his consultations provided regardless of time and date. I also extend my thanks to Mgr. Ondřej Plátek for his valuable last-minute suggestions.

Thanks to all my family and friends who hosted me during the work on this thesis, to Lenka for her assistance with figures and other support, and to Klárka for sharing in the challenges of our studies.

Title: Robust language understanding for a voice-based dialogue system

Author: Borek Požár

Institute: Institute of Formal and Applied Linguistics

Supervisor: Mgr. et Mgr. Ondřej Dušek, Ph.D., Institute of Formal and Applied Linguistics

Abstract: Voice-based task-oriented dialogue systems enable users to complete tasks on computers using their voice. A critical part of spoken language understanding (SLU) in these voice-based dialogue systems is slot filling, i.e., extracting task-specific pieces of information from the utterance. We explore this task in the public transport search domain, where necessary slots include the names of departure and arrival locations. We compile a dataset aimed at spoken slot filling in Czech in this domain, together with baseline results utilizing a speech recognition and rule-based slot filling pipeline. Our primary research question is: How can we utilize prior knowledge of possible slot values (having them in a database) to achieve robust slot filling in SLU with general off-the-shelf automatic speech recognition (without fine-tuning the speech model)? We try to answer this question by proposing a novel architecture that integrates SLU by adding a new decoder layer to the Whisper speech recognition pre-trained model. Unfortunately, the performance of our architecture does not surpass the baseline.

Keywords: spoken language understanding, language understanding, dialogue systems, slot detection

Název práce: Robustní porozumění jazyku pro hlasový dialogový systém

Autor: Borek Požár

Ústav: Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: Mgr. et Mgr. Ondřej Dušek, Ph.D., Ústav formální a aplikované lingvistiky

Abstrakt: Hlasové dialogové systémy orientované na úkoly umožňují uživatelům dosahovat cílů na počítači pomocí hlasu. Důležitou součástí porozumění mluvené řeči (SLU) v těchto hlasových dialogových systémech je vyplňování slotů, tj. extrakce informací specifických pro daný úkol z výpovědi. Tuto úlohu zkoumáme v oblasti vyhledávání ve veřejné dopravě, kde potřebné sloty zahrnují například názvy míst odjezdu a příjezdu. Sestavili jsme dataset zaměřený na vyplňování mluvených slotů v češtině v této oblasti, spolu s baseline výsledky využívajícími pipeline rozpoznávání řeči (ASR) a textového vyplňování slotů založeného na pravidlech. Naše hlavní výzkumná otázka zní: Jak můžeme využít předchozí znalosti možných hodnot slotů (máme je v databázi) k dosažení robustního vyplňování slotů v SLU pomocí obecného předtrénovaného modelu ASR (bez fine-tuningu ASR modelu)? Na tuto otázku se snažíme odpovědět návrhem nové architektury, která integruje SLU přidáním nové vrstvy dekodéru k předtrénovanému modelu rozpoznávání řeči Whisper. Výkonnost naší architektury bohužel nepřekonává baseline.

Klíčová slova: porozumění mluvené řeči, porozumění jazyku, dialogové systémy, detekce slotů

# Contents

# Introduction

Advances in *Natural Language Processing (NLP)* transformed the way we interact with computers. It became possible to progress from rigidly structured control flows to communication in natural language. Programs that enable this communication are called *dialogue systems*. Today's most famous one is probably ChatGPT, a chatbot based on *Large Language Model (LLM)*.

While LLMs present unparalleled performance in many NLP tasks, they are not a universal solution to all of them. One such area where they cannot be utilized without further tweaks is voice-based *Task-Oriented Dialogue (TOD)*. These dialogues are aimed at specific domains or applications, such as travel booking, restaurant reservations, or customer service. Task-oriented dialogue systems typically follow a predefined dialogue flow, guiding users through a series of prompts and responses to achieve specific tasks or objectives. LLMs cannot directly handle text-based TODs well (Cao, 2024; Hudecek; Dusek, 2023; Heck et al., 2023) and the need to process audio adds further complexity to the problem (M. He; Garner, 2023). We will focus on the topic of voice-based TODs in this thesis.

In standard voice-based systems, further processing relies on *Automatic Speech Recognition (ASR)*. While ASR also significantly improved in the last decades, its performance is still far from perfect, especially when faced with a lot of background noise or uncommon words. The so-called *Spoken Language Understanding (SLU)* — interpreting the meaning of spoken user utterance (usually from its recording) — thus becomes much more complex than the same task with textual input (Tur; De Mori, 2011).[1] The effectiveness of task-oriented dialogue systems depends on their ability to comprehend user intents and extract relevant information from their utterances. Robust SLU capabilities are therefore necessary. However, any ASR error propagates through the processing pipeline and usually results in the inability to fulfill the goal of the task.

The comprehension of user intent is usually called *intent detection*. It involves assigning some primary meaning (or more of them) to the user utterance (Tür et al., 2010). For example, when adding an event to a personal calendar, we could detect that the user's primary intent is to add an event and that they are also informing us about its date. Since we decided not to focus on this part of SLU, we will not discuss any more details.

The extraction of relevant information is usually performed in a process called *slot filling*. A list of *slots*, specific pieces of information that need to be extracted from a sentence, is defined to reach some goal of the conversation (Y. He; S. J. Young, 2005). To continue our example with adding an event to a calendar, these could include a name, date, time, and location for the event. The system then tries to fill these slots with *values* extracted from the user utterances. Slot filling in SLU will be the main focus of this thesis.

[1]When the input is textual instead of audio, the term *Natural Language Understanding (NLU)* is usually used. This does not always hold true; these terms are often mixed in the literature.

# Aims of This Thesis

Slot filling in SLU is especially prone to ASR errors mainly for two reasons. Imagine a situation where the user tells the system he wants to start navigation to a particular city. Names are often unique and obscure. Therefore, ASR is more likely to make a mistake than with common words. Then, once a slot is filled with the detected value, even a single wrongly transcribed letter can cause a failure because the city will not be found in the database.

The main question we decided to target in this thesis can be formulated quite simply:

> How can we utilize prior knowledge of possible slot values (e.g., having them in a database) to achieve robust slot filling in SLU with general off-the-shelf ASR (without fine-tuning the ASR model)?

As we shall show later, answering this question is not simple at all.

The reason we opt not to fine-tune the ASR is straightforward. If we allow ASR fine-tuning, the solution would be to fine-tune ASR for the given task and teach it all the specific words. This way, we would achieve high-quality transcripts, and widespread NLU methods could be used. However, this would mean we need a specific fine-tuned ASR for every task. As ASR is a complex problem, it requires fairly large models (Amodei et al., 2016). Therefore, fine-tuning and running task-specific ASR would be computationally challenging and costly. Instead, we aim to use general ASR with a task-specific SLU slot-filling module. Machine learning models in this module should be lightweight compared to the ASR models, resulting in lower overall costs.

We aim to review existing literature relevant to this question, draw some inspiration from it, develop our own approach, and apply it to selected datasets. As there is no suitable dataset in the Czech language for this problem, we create one in the process.

# Structure of This Thesis

The first three chapters aim to introduce the theory relevant to the problem in question. Chapter 1 gives an overview of ASR methods and a general introduction to the machine learning methods. We summarize the ASR architectures most important for our work in Section 1.9. In Chapter 2, we introduce common approaches to textual slot filling (in NLU). Building on this knowledge, Chapter 3 discusses ways to deal with ASR errors in SLU.

Then, we focus on our own approaches in Chapter 4. We present a slot-filling dataset in Czech in Section 4.1. In Section 4.2, we describe a baseline system and propose a novel SLU architecture. Section 4.3 details utilized training and inference methods, and finally, in Section 4.4, we focus on the experimental results and error analysis.

# 1 Automatic Speech Recognition

Automatic Speech Recognition (ASR) is the first module of most voice-based dialogue system pipelines, enabling machines to transcribe spoken language into text. The text is then processed to extract meaning and form a response.

We start this chapter by covering the basics of audio processing in Section 1.1 followed by an introduction to language models in Section 1.2. This introduction is useful for introducing machine learning fundamentals in Section 1.3. Neural networks and their advanced architectures are discussed in Sections 1.4 and 1.6, interleaved by Section 1.5 about input representations. In Section 1.7, we discuss how neural networks are trained. Though focused on ASR, the description of general machine learning methods serves as a basis also for later Chapters 2 and 3 focused on slot filling. A summary of traditional statistical ASR approaches is presented in Section 1.8. Finally, Section 1.9 covers modern open-source pre-trained ASR models essential for our later work.

## 1.1 Audio Data

Audio data is the raw input upon which transcription and understanding tasks are based. In this section, we talk about how audio is recorded and saved on computers, what fundamental properties influence the quality of the recording, and what features we can extract from the recording for use in ASR.

### 1.1.1 Recording of Audio

Audio, as we know it, is the vibration of the air around us as the sound waves propagate through space. The waves are generated by acoustic sources, such as human speech, musical instruments, or ambient noise. To capture the audio, acquiring these sound waves and transforming them into electrical signals is necessary. These signals can then be stored and digitally processed (Gold et al., 2011). The sound wave in the physical world is continuous; it theoretically contains an infinite amount of information, which poses a challenge for recording the audio. To store it on computers, we need to decide how to store the relevant information from the sound wave in a limited memory.

Several components are involved in the recording process, including microphones, analog-to-digital converters (ADCs), and recording devices. Microphones are transducer devices that convert acoustic waves into electrical signals. As the energy of a sound wave is usually very low, the generated electric current is also low. Therefore, it needs to be amplified and only then measured. The measured signal is then passed to an *analog-to-digital converter (ADC)*, which converts the continuous electrical signal into a digital signal. The ADC samples the signal periodically, measures its amplitude, and stores this value.

The frequency at which the signal is sampled is called the *sample rate* and is usually measured in Hertz (Hz), i.e., we measure how many measurements are done in a second (Oppenheim, 1999). The number of bits used to represent the amplitude of the signal is called the *bit depth*. Greater bit depth results in a more precise representation of the signal. The combination of the sample rate and bit depth determines the quality of the recording.

Common sampling rates include 8 kHz, 16 kHz, 44.1 kHz (CD quality), and 48 kHz (DVD quality), but special equipment for certain applications today can get up to hundreds or even thousands of kHz. Higher sampling rates capture more detail but require more storage space and processing power. The sampling rate determines the maximum frequency we can reconstruct from the digital recording. This is known as the Nyquist limit since he was the first to imply this fact (Nyquist, 2002).

Frequencies below 10 kHz are essential in human speech. Therefore, a sampling rate of 16 kHz is usually chosen as a compromise for use in ASR (Jurafsky; Martin, 2024). Sometimes, only an 8 kHz sampling rate is used, but recording captured this way can represent only frequencies up to 4 kHz. Such recording thus may sound muffled, and the missing features may pose a challenge for ASR systems. Common values of the bit depth include 8-bit, 16-bit, and 24-bit. Higher bit depths allow for more precise representation of audio signals but also result in larger file sizes.

The digitized audio data is then stored using a recording device such as a computer sound card. Factors such as background noise, microphone placement, and microphone type can significantly affect the clarity and intelligibility of the recorded speech.

Voice-based dialogue systems are often used in suboptimal conditions. These can include smart speakers where the user speaks meters away from the microphone or smartphone assistants used on the go in noisy surroundings. Understanding the resulting low-quality recordings is challenging. A noisy recording is hard to transcribe for ASR, and noisy transcription is, in turn, hard to process for NLU.

### 1.1.2 Input Features for Further Processing

Once audio data is recorded, it is typically stored as an array of amplitudes representing the sound wave over time. We can visualize it in a graph called a waveform, as shown in Figure 1.1. While waveforms provide basic information about the audio signal, they may not be the best input for ASR tasks due to their lack of frequency information. Therefore, various transformations are applied to convert the raw audio data into feature representations that capture both spectral and temporal characteristics of the signal.

#### Waveforms and Spectrograms

As we have already mentioned, the waveform by itself does not convey information about the frequency content of the audio signal, a critical aspect of speech comprehension. We use the *Discrete Fourier Transform (DFT)* (implemented using *Fast Fourier Transform (FFT)* algorithms) to obtain the frequency spectrum at a specific point in time. DFT transforms a time-domain signal into its constituent frequencies (Heideman et al., 1984).

The frequency spectrum provides a snapshot of the audio signal's frequency content at a particular moment, showing the presence and intensity of different frequency components. However, speech signals change over time. We use the

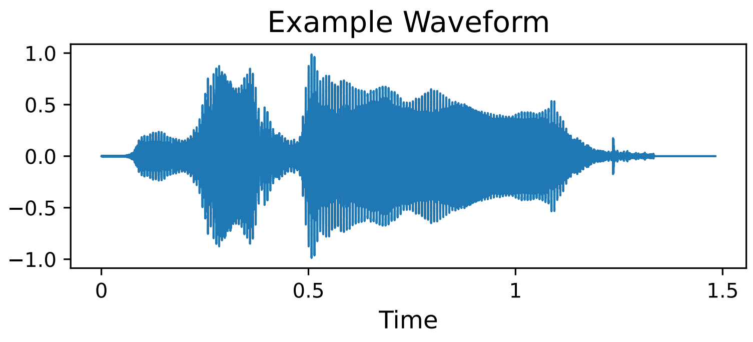


**Figure 1.1** Example of a waveform plot. Image created by the authors using `librosa` Python library.

*Short-Time Fourier Transform (STFT)* to capture these temporal variations (Allen; Rabiner, 1977).

The STFT uses the Fourier Transform to allow us to analyze the frequency content of signals over time. When computing STFT, we first divide the audio signal into short, overlapping segments (windows), and then we compute the DFT for each segment. As a result, we get a time-frequency representation of the signal, known as a spectrogram.

The resulting spectrogram is a two-dimensional array with time represented in one dimension and frequency in the other. When we plot these spectrograms, we usually display time on the horizontal axis and frequency on the vertical axis, using color to represent the magnitude of each frequency component according to a predefined scale.

### Mel-Frequency Spectrograms and Mel-Frequency Cepstral Coefficients (MFCCs)

While spectrograms are useful for visualizing and analyzing speech signals, they may still not be the best option for ASR processing due to their linear spacing of frequency bins, which does not align with human auditory perception. Mel-frequency spectrograms, where the frequency axis is transformed to match the Mel scale (Stevens et al., 1937; Stevens; Volkmann, 1940; Siegel, 1965), are often used to address these issues. The Mel scale was developed to better capture properties of human hearing, where the same perceived difference between pitches corresponds to a larger difference in frequencies when listening to higher frequencies than when listening to lower frequencies.

The Mel scale is defined by the following approximate formula:

$$\text{Mel}(f) = 2595 \cdot \log_{10}\left(1 + \frac{f}{700}\right)$$

where $f$ is the frequency in Hertz. The mel-frequency spectrogram is obtained by applying a filter bank of triangular filters spaced according to the Mel scale to the power spectrum of the STFT. This results in a spectrogram with frequency bins distributed more densely in the lower frequencies, mimicking human auditory perception. Figure 1.2 shows a log-mel spectrogram, where the frequency values are processed by taking their logarithm.

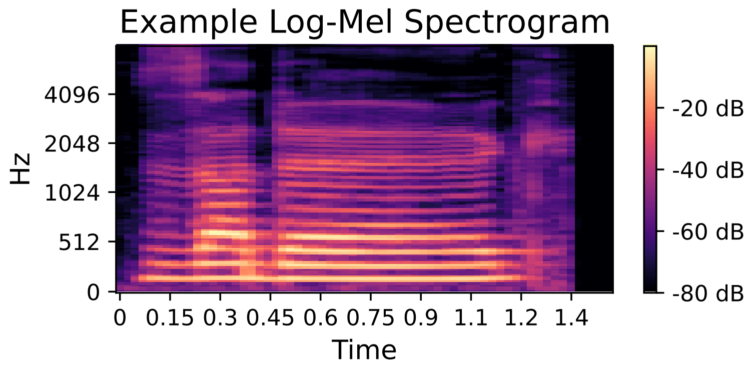


**Figure 1.2** Example of a log-mel spectrogram plot. Image created by the authors using `librosa` Python library.

To further reduce the dimensionality and extract compact features from the mel-frequency spectrogram, we compute *Mel-Frequency Cepstral Coefficients (MFCCs)*. MFCCs are derived by taking the logarithm of the Mel-frequency filter bank energies and then applying the *Discrete Cosine Transform (DCT)* (Xu et al., 2004; Hossan et al., 2010).

MFCCs are widely used as input features for ASR systems due to their effectiveness in capturing relevant acoustic information while reducing dimensionality. According to Davis and Mermelstein (1980), in some cases, as little as six MFCCs can be sufficient to capture most of the relevant information.

## 1.2 Language Models (LMs)

While the primary focus of ASR systems is on accurately transcribing spoken language into text, *Language Models (LMs)* are crucial in enhancing transcription accuracy by providing context and predicting the likelihood of word sequences. Language models are used in conjunction with acoustic models to improve overall system performance (Drugman et al., 2016; Hono et al., 2024). We will give a quite thorough, intuitive introduction to these models because it will be beneficial for the later description of machine learning methods.

Language modeling is usually *probabilistic*, i.e., the model assigns probabilities to different possibilities. We will describe this on an example of a *word n-gram language model* as it is probably the simplest class of language models.

Imagine we have a sequence of words, and we would like to predict (model) what the next word is. That will, of course, depend on the *context*. There is the language context available to the LM (the preceding sequence of words) and other context unavailable to the LM (such as the surroundings of the speakers) (Jurafsky; Martin, 2024). At the same time, there are multiple words that can follow the sequence, which is why we model the probability distribution. If we look at this from another point of view, we are modeling a function that takes the context and a possible next word as inputs and returns the probability that a given word follows the context.

We want the model to reflect the real world. Therefore, we collect a *corpus* (a collection) of texts that we believe represent the real-world usage of language (S. Young et al., 1999). From there, we obtain the *vocabulary* of all words that appear in the texts. Then, we record all the counts of possible contexts of different lengths and the respective following words and estimate probabilities from these counts.

We could call these estimated probabilities *parameters* as they help us evaluate the function for different inputs. However, this becomes next to impossible for longer contexts as the number of necessary parameters grows exponentially. For a context of length seven and vocabulary of size $|V|$, there are already $|V|^7$ possible contexts. The probability distributions would also be very sharp as many such long contexts will be found only once in the corpora.

Consequently, $n$-gram LMs learn and estimate the probability of a word based on a previous context of a fixed size $n-1$ words. For example, a bigram model ($n=2$) considers the previous word, while a trigram model ($n=3$) considers the two preceding words (see Figure 1.3 for an illustration). Even then, the resulting probability distributions are often too sharp, and smoothing methods (S. F. Chen; Goodman, 1999) are used.

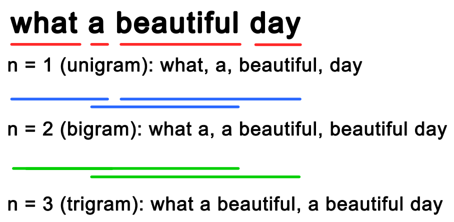

what a beautiful day
n = 1 (unigram): what, a, beautiful, day
n = 2 (bigram): what a, a beautiful, beautiful day
n = 3 (trigram): what a beautiful, a beautiful day

**Figure 1.3** Illustration of $n$-grams for $n \in \{1, 2, 3\}$. Image created by the authors.

The same $n$-gram LMs can then be used to estimate the probability of a word sequence $W = w_1, w_2, ..., w_T$ using the chain rule of probability:

$$P(W) = \prod_{t=1}^{T} P(w_t | w_{t-n+1}, ..., w_{t-1})$$

$n$-gram models have been widely used. They are both easy to implement and efficient in terms of computation. However, they are severely limited by their fixed context window and inability to capture long-range dependencies.

For further discussion of language models and their applications in ASR, refer to foundational texts such as books by Jelinek (1998) or Manning and Schutze (1999).

## 1.3 Machine Learning

We will introduce machine learning on the basis of $n$-gram LMs. The $n$-gram model used its parameters to remember the values of the function for different multidimensional inputs (the context of length $|C|$ corresponds to $C$ individual inputs). We want the model to work with the parameters more efficiently and truly model the function, not just remember its values for different inputs.

This leads us to the definition of a general machine-learning model. The model approximates a function that, for some (usually multidimensional) input $x$, returns a value $y$ using some set of parameters $\theta$ (Goodfellow et al., 2016). Different model architectures then provide different tools for how the inputs and the parameters can be combined to obtain the prediction.

For example, let us have a simple machine learning model that receives a black-and-white photo on the input and decides whether it was taken during the day or during the night. The input $x$ are all the pixel values of the photo (the lighter the pixel, the higher the numerical value). We allow the model only to sum the inputs, and it has a single parameter $\theta$. This parameter represents a threshold under which the model predicts “night” and over which it predicts “day”.

In the *training* stage, we set the threshold so that it maximizes the model accuracy on some *training set* of photos for which we know the correct prediction. This brings in an important aspect and requirement. The model should truly approximate the function, not just remember its values at different points. This way, the model learns to *generalize* and produce correct outputs also for unseen inputs (Bishop; Nasrabadi, 2006). Now, if we show the model a new photo, we have a good chance that it correctly decides when it was taken, even though it has never seen it. When we use the model after the training to make predictions, we call it *inference*.

In order to make sure our model generalizes well, we need to test it on data it has not seen during the training. The available dataset is, therefore, usually split into three parts – *training*, *validation* (or *development*), and *test* sets (Goodfellow et al., 2016). We use the train set to change the model parameters $\theta$ using a specified training algorithm (we will explore some specifics in Subsection 1.7). Then, when the performance of the model on the train set is satisfying, we test the model performance on data it has not seen – on the validation set.

This can show us whether the model is *overfitting*, i.e., remembering too much of the training data and not generalizing well (Hastie et al., 2001). Then, we can change the *hyperparameters* (any parameters not changed during training, e.g., the number of trainable parameters $|\theta|$) and train the model again. We cannot use the validation set for the final model evaluation because we have been changing the model to work on these data better. To make the final measurement of how well the model generalizes, we evaluate it on the test set.

## 1.3.1 Modes of Training

During the training, the model should set its parameters $\theta$ to optimal values so that it approximates the function as effectively as possible. The parameters are changed using a training algorithm. The choice of the training algorithm depends, among others, on the task at hand, the data available, and the model architecture.

The machine learning methods are usually divided into *unsupervised* and *supervised* learning (some also put *reinforcement* learning into a separate category) (Russell; Norvig, 2021). We focus on supervised learning and its variants, as these are the only modes employed in this work.

### Supervised Learning

In supervised learning, we have a dataset with inputs and corresponding outputs (so-called *gold labels* or *ground truth*) available (Murphy, 2012). During training, we give the model one *sample* from the dataset and let it make a prediction. Then, we compare the prediction with the gold label and use a training algorithm to change the model parameters $\theta$ so that next time, its prediction for the same input is closer to the gold label.

Labeled datasets are often hard to obtain. Therefore, an approach of *pre-training* and later *fine-tuning* the model is very frequent (Devlin et al., 2019). During the pre-training, we train the model on a bigger dataset we have available. This dataset may deal with slightly different data or even with a different task. The goal of the pre-training is for the model to learn to extract useful features from the input and create effective inner representations (we will talk about them in Subsection 1.5.2). Then, we fine-tune the model on a smaller dataset with examples very similar to what we will test the model on. Extreme examples of this are *few-shot* and *zero-shot* methods, where we fine-tune the model only on a few or zero examples of the final task.

To further reduce the need for manually labeled datasets, *weakly-supervised* and *self-supervised* approaches were proposed.

**Weakly-supervised Learning** This method's main idea is that we do not need the labels to be perfect for the model to be able to learn from it (Zhou, 2018). We obtain the dataset, for example, by scraping the relevant input-output pairs from the internet or by using a preliminary model to label previously unlabeled data. As we cannot be sure the labels are correct, we usually call them *pseudo-labels*.

**Self-supervised Learning** When trained in a self-supervised manner, the model is trained on huge datasets of unlabeled data by creating surrogate tasks, such as predicting parts of the input from other parts (X. Liu et al., 2023). This approach leverages the data's inherent structure to learn inner representations without requiring manually assigned gold labels. This technique proved very effective in many fields, as large amounts of unlabeled data in different forms (textual, audio, images, ...) can be used to pre-train models that can later be fine-tuned with smaller labeled datasets, significantly improving performance and generalization.

## 1.4 Neural Networks

*Neural Networks (NNs)* as an architecture were widely theoretically studied already in the 20th century (McCulloch; Pitts, 1990; Hopfield, 1982). They got their name because their inner workings are inspired by a simplified understanding of how neurons interact in the human brain when processing information. The practical study of these structures was limited mainly by the lack of computational power. These barriers were removed, and NNs were widely adapted for a wide range of tasks.

### 1.4.1 Feed-Forward Networks

*Perceptron* (Rosenblatt, 1958) was one of the earliest forms of neural networks. Later, it was extended to the multi-layer perceptron. This design serves as a basis for today's feed-forward networks. We describe it on an example with a single layer.

To feed the idea of similarity with the brain, we represent it as a graph, as shown in Figure 1.4. Each input value gets a node, so-called *neuron*. Each of

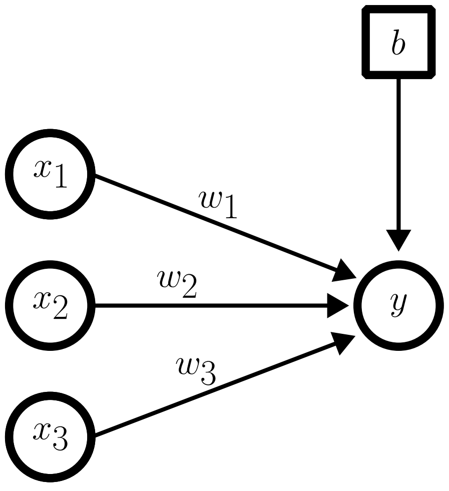


**Figure 1.4** Single-layer feed-forward neural network. Image inspired by Figure 1.2 of the book by Du and Swamy (2013); adapted by the authors.

these input neurons is connected to a single output neuron using an edge. A *weight* on each edge measures how much a given input neuron should influence the final result. Each input neuron is multiplied by its corresponding weight, then we sum these values, add a *bias*, and finally pass this through an *activation function*, receiving the value of the output neuron. Formally:

$$y = a(\sum_i x_i w_i + b)$$

where $y$ is the value of the output neuron, $x_i$ are values of input neurons, $w_i$ are their corresponding weights, $b$ is bias and $a$ is the activation function.

Bias is also a trainable parameter, and it allows the model to modify its predictions independently of the inputs. The (nonlinear) activation function allows the model to produce more than just linear combinations of the input values, drastically widening what functions can be modeled.

We call the not interconnected (visualized in one column) neurons a *layer*. The presented example has only the input layer and the output layer. However, we could add another *hidden* layer of neurons in between, see Figure 1.5. In the most basic form, this layer is *fully-connected*, meaning each neuron is connected (using an edge with a weight) with each neuron from the previous layer. The value of each neuron is computed exactly the same way as before; each has a bias and an activation function. The more layers the model has, the *deeper* we call it.

## 1.4.2 Deep Neural Networks (DNNs)

Neural networks with multiple hidden layers are considered *deep*. Unlike traditional approaches that rely on handcrafted features, thanks to the multiple layers, DNNs can learn hierarchical representations of the input data directly. This capability allows DNNs to capture more complex patterns in the inputs, such as speech, leading to more accurate output (transcriptions). DNNs have significantly improved the performance of ASR systems (Hinton et al., 2012).

DNNs proved to be powerful; however, they have several limitations. The *vanishing gradient problem* is one of them (Hochreiter, 1998). Multiple methods

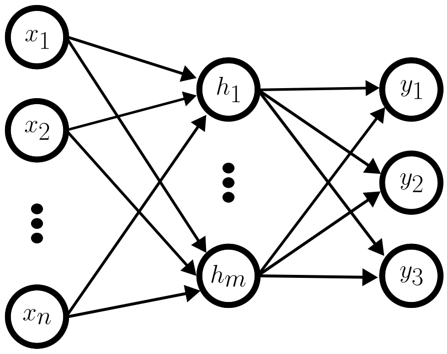


**Figure 1.5** Multi-layer feed-forward neural network. Image inspired by Figure 1.5 of the book by Du and Swamy (2013); adapted by the authors.

were proposed to fight it (Hochreiter; Schmidhuber, 1997; K. He et al., 2016; Basodi et al., 2020; Roodschild et al., 2020).

Another problem is that the basic feed-forward architectures with multiple fully connected layers can accept only a fixed-sized input and process each part of the input with separately trained neurons. The audio recordings (and many other input types) have variable lengths, and we would also like to capture the idea that the basic features are extracted in the same manner in different parts of one input (Goodfellow et al., 2016).

### Recurrent Neural Networks (RNNs)

RNNs are designed to work with sequential data by iteratively processing small successive parts by a specific neural layer or set of layers (usually called a *cell*) while maintaining a hidden state that retains information from previous steps. In each timestep, the cell consumes one part of the input and the hidden state from the previous timestep (Jurafsky; Martin, 2024). This method allows for processing variable length input using a single model to handle different segments. Additionally, it reduces the number of necessary parameters. In ASR, the input sequence is an audio recording, and this approach ensures that the same word is transcribed consistently in various parts of the recording.

As the single cell is "unrolled" over the sequence, the resulting network is very deep. Unless explicitly targeted, it suffers severely from the vanishing gradient problem. *Long Short-Term Memory (LSTM)* (Hochreiter; Schmidhuber, 1997) networks address this issue by introducing memory cells and gating mechanisms that regulate the flow of information. The gates are controlled by trained layers. LSTMs can capture long-range dependencies in speech, leading to significant improvements in ASR performance (Graves et al., 2013). The computationally less intensive GRU (Cho et al., 2014) and other variants of memory cells were later introduced. However, they are not generally superior to the LSTM cell (Chung et al., 2014).

LSTMs (and RNNs in general) are often used *bidirectionally.* When consuming the input with an RNN from one side to another, the context propagates only in this direction. In other words, the hidden state produced after consuming the last word depends on the first word but not vice versa. For these reasons, the input is often consumed in one direction, then in the other, and the resulting hidden states are concatenated (Graves; Schmidhuber, 2005). See illustration in Figure 1.6.

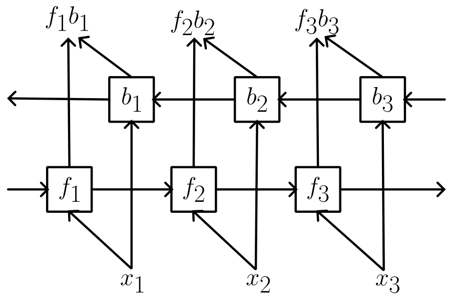


**Figure 1.6** Bidirectional RNN. Forward $f_i$ and backward $b_i$ representations are concatenated to create the final representation. Image inspired by Figure 10.11 of the Deep Learning book (Goodfellow et al., 2016); adapted by the authors.

#### Convolutional Neural Networks (CNNs)

Another type of architectures that target the idea that different parts of the input should be processed in the same manner are *Convolutional Neural Networks (CNNs).* CNNs use *kernels* to learn local patterns in the input data. Kernels are small layers that process parts (windows) of the input. They are moved (usually with an overlay) over the input, producing a more compact representation. By stacking multiple convolutional layers, CNNs can learn increasingly abstract representations of the input. CNNs also found their application in ASR, particularly for extracting features from spectrograms (Abdel-Hamid et al., 2014; Gulati et al., 2020; Baevski et al., 2020).

## 1.5 Representations

In this section, we explore how the text and other inputs are converted into representations suitable for neural networks. All the input needs to be numeric. To convert text into numeric representations, we first *tokenize* it. Then, we further process the representations into dense vectors called *embeddings* that encode the semantic meaning efficiently and simplify further modeling.

### 1.5.1 Tokenization

As language is comprised of smaller units, we say that the goal of language models is to model sequences of *tokens.* By token, we refer to a unit that constitutes the sequence, such as words, characters, or so-called *subwords.* We call the set of tokens available a *vocabulary.*

Straightforward units in text are characters or words. Characters are too granular; we would need many of them to represent a sentence, and they bear little meaning by themselves. On the positive side, the vocabulary of characters is small and easy to represent (at least with the Latin script). Words, on the other hand, are too coarse. For many languages, there are hundreds of thousands of words in the vocabulary. Furthermore, there are usually some morphological processes present, and the meaning of a word depends on its parts.

For these reasons, words are usually split into parts (groups of characters), so-called *subwords* (Kocmi; Bojar, 2016; Wieting et al., 2016). One of the widely used methods of subword vocabulary creation is *Byte Pair Encoding (BPE)* (Sennrich et al., 2016). In BPE, we start with characters as initial tokens, and we set the required vocabulary size. Then, based on our corpus, we iteratively merge the pairs of tokens that appear next to each other the most often until the required vocabulary size is reached. We end up with a vocabulary of tokens (with different lengths) that are the most efficient in a sense (Gage, 1994).

### 1.5.2 Embeddings

We have talked about effective inner representations. The goal is to take the input and process it in a way that results in representations that are easy to deal with for the subsequent parts of the model. In other words, we need to project – "embed" – the input into some descriptive feature space.

The simplest way to represent tokens is *one-hot encoding*. We get a vector of the same size as vocabulary, fill it with 0 (zeros), and each word will have 1 (one) at one specific place of the vector. This representation is not efficient at all. First, the required vectors are enormous. Second, different words have completely different representations, no matter how similar or dissimilar the words are.

We want the vector representations to be continuous to allow for shorter vectors. Ideally, we also want the embeddings to capture the syntactic and semantic properties of the words so that similar words are represented similarly. Such embeddings enable models to understand relationships between words based on their contexts in large corpora.

The first famous approach to self-supervised embedding pre-training was Word2Wec (Mikolov et al., 2013); other soon followed (Pennington et al., 2014; Bojanowski et al., 2017). Word2Vec uses two main approaches to learn the embeddings: Continuous Bag of Words (CBOW) and Skip-gram. With CBOW, a word from a sequence is masked, and the model has to predict the best-fitting word based on the context. On the opposite, with Skip-gram, the model has to predict the context words from a target word. Both approaches aim to maximize the likelihood of word-context pairs in a large corpus, resulting in embeddings that capture word meanings effectively.

The idea of self-supervised training is crucial for efficient embeddings. It allows us to use previously unparalleled amounts of text for the model training. Similar approaches can be adapted for input modalities other than text. As such, pre-trained embeddings became a new standard for many tasks (Dosovitskiy et al., 2021), including ASR (Baevski et al., 2020). In ASR, we usually use segments of the spectrogram (or raw waveform) and embed these.

## 1.6 Advanced Neural Network Architectures

A large part of NLP deals with *sequence-to-sequence (seq2seq)* tasks (or problems that can be formulated that way). Generally, seq2seq tasks involve consuming some sequence as an input and producing another sequence on the output. ASR is a prime example.

In this section, we discuss the *encoder-decoder* framework targeted specifically at seq2seq tasks. Then, we talk about *attention* that improves the encoder-decoder performance and about the *Transformer* model that builds on the idea of attention.

### 1.6.1 Encoder-Decoder Framework and Attention

With RNNs, we have a tool to process variable-length sequences. Using the intermediate hidden states, we can get embeddings of each segment of the sequence and classify it. However, there is no straightforward method to handle seq2seq tasks where input and output sequences have different structures or lengths. The *encoder-decoder* architecture was proposed to address these tasks.

The encoder-decoder architecture was first proposed by Cho et al. (2014) and further popularized by Sutskever et al. (2014). As the name suggests, this architecture has two crucial parts – an *encoder* and a *decoder*. The encoder processes the input sequence and compresses it into a fixed-size context vector.

This context vector ideally captures all the relevant information of the input sequence. The decoder then usually generates the output sequence *autoregressively*. First, it consumes the context vector and a special *start of sequence token* (usually denoted `<SOS>`). Based on this, it generates the first output token and a new context vector. The new context vector should reflect that the first output token already expressed part of the information. In the next timestep, the decoder consumes the first output token it generated, the new context vector, and generates a second output vector. The decoding continues this way until the *end of sequence token* (`<EOS>`) is generated. By separating the encoding and decoding processes, the architecture allows for more flexibility and better handling of variable-length input and output sequences.

**Teacher Forcing** During the training, we could feed the input into the encoder, get the encoded context vector, pass it to the decoder, and let it generate the output autoregressively. However, for such a model, it is tough to learn, as at the start of the training, the first output of the decoder will be random. Consequently, all the following output tokens will probably be wrong, as they are based on the first output. To help the model learn, instead of training the decoder with its own intermediate outputs, we feed it with the gold labels. This allows it to train at each timestep properly.

**Scheduled Sampling** Teacher forcing can help the decoder, especially at the start of the training. However, it also has drawbacks – the decoder is always trained with all the preceding timesteps outputs correct. During inference, the decoder can make a mistake at the beginning of the sequence and then cannot recover from it, as it never saw such a mistake during the training. *Scheduled sampling* (Bengio et al., 2015) suggests mixing the teacher-forced gold labels with autoregressive decoding. At the start of the training, the learning is mostly teacher-forced, but a gradually increasing proportion of the gold labels is replaced by the current decoder outputs.

#### Attention Mechanism

Using a single context vector often led to a loss of information, especially for long sequences. There is a bottleneck issue – the encoder has to save all the information from the input into a fixed-size vector. The attention mechanism (Bahdanau et al., 2015) was introduced to address this limitation. Instead of relying on a single context vector, attention allows the model to focus dynamically on different parts of the input sequence during the generation of each output element.

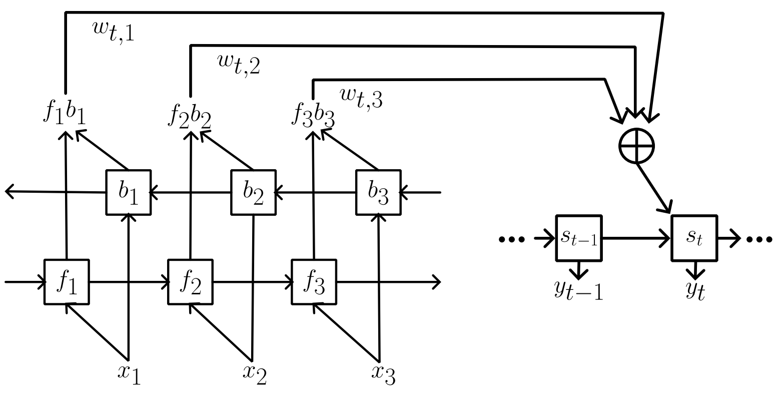


**Figure 1.7** BiRNN-based Encoder-decoder with attention in autoregressive generation timestep $t$. Image inspired by Figure 1 of the original work (Bahdanau et al., 2015); adapted by the authors.

In the attention mechanism, a set of weights is computed for each output timestep to determine the relevance of each input element, as illustrated in Figure 1.7. The weights are utilized to generate a weighted sum of the encoder's hidden states, which acts as the context vector for that particular output step. This way, the decoder can "attend" to different parts of the input sequence as needed.

The introduction of the attention mechanism resolved the bottleneck issue. However, another issue of RNNs is their speed. They work sequentially both during inference and during training. Due to the sequential nature, the training cannot be parallelized. The lack of parallelization is an issue, as it limits the use of large training datasets.

### 1.6.2 Transformer Architecture

Building on the success of the attention mechanism, the *Transformer* architecture was introduced (Vaswani et al., 2017). Different models based on this architecture achieved state-of-the-art results on many NLP tasks, including ASR, see Section 1.9 (Baevski et al., 2020; Radford et al., 2023). It found its use also in other fields, such as in computer vision (Dosovitskiy et al., 2021).

The Transformer architecture completely drops the recurrences that slow down RNNs. Instead, it also utilizes the attention in the encoder to compute the encodings. Because the encoder attends over its own input, this is called *self-attention*. The decoder is then conditioned using attention both on its own input (self-attention) and on the encoder output (*cross-attention*). Attention, in combination with teacher forcing, allows for highly parallelizable computations since the attention scores for all positions can be calculated simultaneously.

See Figure 1.8 for an illustration of the architecture. The encoder in the Transformer model consists of multiple identical layers, each featuring two main components: a *multi-head* self-attention mechanism and a position-wise fully connected feed-forward network. *Residual connections*, i.e., direct connections between two not directly neighboring layers, allow for better information retention.

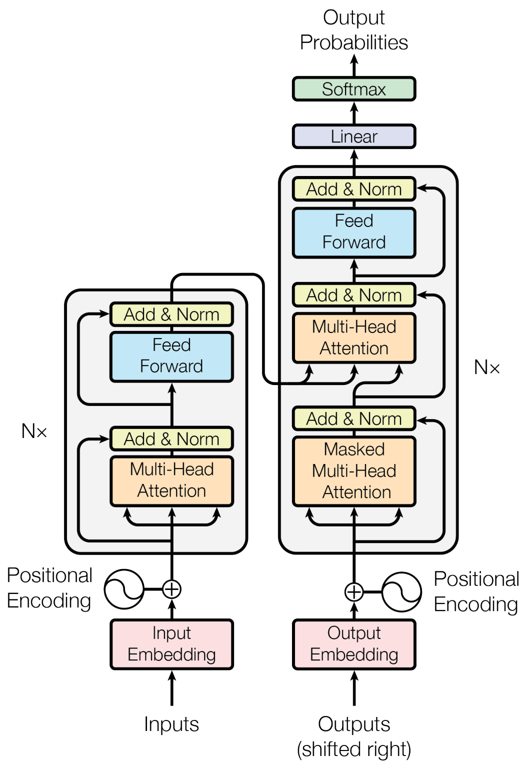


**Figure 1.8** The Transformer architecture. Note that both the encoder and the decoder can consist of multiple identical layers, as suggested by the N×. Figure 1 of the original work (Vaswani et al., 2017).

Multi-head attention computes multiple context vectors in parallel, concatenates them, and projects them into the resulting space. For this process to be beneficial, the context vectors should differ from each other. This is achieved through linear projection of the parameters into different spaces before computing the attention scores. Multi-head attention allows the model to retain more context information than just with a single head.

The decoder also consists of multiple identical layers, with an additional sublayer to attend to the encoder's output. It utilizes: a multi-head self-attention mechanism, a multi-head attention mechanism to attend to the encoder's output, and a position-wise fully connected feed-forward network. Similar to the encoder, residual connections are used.

Unlike RNNs, the Transformer architecture does not inherently capture the order of the sequence. The input embeddings are combined with positional encodings to introduce the tokens' positional information in the sequence. There are multiple ways to compute positional encodings; they can be derived from trigonometric functions but can also be learned (Gehring et al., 2017).

The Transformer networks can process only fixed-sized inputs because they lack the recurrences. However, the parallelization and accurate context-aware representations seem to outweigh this negative. Special *padding tokens* (`<PAD>`) are utilized to account for variable-length sequences. These tokens are not considered for attention or loss computation.

The Transformer encoder and decoder can also be used separately. BERT (Bidirectional Encoder Representations from Transformers) (Devlin et al., 2019) represents the encoder-only, while GPT (Generative Pre-trained Transformer) (Radford; Narasimhan, 2018; Brown et al., 2020) is decoder-only.

## 1.7 Neural Network Training

Now, we finally get to how the machine learning models, specifically neural networks, are trained, i.e., how their parameters are updated to produce better results on the training data. We need to discuss how to measure the size of errors made by our models and how to update the parameters to reduce these errors.

### 1.7.1 Loss Function

To measure how far off the model is from the correct output, we define a *loss function* (Goodfellow et al., 2016). Note that it is important not to mistake it with the function the model is trying to approximate that describes the data. Many loss functions have been defined; let us mention the ones most relevant to our work.

#### Mean Squared Error

Probably the simplest loss function to start with is the *mean squared error (MSE)* (Hastie et al., 2001). It is utilized for models that perform *regression*, meaning they predict a single continuous numeric value. The mean squared error of the model simply measures the average squared distance between the model prediction and the gold label:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(f(x_i;\theta) - y_i)^2$$

where $f(x_i;\theta)$ denotes the output of the model for the input $x_i$, $y_i$ is the corresponding gold label and we compute the average over $n$ samples.

#### Cross-Entropy Loss

Cross entropy loss is useful when the task of the model is *classification*, i.e., prediction of discrete class labels (Jurafsky; Martin, 2024). For example, when performing next-word prediction, the model predicts a probability distribution over possible next words. To assess the model's error, we need to compute how far the predicted distribution is from the gold label distribution. Cross entropy loss serves exactly this purpose.

#### CTC Loss

We describe the CTC loss (Graves et al., 2006) on the ASR example. Let us assume our model has audio as an input, and it creates embeddings of fixed-length segments of the audio. Then, we would like to classify the embedding of each audio segment into a character based on what was pronounced. However, we do not know the *alignment* – sometimes a character can span over multiple segments; sometimes, there was silence in the given segment.

CTC loss introduces a new *blank* output character $\epsilon$ (Hannun, 2017), follow along in Figure 1.9. The $\epsilon$ character denotes an empty segment. It also serves as a boundary. CTC loss first merges all the repeating occurrences of one character unless they are separated by $\epsilon$ (lines 1 and 2). Then, it removes the $\epsilon$ from the sequence (lines 2 and 3) to receive the final reduced sequence (line 4).

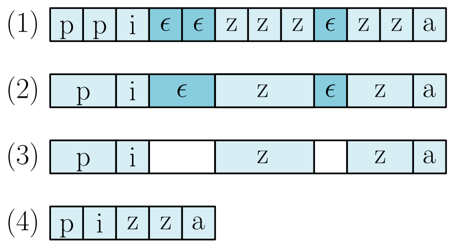


**Figure 1.9** Illustration of the CTC loss sequence reduction. Image inspired by Sequence Modeling With CTC (Hannun, 2017); adapted by the authors.

The model then predicts a distribution over characters (including $\epsilon$) for each segment embedding. We can compute probabilities of different sequences, reduce each sequence, and sum the original probabilities to obtain a probability of each resulting sequence. This way, we can compute the probability of each resulting sequence without an explicit alignment. CTC loss then uses a dynamic programming algorithm (Bellman; Dreyfus, 2010) to efficiently compute how each of the predicted segment probability distributions contributes to the results and measure their error.

### 1.7.2 Stochastic Gradient Descent (SGD)

Now that we have defined how the model's error is measured, we need to determine how to improve its prediction. *Stochastic gradient descent* (Robbins; Monro, 1951) serves exactly this purpose. We can evaluate the loss function at one point using the current model parameters $\theta$ and one input sample $x$. Now, we need to find how to change the parameters $\theta$ so that we minimize the loss function (the error of our model).

There is no simple way to find parameters $\theta$ that minimize the loss function directly. Thus, we will proceed iteratively. We compute the *gradient* (vector of derivatives) of the loss function with respect to all the parameters $\theta$ to obtain the direction in which the loss function changes the fastest. Then, we change the parameters to move in the direction that gets us closer to the minimum of the function.

Estimating the gradient based on one input sample $x$ produces noisy results, as this one sample can hardly accurately represent the whole dataset. On the other hand, estimating it on the whole dataset would be computationally infeasible. As a compromise, a *batch* of samples (more than one and less than the entire dataset) is usually used to estimate the gradient.

### 1.7.3 Backpropagation

Backpropagation (Rumelhart et al., 1986) is used to obtain the exact partial derivatives with respect to each model parameter. In a neural network, each neuron's value in a subsequent layer is computed as a linear combination of the previous layer's neuron values, followed by an activation function. Therefore, the value of an output neuron can be seen as a nested composite function of the inputs and the weights (parameters) across multiple layers. When evaluating the loss function based on the output neuron's value, we can use the chain rule to calculate the derivatives and propagate the error back through the network.

## 1.8 Traditional ASR Methods

Traditionally, ASR was dominated by methods based on *Hidden Markov Models (HMMs).* An HMM consists of a set of states, each associated with a probability distribution, and transitions between these states, again with their respective probabilities (Rabiner, 1989). The states themselves are not directly observable (hence "hidden"), but each state generates an observable output. In the case of ASR, the hidden states can be represented by linguistic units (subwords, phonemes, graphemes), and the observable output from each state corresponds to the acoustic features extracted from the audio signal. To decode a sequence of observed features (i.e., to determine the most likely sequence of hidden states), we use algorithms such as the Viterbi algorithm (Viterbi, 1967).

HMMs were later combined with *Gaussian Mixture Models (GMMs)* (Gauvain; Lee, 1994) to improve the modeling of acoustic features. GMMs are also probabilistic models, but they make an assumption that the data is generated from a mixture of several Gaussian distributions. During training, the parameters of the GMMs and the HMM transition probabilities are estimated using the *Expectation–Maximization (EM) algorithm*, which iteratively maximizes the likelihood of the observed data. To be specific, the *Baum–Welch algorithm* (Baum et al., 1970) is usually used when dealing with HMMs.

HMM-GMM models represented state-of-the-art ASR for a long time. However, they rely heavily on handcrafted features and linear assumptions, which often do not adequately capture the complexities of speech. Neural network methods were initially used to estimate HMM model parameters (Hinton et al., 2012) and later progressed to *end-to-end (E2E)* approaches that now dominate ASR. With the E2E approach, the audio (possibly with some deterministic preprocessing) is input into the neural network on one end, and transcription is output on the other.

## 1.9 Modern ASR Systems

The neural approaches presented in Sections 1.4 and 1.6 were successfully used for ASR. Chorowski et al. (2014) utilized BiRNN encoder-decoder with attention, Graves et al. (2013) explored the use of BiRNN encoder with CTC loss. Most of today's methods are Transformer-based.

In this section, we will explore two of the most prominent state-of-the-art open-source Transformer-based ASR architectures: Wav2Vec 2.0 (we shall call it only Wav2Vec2 for simplicity) (Baevski et al., 2020) and Whisper (Radford et al., 2023). We have selected these architectures due to their exceptional performance, availability of pre-trained models, and support for multiple languages, including Czech. They are also interesting in their differences – while Wav2Vec2 utilizes large amounts of unlabeled data through self-supervised learning, Whisper relies on weak supervision. Both of the architectures also reduce the dependency on large labeled datasets, making them accessible for a broader range of applications.

### 1.9.1 Wav2Vec 2.0

Wav2Vec2, introduced by Baevski et al. (2020), utilizes self-supervised learning to create robust speech representations (embeddings) of raw audio data. The authors argue that this approach may pose great hope for low-resource languages, where a very limited amount of labeled data is available for ASR training. We will provide a short overview of the model architecture and the training process.

#### Architecture

The Wav2Vec2 model architecture has three main parts that we further describe: feature extraction, context network, and quantization. An illustration of the architecture can be seen in Figure 1.10.

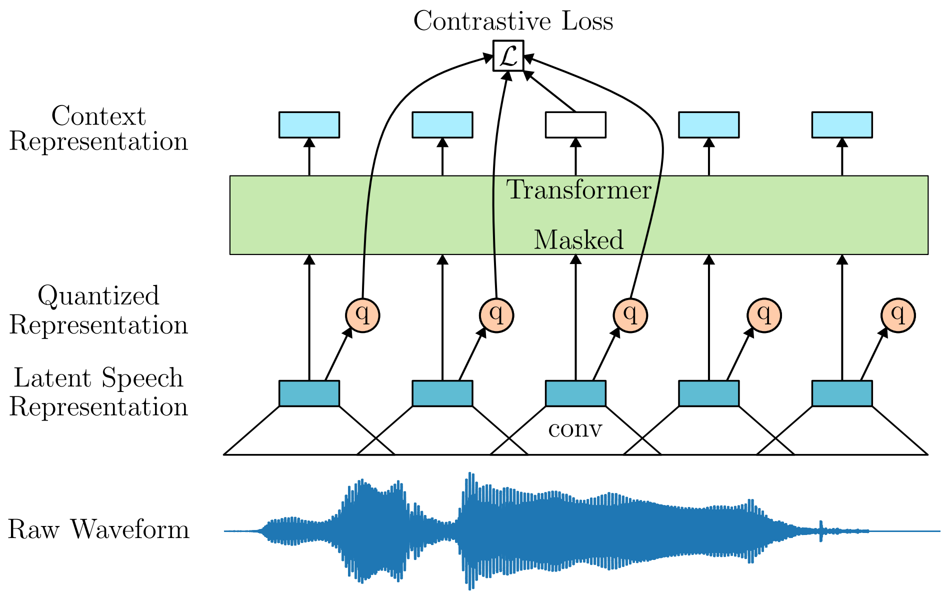


**Figure 1.10** Wav2Vec2 model architecture. The yellow tokens represent special prompt tokens, and the blue tokens denote standard predicted text tokens. Image inspired by Figure 1 of the original work (Baevski et al., 2020); adapted by the authors.

**Feature Extraction** The feature extraction stage of Wav2Vec2 uses a multi-layer convolutional neural network to encode the raw audio input (the waveform) into latent speech representations. These representations should capture various aspects of the speech signal, such as phonetic and prosodic features.

**Context Network** Following feature extraction, the latent representations are fed into a Transformer network, which builds contextualized representations. This process involves masking spans of the latent speech representations, similar to the Word2Vec. The Transformer network's role is to model the long-range dependencies in the speech data, enhancing the robustness of the representations.

**Quantization** Predicting continuous speech representations would be complicated. Therefore, Wav2Vec2 employs a quantization mechanism to discretize the continuous latent representations into a finite set of learned speech units. This step is crucial for the contrastive loss used during the model's training.

#### Training

The main goal of Wav2Vec2 is efficient audio encoding, i.e., creating robust embeddings. This is done using the self-supervised pre-training (with unlabeled audio). To make the model usable for ASR itself, it needs to be fine-tuned using labeled data.

**Pre-Training with Contrastive Loss** The primary training objective of Wav2Vec2 is a contrastive task. Simply put, part of the created audio representation is masked, and the model needs to decide which quantized unit from those offered originally was at the masked place. This self-supervised learning approach enables the model to learn meaningful speech representations from vast amounts of unlabeled audio data.

**Fine-Tuning with CTC Loss** After pre-training, Wav2Vec2 is fine-tuned using labeled speech data. A randomly initialized linear layer is added on top of the network. This layer is then trained using CTC loss to classify the embedded parts of audio into vocabulary tokens.

### 1.9.2 Whisper

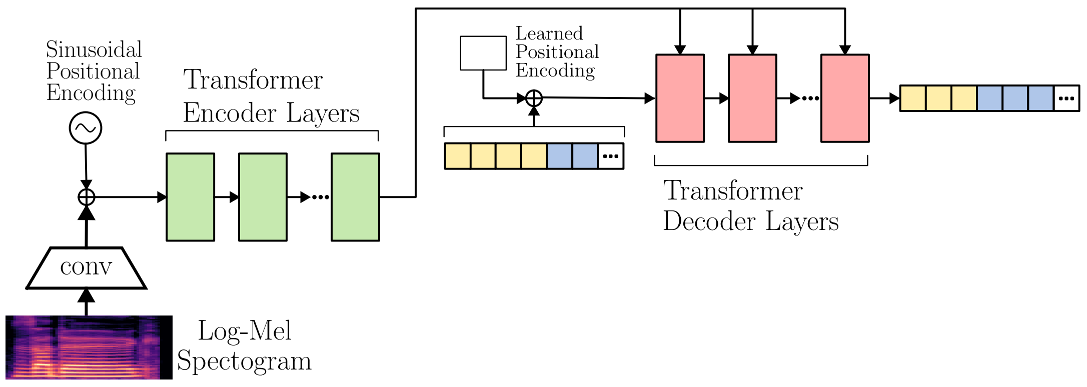
*Whisper* was introduced by Radford et al. (2023). They use an off-the-shelf Transformer-based architecture, but the approach to training it is very much new in the ASR field. Whisper has demonstrated remarkable performance in various benchmarks, achieving state-of-the-art results in many speech recognition tasks. It is strong, especially in the zero-shot settings, as it can be used without fine-tuning.

#### Architecture

The model architecture is presented in Figure 1.11. Whisper processes the input audio (in the form of a Log-Mel spectrogram) with a block of Convolutional layers and then adds sinusoidal positional encodings. This preprocessed input is then fed into a Transformer encoder connected to a Transformer decoder with learned positional encodings. The importance then lies in the large-scale, weakly supervised training stage.

#### Training

The model is trained using weak supervision on a huge dataset. Furthermore, the model is designed to handle multiple tasks directly instead of relying on other supporting components.



Sinusoidal Positional Encoding
Transformer Encoder Layers
Learned Positional Encoding
conv
Log-Mel Spectogram
Transformer Decoder Layers

**Figure 1.11** Whisper model architecture. The yellow tokens represent special prompt tokens, and the blue tokens denote standard predicted text tokens. Image inspired by Figure 1 of the original work (Radford et al., 2023); adapted by the authors.

**Weak Supervision** Weak supervision in Whisper involves training the model on 680k hours of speech data in different languages. The authors state that the previous largest dataset contained 30k hours of audio. Whisper dataset creation started with well-known ASR datasets such as LibriSpeech (Panayotov et al., 2015). Then, the researchers added a vast amount of audio scraped from the internet that had some transcription available. This is classified as weak supervision because the labels were not manually checked. The dataset was filtered using heuristics in order to remove the noisiest data pairs (for example, automatically generated transcriptions).

**Multitask Setting** Whisper's architecture is designed to support multiple languages and resolve multiple tasks. Of the 680k hours of training data, 117k hours cover 96 languages other than English, and another 125k hours contain audio in different languages with transcriptions translated into English. Where the audio language was unknown, the language was pseudo-labeled using a model trained on the VoxLingua107 dataset (C. Wang et al., 2021).

As such, the model is trained in a multitask setting and is able to perform the following:

- voice activity detection (distinguishing parts of the audio with silence),
- language identification,
- audio transcription (i.e., ASR per se),
- direct audio translation from any language into English text,
- text-aligned transcription (interleaving the transcript with time information).

The task requirements are prepended as special tokens so that the Transformer decoder can attend to them. Furthermore, some previous text tokens can be prepended to the input so that the decoder can also take into account long-range dependencies.

# 2 Slot Filling in Text-Based Systems

Slot filling in NLU involves identifying and extracting specific pieces of information from a given text input. This task is crucial for various applications, including virtual assistants, chatbots, and information retrieval systems. Over the years, slot-filling techniques have evolved significantly, transitioning from manually crafted rule-based approaches to sophisticated machine learning-based methods.

We will start this chapter by introducing the most common metric for slot filling in Section 2.1. Then, we will discuss different slot-filling approaches. We have tried to split them into categories, mainly based on how they form their outputs. Many specific methods are hard to classify as they often lie on a border of the categories; many proposed methods can't be classified into any of them.

Most of today's research on slot filling uses RNN-based or Transformer-based encoders to create contextualized embeddings of the input utterance, possibly enriched with other contexts like previous utterances or dialogue states. Then, it either classifies the whole sequence (we cover in Subsection 2.2.2) or individual tokens (Subsection 2.3.2). Another possibility is to follow the encoder by a decoder and take a generative approach (Subsection 2.3.3). We will not dive into many specific architectures and improvements in this chapter; there are multiple surveys and reviews available that cover them sufficiently (Zailan et al., 2023; Louvan; Magnini, 2020).

Unrelated to the specific task formalization and model architecture, slot filling can be performed jointly with intent detection. As often specific types of slots are necessary to fulfill a specific intent (and vice versa), information about one helps to predict the other (B. Liu; Lane, 2016; X. Zhang; H. Wang, 2016; Chao; Lane, 2019). Weld et al. (2023) provide a survey of joint NLU models.

## 2.1 Evaluation Metrics

In slot filling, the objective is to extract slot-value pairs from the input text. The systems are usually evaluated based on the comparison of each of the predicted pairs against some ground truth. Common metrics include:

- **Precision**: The ratio of correctly predicted slots (true positives) to the total predicted slots (true positives + false positives). High precision indicates that the system's predictions are accurate.
- **Recall**: The ratio of correctly predicted slots (true positives) to the total actual slots (true positives + false negatives). High recall signifies that the system successfully identifies most of the true slots.
- **F1-Score**: The harmonic mean of precision and recall. This metric provides a balanced measure that considers both precision and recall:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

In most of the benchmarks, F1-Score on slot-value pairs is used as the main evaluation metric.

As there are usually multiple slot types, there are two ways to compute the F1-Score. *Micro-averaged* F1 first collects the partial information (true and false positives, false negatives) across all the classes, sums the partial values and only then computes the F1-Score. *Macro-averaged* F1 computes the F1-Scores separately for each class and only then averages them. Therefore, it assigns equal weight to all the classes, no matter how many samples represented them.

The metric can be calculated on the token level instead of the slot-value pair level (each value can consist of more tokens) for a more granular evaluation. Conversly, slot filling can also be assessed in a more integrated manner. If the slot-filling module is developed alongside a complete system, the ratio of successfully completed dialogues can be measured as a performance of the whole system.

## 2.2 Slot Filling as Direct Classification

In the context of slot filling, direct classification approaches aim to identify and extract slot-value pairs from the input text without directly considering the sequential nature of the data. These methods usually return a set of slot-value pairs based on the input sequence $X$. The primary techniques include string matching and rule-based systems, which leverage predefined patterns or databases. More advanced classification approaches use machine learning algorithms to assign slot labels based on learned features. The main advantage of direct classification is its simplicity and efficiency, particularly for structured and predictable inputs.

### 2.2.1 String Matching and Rule-Based

Early slot-filling systems relied heavily on string matching and rule-based techniques. These methods, while effective for structured and predictable inputs, require extensive manual effort.

In some applications, simple string matching is quite sufficient. Such a system looks for exact matches of predefined keywords or phrases in the input text. This can be the case, for example, when we need to fill only one slot that deals with the names of cities. Fuzzy matching techniques, such as those based on Levenshtein distance (Levenshtein, 1965), may be introduced to handle minor variations and errors in the input text. For example, matching “San Francisco” with “San Fransisco” or “San-Francisco.”

Rule-based systems then use handcrafted rules, often implemented as regular expressions, to identify and extract slots from the text. For instance, a rule-based system for booking travel might use regular expressions to identify departure and destination locations using phrases (templates) like `from <value>` and `to <value>`. Although rule-based approaches can be very precise, they are labor-intensive to develop and maintain. The number of such rules may grow very fast, especially for inflectional languages with free word order like Czech. An initial version of the system called Alex by Dusek et al. (2014) may serve as an example.

### 2.2.2 Statistical Classification

With advancements in machine learning, slot-filling approaches evolved to use statistical classification models. As with all statistical models, a nontrivial amount of training data is necessary. A possible approach to slot filling by classifying the whole utterance is to make multiple predictions – one for each slot type. This is usually done by embedding the utterance and then adding a prediction head for each slot type. Each of these predictions classifies whether the user mentioned a specific value for a given slot type. Values "none" and "don't care" are usually added for each slot type.

Issues with such an approach are apparent. First, we need to make a prediction for each slot type separately, which is computationally expensive. Second, this approach cannot detect multiple values for one slot type in one utterance. On the other hand, it is possible to easily add a new slot type by adding a new classification head (and training it), possibly without the need for retraining the whole model.

Chao and Lane (2019) use a similar approach in their BERT-DST model. It processes the dialogue context and utterance with BERT to create contextualized embeddings. Then, a classification model takes the embedding token `[CLS]` provided by BERT and predicts if slot value is present for each slot type. This is then followed by a span prediction module. This module works already with token embeddings and predicts a span where the slot value should be found.

## 2.3 Slot Filling as Sequence Labeling

Sequence labeling formalizes slot filling as a task where the input text is processed token by token, and each token is assigned a label that denotes its role within the slot structure. Formally, for a sequence of tokens $X = x_1, x_2, \dots, x_n$, the goal is to predict a sequence of corresponding categorical labels $Y = y_1, y_2, \dots, y_n$. This approach directly captures the sequential dependencies between tokens, allowing for a more nuanced and accurate extraction of slot-value pairs.

The *Inside-Outside-Beginning (IOB)* format is commonly used to annotate sequences, making it easier to identify which tokens of a sequence are at the beginning, inside, and outside of slot values. Machine learning models (such as neural networks with CRFs (Lafferty et al., 2001) or encoders with classification layers) are then trained to predict these labels, ensuring that the slot-filling process respects the order and context of the input tokens. Advanced machine learning models are able to handle slot filling on complex input data without the need for tedious formulation of rules. These approaches are widely prevalent in the current research.

### 2.3.1 IOB Format

The Inside-Outside-Beginning (IOB) format, probably first introduced by Ramshaw and Marcus (1995), is a widely used annotation scheme for sequence labeling tasks, including slot filling. In the IOB format:

- `B-slot_type` marks the beginning of a value belonging to a `slot_type`,

- `I-slot_type` indicates continuation of a value belonging to a `slot_type`,
- `O` denotes a token that is outside of any slots.

This format has become standard due to its simplicity and effectiveness in capturing the structure of labeled sequences. With only $(2 \cdot \text{number of slots} + 1)$ markers, we are able to exactly describe the sequence.

### 2.3.2 Sequence Labeling Through Classification

In classification-based sequence labeling, each token in the input sequence is usually embedded into a vector representation. This vector is then fed into a classifier to predict the corresponding slot label. Context-aware embeddings (from RNNs or Transformers), as well as additional features, naturally lead to better results. Sequence labeling models can capture dependencies between tokens and provide more accurate slot labeling because they deal with the utterance on the token level.

Z. Huang et al. (2015) use LSTMs and CRFs to address this task. Goo et al. (2018) perform joint NLU using an attention-based RNN model with a slot-gated mechanism trained to predict only corresponding slots and intents. Q. Chen et al. (2019) use BERT to encode the sequence and add a fully connected layer to classify each word.

### 2.3.3 Generative Sequence Labeling (seq2seq)

Rather than directly classifying each token, we can train the model to generate a sequence of labels corresponding to each token in the correct order using a seq2seq approach. This is typically done with an encoder-decoder architecture, often Transformer-based for both components. The encoder embeds the input sequence tokens and processes them into a fixed-sized sequence representation. The decoder then (usually) autoregressively generates labels from this representation, using attention to consider both the input token embeddings and the labels generated up to the current timestep.

In comparison with the classification approach, autoregressive generation allows the model to consider labels generated so far, possibly allowing for more coherent output. At the same time, the model needs to learn all the dependencies so that it generates the labels in the correct order. Multiple works utilize encoder-decoder models with attention based on bidirectional LSTM for this task (B. Liu; Lane, 2016; Zhao; Z. Feng, 2018).

With generative models, we also return back in circle to the direct classification. The models may be trained to generate basically arbitrary sequences. Instead of labeling individual tokens, we can diverge from the sequence-labeling formalization and train the models to output slot-value pairs directly. Some research takes this approach (X. Li et al., 2023), and with the boom of LLMs, we believe it may be perspective.

# 3 Slot Filling in Voice-Based Systems

As we have outlined in the introduction, the problem of extracting meaning and important values from users' utterances gets much more complicated when faced with speech input. For these reasons, we have also witnessed the emergence of different purely voice-based understanding benchmarks (Bastianelli et al., 2020; Filice et al., 2021; L. Feng et al., 2021). Suboptimal experimental results of the pipelined systems on these benchmarks prove that the problem requires new solutions.

To reiterate, the pipelined systems consist of an ASR module to transcribe the audio and an NLU module to perform the understanding of the transcribed text. We refer to it as a basic approach because it does not make any changes to the NLU module. While such an approach allows for fast and independent reiteration of both modules, this independence also causes information loss. Errors in the ASR transcription propagate through the pipeline and inevitably lead to poor NLU results. Furthermore, some information is not contained in the text at all. For example, for an emotion detection module, the information about the speed, pitch, or volume of the speech can be irreplaceable.

On the other side of the spectrum lie end-to-end SLU systems. These systems merge ASR and NLU into one module, taking audio as the input and producing an understanding representation of the output. As the availability of data and computing power grows, there is also plenty of work in this direction (Serdyuk et al., 2018; Lugosch et al., 2019; P. Wang et al., 2020; Denisov; Vu, 2020; Desot et al., 2022; Sun; C. Zhang; Woodland, 2023; Sun; C. Zhang; Vulic, et al., 2023). However, as we have already mentioned, the barriers of data and computing power availability are, in many cases, still significant. For these reasons, we will not discuss these approaches here, as our goal lies elsewhere.

In this chapter, we have tried to split different methods of reducing the boundaries between ASR and NLU into categories based on the aspect we consider predominant. We have focused on slot filling, but not uniquely, because we believe it is possible to draw inspiration also from methods aimed at other NLP tasks. Especially *Named Entity Recognition (NER)* can often serve for the same purpose. The presented overview is in no way exhaustive; its main purpose is to provide a grasp of the ideas.

Section 3.1 discusses approaches utilizing multiple best ASR hypotheses, often through reranking. Methods covered in Section 3.2 utilize the multiple hypotheses and other ways to fix the ASR output along with understanding. Section 3.3 explores techniques that utilize phoneme representations as an additional model input. Section 3.4 returns to purely textual inference-time input with a goal of pronunciation-aware embeddings. Finally, in Section 3.5, we discuss methods that utilize some intermediate representations to perform NLU joint with final transcription decoding.

## 3.1 Multiple ASR Hypotheses

One of the most straightforward ways to improve the accuracy of SLU is to take into account multiple hypotheses proposed by the ASR. These hypotheses are usually called n-best lists.

Morbini et al. (2012) work with a dataset of around 14k utterances collected in a museum. The goal is domain classification into one of the 168 response labels, i.e., very similar to intent detection. They first train a classifier on the gold transcriptions with gold labels and then use it to generate pseudo labels for different ASR hypotheses. Afterwards, they use this new dataset to train another classifier. This classifier rescores the hypotheses along with their labels and picks the best one.

They use three ASR engines to get three n-best lists ($n = 30$) of transcribed audio. Two of the engines are off-the-shelf (one with a custom LM), and one is trained on the dataset. Their WER on the test set is between 20 and 30 %. The NLU on the custom ASR transcriptions outperforms the other variants, and reranking does not bring any improvements. However, for the off-the-shelf ASR variants and their combinations, the reranking results in absolute accuracy improvements of 1 to 10 %. The NLU accuracy on the reranked combined off-the-shelf ASR transcriptions gets close to that on the custom ASR transcript.

M. Li et al. (2020) report that on their explored dataset, in 50 %, the correct ASR transcription was in the 5-best list, but it was not the 1-best. They experiment on a dataset of around 9 million utterances classified into 23 domains. Though not explicitly stated, it is probably extracted from Alexa (Amazon virtual assistant), as all of the authors are affiliated with the company.

Instead of rescoring, their goal is to utilize all the information available in the different hypotheses. They try two versions; in the first one, they concatenate all the hypotheses, separating them with a special token. BiLSTM is then utilized to embed the sequences. In the second version, they embed each sequence separately (again using BiLSTM) and then concatenate the embeddings. The latter version proves more effective with 14% relative F1 improvement.

X. Liu et al. (2021) also perform domain classification on a dataset of around 10 million utterances extracted from Alexa, and their approach is quite similar. They use all the hypotheses from the n-best list, pass them through a shared decoder, and concatenate the representations. Using multiple CNNs, they then extract the fused features and perform a classification of them. They report a 22 % relative classification error rate reduction on the part of the dataset where NLU on the 1-best ASR hypothesis was erroneous. The absolute F1 improvement on the full test set is 0.14 % only, but they argue this is largely because of the size of the dataset.

Peris et al. (2020) look at n-best utilization as a summarization task with the goal of extracting information from the hypotheses. They use a dataset from Alexa again but in German and Portuguese (around 3 million utterances). Their primary focus is domain classification, but they also experiment with joint indent detection and named entity recognition. Especially their NER part deals basically with slot filling.

They treat the task as a seq2seq problem. They separate the n-best list with a special token, embed it with BERT, and concatenate the encodings. A decoder then generates either intent/slot boundaries or pointers to the input sequence. They use semantic error rate as a metric and report around 10% relative improvement for both languages on the part of the test set where the 1-best ASR hypothesis is wrong. However, on the full test set, the results are significantly worse than the baseline (19 % for Portuguese and 107 % for German).

Some works take this approach one step further, and instead of multiple hypotheses, they utilize the whole graph of possible paths over tokens that represent the audio sequence (Tür et al., 2013; Ladhak et al., 2016; Shivakumar et al., 2019; C. Huang; Y. Chen, 2020b).

## 3.2 Fixing ASR Output

A second branch of work we identified focuses on fixing the ASR output. There has been research on doing so independently on understanding (Tam et al., 2014; Ogawa et al., 2018; Ogawa et al., 2019). However, our main interest lies in methods that fix ASR in conjunction with understanding.

Schumann and Angkititrakul (2018) generate SLU dataset using an approach utilized in many other studies. They start with a textual NLU dataset of Airline Travel Information Systems (ATIS) (Hemphill et al., 1990). Using a text-to-speech (TTS) engine, they generate spoken versions of the utterances and add artificial noise. ASR transcriptions of such audio thus become erroneous, simulating real-world scenarios. While this approach allows to obtain some simulated data when only texts are available, it is still not quite realistic as it lacks some authenticity.

They propose an encoder-decoder model based on RNNs. They train the model jointly for indent detection, slot filling, and ASR hypothesis correction. A single encoder processes the ASR output. One decoder classifies the sequence embedding into the intent category. This predicted intent is fed along with the encoder representation to a second decoder, which autoregressively predicts the corrected sequence. This sequence is again encoded by the same encoder and, together with the predicted intent, passed to the third decoder. This decoder autoregressively predicts slot labels.

The WER of the transcribed audio is around 14 %. The proposed joint model surpasses the baselines by 1 to 3 % absolute slot F1 while reducing the absolute WER by 4 %.

Weng et al. (2020) present the most exhaustive study of the ones discussed so far. They use a slightly modified DSTC2 dataset (Henderson et al., 2014), where the goal of the system is restaurant recommendation based on provided user preferences. A 10-best list of ASR transcriptions is available.

Multiple approaches are proposed. Two of them perform n-best list reranking, using LM-based GPT or hierarchical CNN/RNN, possibly joint with the understanding. Best slot F1 is achieved using the third approach. In this approach, the ASR hypotheses are aligned using the word-level Levenshtein distance to obtain a word confusion network. All possible words for each timestep are then embedded, and these embeddings are concatenated and passed through a BiLSTM RNN. The original embeddings are added using residual connections, and then multi-head self-attention is applied to obtain the final predictions. For each of the input timesteps, the model predicts slot IOB tag as well as corrected output word. This approach achieves 2 % absolute slot F1 improvement over the baseline.

## 3.3 Phoneme Representations

Another class of research focuses on exploiting phoneme representations. They argue that ASR engines usually produce outputs whose pronunciation is similar to the correct transcription. Here, we focus on methods that take phoneme sequences as an input. As phonemes record the pronunciation, they should serve as a better medium of representing speech. Fang et al. (2020) confirm this theory by showing that the phoneme error rate (PER) of an ASR is considerably lower than WER on multiple datasets.

They then proceed to train context-aware phoneme embeddings using algorithms inspired by Word2Vec (Mikolov et al., 2013). These phoneme embeddings are then plugged into CNN/RNN architectures and tested on domain classification on multiple datasets. The reported results show consistent improvements over using only word and/or character embeddings.

Kim et al. (2021) present a similar approach with their Speech-Text BERT (ST-BERT) model. As the name suggests, they integrate the phoneme and word embeddings into one BERT-like model and train it using masked language modeling. They also incorporate the acoustic model, which produces phoneme posteriors, into their architecture. Nevertheless, we place their method here, as the phoneme posteriors could be produced by an external ASR. Improvement in intent detection on multiple datasets over baselines is not surprising; their most interesting feat is very competitive results when limiting the training to only 1 % of the dataset.

Z. Wang et al. (2022) achieve good results using simple BiLSTM RNN encoders. They encode the word sequence and the phoneme sequence separately and then perform cross-attention over the two representations. Yenigalla et al. (2018) use both spectrograms and phoneme sequences as an input for their model, placing them nearly into the field of end-to-end systems. They find that their model performs better on emotion classification when also supplied with the phoneme sequences when compared to their and other models using spectrograms only. We argue this result is not surprising since providing the model with phoneme sequences allows it to focus on other speech features.

## 3.4 Pronunciation Robust Embeddings

Here, we get back to models that take ASR transcriptions in the form of words as input. With contextual word embeddings, the goal is to embed words/sentences with similar meanings into similar regions of the embedding space. The focus of pronunciation (or ASR) robust embeddings is to further improve these representations so that also similar sounding words are close in the embedding space. The idea is that subsequent models processing the representations should then be robust enough to swap similar-sounding words.

Sundararaman et al. (2021) present PhonemeBERT. Overall, the scheme is very similar to the ST-BERT discussed in the previous section, but PhonemeBERT does not include the acoustic model. The token embeddings and model weights for the word transformer are initialized from RoBERTa (Y. Liu et al., 2019). During training, the model gets both the phoneme sequence and the ASR transcript. They train the model using masked language modeling, where both the words and the phonemes can be masked. The loss consists of three parts – word MLM loss, phoneme MLM loss, and joint MLM loss – where the model can always attend to the respective part of the input. They argue that this should encourage the model to align the phoneme and word representations.

We include this model in this section because, as the authors show in the experiments, PhonemeBERT can be pretrained and then used in a low-resource scenario where only the ASR transcriptions (and not phoneme sequences) are available. They test the model on indent detection on multiple datasets and report that even when the phoneme sequence is not available during testing, the model outperforms a version of the model trained on words only. The ability to process erroneous ASR transcripts demonstrates that the learned word embeddings are pronunciation-aware to some extent.

C. Huang and Y. Chen (2020a) take a very different approach. Instead of expanding the input to the model, they propose radical changes to the fine-tuning scheme. They start with ELMo (Peters et al., 2018) and finetune this language model using word confusion networks.

These networks are extracted either supervised by aligning the ASR hypotheses to the gold transcription or unsupervised by aligning ASR hypotheses to one another using Levenshtein distance. Special confusion loss is added to the standard language modeling loss. This confusion loss computes the cosine distance between model embeddings of words that are likely to be swapped according to the word confusion network. By minimizing this loss, the representations of words often mistaken by the ASR should appear closer in the embedding space.

BiLSTM is then added on top for intent detection on three different datasets. The model fine-tuned with the confusion loss outperforms the one fine-tuned using language modeling only by around 2 %. Interestingly, for some datasets, the supervised confusion network extraction outperforms the unsupervised; for some, it is the other way around.

Ruan et al. (2020) focus both on intent detection and on NER/slot filling. We will discuss the latter here. They propose to obtain ASR hypotheses for the training audio and pseudo-label them according to the labels of the gold transcriptions. Two methods are presented for pseudo-labeling – token exact match and entity exact match. The former assigns the gold slot labels to all tokens (words) from the ASR hypothesis that match the gold entity partially, while the latter labels only the whole matching entities and assigns a special label to the noncomplete matches.

This pseudo-labeled data is then added to the training dataset. The approach brings close to no improvements over the baseline on the ATIS dataset. On their private Alexa dataset, they report a 1 to 11 % relative improvement of slot F1 metric on different domains.

## 3.5 NLU Joint With Decoding

In this section, we will explore methods that use intermediate representations from the ASR as input. These can include the encoded audio or phoneme sequences. Performing NLU (or other tasks) on these representations then allows the models to aid with final text transcription decoding.

P. Wang et al. (2022) propose a framework composed of two parts. The first part is an acoustic model that produces the phoneme sequence posterior. An external acoustic model can be used, satisfying our condition to use a general ASR module.

We are interested in the second part of the framework, dubbed Speech2Slot. It is a Transformer-based encoder-decoder model. The encoder takes the utterance phoneme posterior as an input and embeds it. A fully connected layer is added on top to perform masked language modeling of the phoneme sequence during training. This is added as a regularization technique to prevent overfitting.

The decoder then decodes the slot value, performing multi-head attention over the encoder output. Additionally, gated attention is proposed to further attend to the part of the encoded sequence where the slot value is predicted to lie. This decoding is further constrained using a trie structure during the inference. The trie holds available slot values. The authors try decoding the value from three directions – forward, backward, and from the center to both sides. They argue this should help when the audio is clearer in some parts than in others.

Speech2Slot shows big improvements over the baselines on a TTS-generated noisy dataset of navigation prompts in Chinese. A big advantage of this framework lies in the possibility of expanding the trie with new slot values without the need for retraining the model. It also has some significant limitations, though. Mainly, as far as our understanding goes, it is able to detect only one slot value in the input.

Gaido et al. (2023) focus on NER in speech-to-text translation. As neural methods struggle to correctly translate named entities, their goal is to detect them in the input and translate them using a named entity dictionary. They report an unsuccessful attempt to use Speech2Slot architecture, where their models failed to converge. Instead, they utilize a different speech encoder and add a detector. This detector consumes the encoder output and a named entity representation and predicts whether this named entity is present in the input or not. While quite successful, we argue this approach is very inefficient and next to impossible to use with a database of thousands of named entities (or slot values, in our case).

# 4 Our Approach

In this chapter, we present our own approach to slot filling in SLU. Our contributions are mainly two-fold – first, we present a dataset for SLU slot filling in the Czech language (Section 4.1); second, we propose a novel architecture for slot filling joint with decoding (Section 4.2). We describe the details of training and inference in Section 4.3. In Section 4.4 we document the results of our experiments, notably in Section 4.4.3 we perform error analysis of our best architecture.

## 4.1 Dataset

The presented dataset deals with the public transport domain. The original audio recordings with gold transcriptions (Plátek et al., 2016) come from the Vystadial project.[1] This section discusses how we have utilized the original data to create a slot-filling dataset.

Data in Vystadial were collected using the Alex dialogue system (Dusek et al., 2014). Users could call a specific telephone number to talk with the system in Czech. Supported queries included weather, time, and mainly public transport (initially in Prague only, later in Czechia in general). The users could query the system for information about public transport connections between different places in Czechia, specifying their departure or arrival stops/cities (we will refer to them as *locations*), as well as intermediate travel points. Therefore, the original dataset (Plátek et al., 2016) contains thousands of Czech spoken queries such as "I want to travel from Brno to Prague," together with their respective gold transcriptions. As the system was able to ask clarifying questions, some user utterances included only the name of a location. However, the released dataset is audio only, with no NLU annotation.

### 4.1.1 Annotation

The original Alex rule-based NLU served as a basis for annotating the slot values in the dataset. We annotate the golden transcriptions using a modified version of the Alex NLU. Furthermore, we use original, previously unreleased XML logs from the Alex system to restore the dialogue context. We keep only utterances where the NLU detects a slot value of interest, resulting in roughly 30k utterances. For each of the utterances, we keep the previous system utterance and its respective dialogue act (representation of the utterance meaning) based on the dialogue context. As the user sometimes responds to the system's question (e.g., "What is your departure stop?") with the stop's name only, the previous system dialogue act is necessary to determine which type of stop it is. We also record whether this is necessary for a given utterance in our dataset. The caller ID corresponding to the given sample is retrieved from the Alex logs.

We identify 8 slot types of interest in the Alex NLU. It originally worked with additional slot types but we deem them not suitable for our task. The slot types of interest include a combination of either a city or a specific stop, with their uses on

[1]`https://ufal.mff.cuni.cz/grants/vystadial`

the route: departure (`from`), intermediate (`via`), arrival (`to`), and unknown, which appears when the dialogue context does not contain the information necessary to determine any other type. In the dataset, we abbreviate these slot types as `fc, vc, tc, cc` for cities and `fs, vs, ts, ss` for stops.

The Alex NLU is aided by a database of all the Czech cities and public transport stops. Because Czech is an inflectional language, the database is enriched by all the inflected forms of names using MorphoDiTa (Straková et al., 2014). We include this database in the dataset as a `.txt` file.

Alex NLU marks all the segments of the user utterance that are found in the database. Then, it uses rules implemented by regular expressions to determine the slot type of the location. Although simple in its principles, this NLU was iteratively tuned to handle dialogues it struggled with before, and the result is quite robust.

We had to make certain improvements in the Alex NLU to adapt it for our annotation goals. First and most importantly, the original NLU saved only the lemmas of locations found in the utterance with their slot type. We adjust the NLU to provide more information. We post-process this information further to obtain the exact forms mentioned in the utterance, the spans where they appear in the utterance, and their lemmas.

Second, the NLU worked with the outputs of a proprietary ASR that were all lowercase. We adapt it to work with true-case data, as the gold transcriptions are true-case, and lowercasing would lead to unnecessary data loss. We also remove the normalization that the NLU did (some words were replaced for easier rule application) and apply our own normalization, removing nonspeech tokens, like `(unint)`, from the transcriptions. We keep both the original transcription and our normalization in the dataset.

The dataset splits are kept in separate `.jsonl` files. An example sample with an explanation is shown in Figure 4.1. Here, we provide a detailed description of the values saved for each sample:

- `normalized`: our version of normalized transcription; this sample translates to roughly: "[I want to travel] from Hradec Králové to Litomyšl by bus on Friday at twelve o'clock."
- `slot_values`: a list of slot values present in the utterance, each with the exact form that appears in the utterance (`value`), lemma for this form (`value_lemma`), slot type (`slot`) and `span` that points to the `normalized` version of the utterance.
- `need_both`: marks if the previous system utterance was necessary to determine the slot type.
- `original`: the original gold transcription.
- `prev_sys_utt`: the previous system utterance; here roughly translates to: "Good morning, this is Alex, public transportation information. The call is recorded. How can I help you?"
- `prev_sys_da`: the dialogue act semantic representation of the previous system utterance.

- `path`: relative path to the audio recording as saved in the dataset. Without the split identification, it corresponds to the structure in the original dataset (Plátek et al., 2016).
- `caller_id`: hashed telephone number that identifies the caller.

```
{"normalized": "Z Hradce Králové autobusem do Litomyšle
               v pátek ve dvanáct hodin.",
 "slot_values": [
   { "value": "Hradce Králové",
     "value_lemma": "Hradec Králové",
     "slot": "fc",
     "span": [2, 16]},
   { "value": "Litomyšle",
     "value_lemma": "Litomyšl",
     "slot": "tc",
     "span": [30, 39]}],
 "need_both": false,
 "original": "(hum) z Hradce Králové autobusem do Litomyšle
             v pátek ve dvanáct hodin",
 "prev_sys_ut": "Dobrý den, tady Alex, informace o veřejné dopravě.
                Hovor je nahráván. Jak Vám můžu pomoci?",
 "prev_sys_da": "hello()",
 "path": "dev/part18/2015-04-20--19-33-00.830072-CEST-2204d2213de53dd9
         /vad-2015-04-20--19-33-14.814678.wav",
 "caller_id": "2204d2213de53dd9"}
```

**Figure 4.1** A sample from our new dataset. Refer to Section 4.1.1 for detailed description.

### 4.1.2 Dataset Splits

The original data contained user identification using hashed phone numbers for privacy reasons, but this anonymization was not applied to a part of the data. We have applied the anonymization consistently on all data and utilized the phone number hashes to split the dataset properly into train, development, and test sets. This means that the distinct data splits include non-overlapping sets of speakers. The goal is to ensure the splits truly measure the generalization abilities and that the test set is not contaminated by samples very similar to the ones used in training.

The first step is to ensure two utterances (dialogue turns) from the same user do not appear in different splits. We categorize the users into three groups: those with more than 40 turns (we call this category `many`), those with between 15 and 40 turns (`mid`), and those with less than 15 turns in the data (`few`).[2] Then, we distribute the users of different categories into splits, aiming at a roughly

[2]The boundaries were selected after manual inspection of the data in order to produce reasonably sized groups.

60/20/20% split of the data. Detailed statistics of the dataset can be found in Table 4.1.

| | Number of turns from user categories | | | | #users |
|---|---|---|---|---|---|
| | `many` | `mid` | `few` | Total | |
| **Train** | 14775 | 2472 | 1493 | 18740 | 474 |
| **Development** | 3001 | 1438 | 1492 | 5931 | 334 |
| **Test** | 2014 | 2127 | 2016 | 6157 | 449 |
| **Total** | 19790 | 6037 | 5001 | 30828 | 1257 |

**Table 4.1** Number of turns from different user categories in the presented dataset and its splits. **#users** refers to the number of unique users. Distribution of the users into categories is discussed in Section 4.1.2.

## 4.2 Architecture Variants

In this section, we describe different architectures we evaluated in our experiments. We present a straightforward baseline followed by our proposed architecture and its iterative improvements. In all of our architectures, we use a version of Whisper `mikr/whisper-large-v3-czech-cv13` fine-tuned on the Czech part of the Common Voice dataset (Ardila et al., 2020). Both the model[3] and the dataset[4] are publicly available on the HuggingFace hub (Wolf et al., 2019). From now on, when we talk about Whisper, we refer to this fine-tuned version. We also always prompt it with special tokens `<|startoftranscript|><|cs|><|transcribe|>` `<|notimestamps|>` to specify the task.

In all of our experiments, we work only with 4 slot labels instead of the initial 8 identified in the Alex NLU. We argue that all methods, apart from rule-based, have next to no chance of differentiating between stops and cities. Therefore, we merge the respective pairs into a general *location* (such as departure stop and departure city to departure location) and work only with 4 slot labels.

### 4.2.1 Baseline

We utilize the classic ASR + NLU pipeline as a baseline. We transcribe the audio using the already mentioned fine-tuned Whisper. We choose a beam search with a beam of size 3 to obtain more accurate ASR transcription. Then, we annotate the transcriptions using the same Alex-based NLU used for dataset annotation. Thus, this approach should yield perfect results if the ASR was perfect. Because it is not, we obtain an interesting baseline to beat with more integrated SLU methods, see Section 4.4.

### 4.2.2 Whisper-based Joint Architecture

We propose a novel Whisper-based architecture that joins slot filling with decoding. As we have already mentioned in the introduction, the goal was to use

[3]`https://huggingface.co/mikr/whisper-large-v3-czech-cv13`
[4]`https://huggingface.co/datasets/mozilla-foundation/common_voice_13_0`

an off-the-shelf ASR and possibly utilize its inner representations, as the final output may be erroneous. Furthermore, we wanted to leverage the preliminary knowledge of the slot value database.

From the approaches discussed in Chapter 3, Speech2Slot (P. Wang et al., 2022) bears the most similarities. However, in theory, our architecture should be able to handle multiple slot types as well as multiple slot values in a single utterance.

We treat the task as a sequence labeling problem. The units of our sequence are tokens generated by the Whisper decoder. As we describe in Section 1.9, the Whisper decoder generates the audio transcription autoregressively (token-by-token) on the level of subwords. In each timestep, Whisper is conditioned on the audio encoding, prompt tokens, and tokens generated in previous timesteps. Based on this, the Transformer-based part of the Whisper decoder produces the output encoding for a given timestep. This output is then passed through a fully connected layer to obtain a probability distribution over vocabulary tokens. Further handling of this distribution depends on the specific decoding method.

See Figure 4.2 for an illustration of the architecture described in the following paragraphs. We intentionally leave the diagram high-level here as we later propose multiple approaches for how exactly the model should be connected. The main idea is that we add a second decoder (referred to as *Slot decoder*) for slot labeling that is conditioned on the encoded audio, the so far decoded transcription, and on its own sequence generated in the previous timesteps.

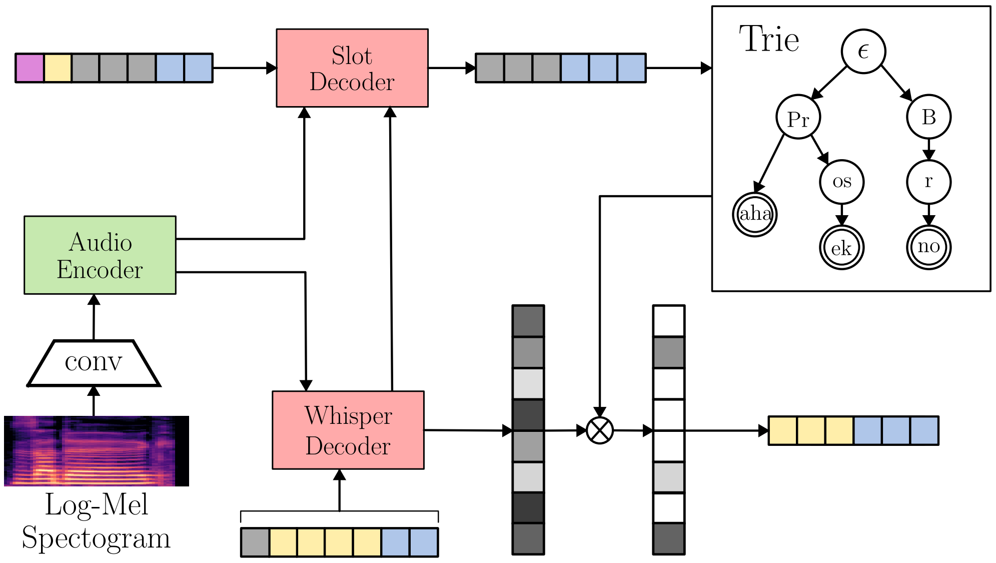


**Figure 4.2** High-level overview of the proposed architecture. The yellow tokens represent special prompt tokens, the blue tokens denote standard predicted text tokens. The grey tokens represent padding and the violet token the dialogue context. The token probability distribution predicted by the Whisper Decoder represented by the vertical rectangle is constrained by the Trie. Image inspired by Figure 1 of the original work (Radford et al., 2023); adapted by the authors.

### Whisper Notation

Let us introduce a notation here for a better orientation in the following descriptions. $||$ denotes concatenation. Audio recording is the input to the Whisper encoder, and we will denote its output $WhisperEnc(x_1, \ldots, x_n)$. In timestep $t$, we generate a probability distribution over vocabulary tokens $p_t$. Tokens generated in the previous steps $y_1, \ldots, y_{t-1}$ (together with the prompt tokens; we will not consider them separately) are embedded by the Whisper decoder embedding layer as $WhisperEmb(y_1, \ldots, y_{t-1})$. These embeddings and the audio encoding are consumed by the Transformer-based part of the Whisper decoder to obtain its output for all the timesteps:

$$\begin{aligned}WhisperDecOut_{1,\ldots,t} = WhisperDec[&WhisperEmb(y_1, \ldots, y_{t-1}),\\ &WhisperEnc(x_1, \ldots, x_n)]\end{aligned}$$

The resulting next-token logits are then obtained by passing the timestep $t$ output through the final fully connected layer $p_t = WhisperFFCL(WhisperDecOut_t)$.[5]

### Our Slot Decoder

We add a second decoder *SlotDec* with the same architecture to label the tokens. The vocabulary of the Slot decoder is *token slot labels* in the IOB format, i.e., whether a slot value starts on the given token, continues there, or this token is out of any slot value (as we describe in Section 2.3). In all our model versions, the Slot decoder receives the audio encoded by the Whisper encoder $WhisperEnc(x_1, \ldots, x_n)$ as the encoder context vector. The decoding is then also autoregressive. The Slot decoder outputs a distribution over token slot labels $s_t$ (for timestep $t$):

$$s_t = SlotFFCL(SlotDecOut_t)$$

Based on this distribution, we choose the token slot label with the highest probability $l_t = \text{argmax}(s_t)$.

We utilize the same Transformer-based decoder architecture as the Whisper. In preliminary experiments, we tested applying $\{1, 2, 4, 8, 16\}$ of the decoder layers. Utilizing a higher number of layers did not provide any improvements. Therefore, our Slot decoder always has 1 Transformer-based decoder layer. The Slot decoder has roughly 27M trainable parameters, compared to the 1543M of the Whisper used for ASR.

Note that the tokens $y_1, \ldots, y_{t-1}$ include the Whisper prompt tokens. We need the sequences to be aligned, i.e., $l_1$ be the token slot label of the token $y_1$. We label the Whisper prompt tokens with a token slot label representing padding and do not mention this explicitly further.

**Context Input** As mentioned in the previous section, some utterances in the dataset are not self-contained, and the prior context is necessary for slot type determination. We encode the context into the same vocabulary, e.g., if the system asked the user about the departure location, we use the token slot label that

[5]To obtain a proper probability distribution, we would need to apply *softmax* over the output of the fully connected layer. This function, however, does not change the ordering of the outputs, therefore we omit it here.

represents the departure location. We prepend these tokens as an initial input to the Slot decoder and denote it $c_1, \ldots, c_m$ (as the system may have queried multiple slots).

**Combining Whisper and Slot Representations** In all of our versions, we combine a part of the Whisper decoder representations with the embeddings of the context and so far generated token slot labels $SlotEmb(c_1, \ldots, c_m, l_1, \ldots, l_{t-1})$. We then sum these representations with the Whisper decoder representations, as this is a common approach for combining representations (Vaswani et al., 2017). In order to sum them, we need them to be the same shape. Therefore, we pad the Whisper decoder representations using a sequence of Whisper pad tokens $a_1, \ldots, a_m$ to account for the Slot decoder prompt tokens.

**Trie-Constrained Decoding** Based on the available database, we build a trie (Bodon; Rónyai, 2003) of all possible location names. Then, we use it together with the token slot label predicted by the Slot decoder $l_t$ to modify the token probabilities predicted by the Whisper decoder $p_t$. If the token slot label signifies that the given token belongs to a slot value $l_t \in \{I, B\}$, we query the trie for all the possible next tokens. Then, we set the probabilities of all other tokens in $p_t$ to minus infinity. In the next timesteps $t+1, \ldots, t+x$, we do not allow the model to stop decoding a slot value unless the trie indicates that the result produced so far is valid. This way, we constrain the decoding so that once the Slot decoder detects a slot value, the decoded tokens must form a valid location name.

## Model Versions

**Initial Version (`Initial`)** In the initial version, the idea was to leverage as much of Whisper's inner representations as possible. Therefore, when predicting the token slot label for timestep $t$, the Slot decoder receives the output of the Transformer-based part of the Whisper decoder for timesteps $1, \ldots, t$. Using our notation:

$$\begin{aligned} SlotDecOut_{1,\ldots,t} = SlotDec[&((a_1, \ldots, a_m)||WhisperDecOut_{1,\ldots,t}) \\ &+SlotEmb(c_1, \ldots, c_m, l_1, \ldots, l_{t-1}), \\ &WhisperEnc(x_1, \ldots, x_n)] \end{aligned}$$

$$s_t = SlotFFCL(SlotDecOut_t)$$

**Correct Sequence Shift (`Seq-Shift`)** During preliminary experiments, we have realized that the initially proposed approach introduces a misalignment in the sequences. The Whisper decoder output $WhisperDecOut_{1,\ldots,t}$ already contains the output for the timestep $t$. The approach, therefore, sums $SlotEmb(l_{t-1})$ with $WhisperDecOut_t$, and the whole sequences are shifted, i.e., we combine a Whisper decoder representation with the representation of the token slot label for the previous token.

We decide to align the sequences, and because all the timesteps $1, \ldots, (t-1)$ are already decoded, we use token embeddings $WhisperEmb(y_1, \ldots, y_{t-1})$ instead of the $WhisperDecOut_{1,\ldots,t}$. To still utilize the information from the Whisper decoder for the given timestep ($WhisperDecOut_t$), we concatenate it to the input of the final fully connected layer:

$$\begin{aligned}SlotDecOut_{1,\ldots,t} = SlotDec[&WhisperEmb(a_1,\ldots,a_m,y_1,\ldots,y_{t-1})\\ &+SlotEmb(c_1,\ldots,c_m,l_1,\ldots,l_{t-1}),\\ &WhisperEnc(x_1,\ldots,x_n)]\\ s_t = SlotFFCL(&SlotDecOut_t || WhisperDecOut_t)\end{aligned}$$

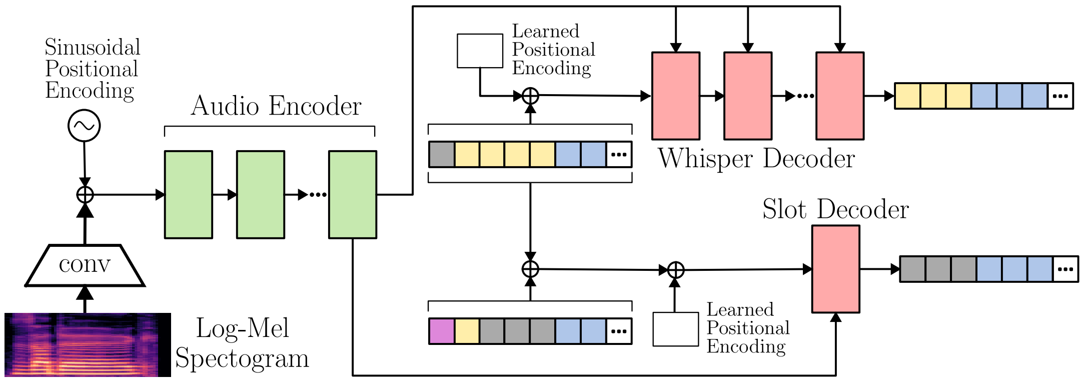

Sinusoidal Positional Encoding
Audio Encoder
conv
Log-Mel Spectogram
Learned Positional Encoding
Whisper Decoder
Slot Decoder
Learned Positional Encoding

**Figure 4.3** The details of model connection in our final architecture. The decoding is constrained by the Trie, as shown in Figure 4.2 (we do not repeat it in this figure). Again, the yellow tokens represent special prompt tokens, and the blue tokens denote standard predicted text tokens. The grey tokens represent padding, and the violet token the dialogue context. Image inspired by Figure 1 of the original work (Radford et al., 2023); adapted by the authors.

**No Decoder Output Concatenation (`Final`)** We argue the Whisper decoder output $WhisperDecOut_t$ may be noisy during the autoregressive decoding. Therefore, our final architecture illustrated in Figure 4.3 does not include it in the last layer:

$$\begin{aligned}SlotDecOut_{1,\ldots,t} = SlotDec[&WhisperEmb(a_1,\ldots,a_m,y_1,\ldots,y_{t-1})\\ &+SlotEmb(c_1,\ldots,c_m,l_1,\ldots,l_{t-1}),\\ &WhisperEnc(x_1,\ldots,x_n)]\\ s_t = SlotFFCL(&SlotDecOut_t)\end{aligned}$$

## Ablation Versions

We perform two ablation experiments to assess the utility of different inputs to the Slot decoder.

**No Encoder (`Final-NoEnc`)** In this variant, we want to determine whether the Slot decoder benefits from the availability of audio encodings. Therefore, we zero out the audio encoding, resulting in an equivalent of:

$$\begin{aligned}SlotDecOut_{1,\ldots,t} = SlotDec[&WhisperEmb(a_1,\ldots,a_m,y_1,\ldots,y_{t-1})\\ &+SlotEmb(c_1,\ldots,c_m,l_1,\ldots,l_{t-1})]\\ s_t = SlotFFCL(&SlotDecOut_t)\end{aligned}$$

**No Decoder (`Final-NoDec`)** The goal of the second ablation experiment is to evaluate how the Slot decoder uses the preceding text tokens. This version is conditioned only on the audio encoding and the preceding token slot labels:

$$\begin{aligned} SlotDecOut_{1,\ldots,t} &= SlotDec[SlotEmb(c_1,\ldots,c_m,l_1,\ldots,l_{t-1}), \\ &\qquad WhisperEnc(x_1,\ldots,x_n)] \\ s_t &= SlotFFCL(SlotDecOut_t) \end{aligned}$$

## 4.3 Training and Inference Methods

We implement our model using PyTorch (Paszke et al., 2019) and HuggingFace (Wolf et al., 2019) Python libraries. We train it on the ÚFAL AIC cluster using NVIDIA GPUs.[6] Our default training regime works with the common Transformer teacher forcing for both the token slot labels and the text tokens. We use a batch size of 2 samples because of GPU memory limitation and a learning rate of 0.0001, as it worked well in our preliminary experiments. We train the models for 2 to 8 epochs while evaluating them on the development set every even epoch. We then keep the model with the best development set results.

The default inference regime uses greedy decoding, i.e., it chooses only the one most probable prediction at every timestep. We use a batch size of 1 during inference to simplify the implementation. We also implement two extensions to the default setup; we detail them in this section.

### 4.3.1 Pseudo Scheduled Sampling

Inspired by scheduled sampling (Bengio et al., 2015), we try to implement a similar approach in order to better train the models for autoregressive decoding. Unfortunately, there is no straightforward utilization of this technique in Transformer models with teacher forcing, as they compute all the attention scores in parallel.

To bridge the gap between training and inference time, we first let the model generate the label sequence (text tokens and token slot labels) without teacher forcing. We sample from these generated sequences and intermix them with the gold labels. Then, we use these mixed tokens as input for the standard teacher-forcing method. We naturally compute the loss and gradients based on the (non-mixed) gold labels. As we need to run the inference code here, we also use the `batch_size=1` for this training.

Following Bengio et al. (2015), we start with the mixing ratio of 0.5 of non-gold tokens in the first epoch. We gradually increase it to 0.6, 0.7, 0.9, and 1.0 in the next epochs. We experiment with two variants: first, we mix only the token slot labels; second, we mix both token slot labels and the text tokens.

### 4.3.2 Beam Search

We implement a simplified version the a constrained beam search (Hokamp; Q. Liu, 2017). Initially, we choose the most probable token slot label $l_1$ and

[6]Based on current availability, one of GTX1080Ti, RTX2080Ti, RTX3090.

accordingly constrain the text token probabilities $p_1$. Then, we keep `beam_width=3` (to correspond with the baseline) best text tokens and assign each of the tokens a token slot label $l_1$. This creates the initial set of hypotheses.

At every following timestep $t$, we expand each of the hypotheses independently. Let us assume we work with hypothesis $h$. First, we predict the most probable token slot label $l_{t,h}$ for the given hypothesis. We do not consider multiple token slot labels as there is no straightforward way to score the combined hypotheses later. Based on this label $l_{t,h}$, we constrain the text token probabilities $p_{t,h}$ and expand the hypothesis by every possible next token. Then, we compute the score of the expanded hypotheses by summing the log probabilities.

We do this for each of the hypotheses in the beam and keep the `beam_width` top-scoring expanded hypotheses for the next timestep. Once we close a hypothesis in the beam, i.e., predict the end of sequence token, we reduce the beam width. Therefore, we end up with `beam_width` closed hypotheses. We choose the top-scoring one as the prediction.

## 4.4 Experimental Results

In this section, we present the results of the proposed architecture variants. First, we explain the metrics we utilize. Next, we describe the results achieved on the development set. We select the best architectures based on these results and report their performance on the test set. Finally, we conduct an error analysis and discuss the findings.

### 4.4.1 Metrics

As an intermediate metric to monitor the model training, we use an ad-hoc metric dubbed *token slot label F1 (TSL-F1)*. This is a straightforward metric, where we simply compute the F1-Score across slot labels for all tokens (not considering whether the recognized tokens, i.e., slot values, match the reference). For simplicity, we also include the no-slot class (O) in this metric. We use a macro average of F1 scores for all classes. Therefore, the prevailing no-slot class cannot completely outweigh the slot classes.

This metric provides perfectly matched results only when paired with teacher forcing. During autoregressive inference, the model may predict a slightly different transcription that has a different length when represented as a sequence of Whisper tokens. This can introduce a shift in the sequence against the gold label. Then, even though the prediction may be completely valid, TSL-F1 will be low. Conversely, it may overestimate the performance when the sequence is approximately correctly aligned, but the decoded tokens are wrong.

We employ *slot F1 (Slot-F1)* as the final representative metric (discussed in Section 2.1). This metric works with the slot values detected in the utterance. We consider a slot value correct only when the lemma is exactly equal to the gold lemma and the predicted slot type is correct. We report both the micro-averaged and the macro-averaged Slot-F1 variant over slot types described in Section 4.1.1.

### 4.4.2 Results

The F1-score results on the development set of our different proposed architectures and the baseline are summarized in Table 4.2. The names correspond to the architecture presentation in Section 4.2. "Sampling" denotes that pseudo scheduled sampling was used to train the model, "BS" signifies beam search of size 3 during inference (details of these techniques are discussed in Section 4.3). We report the TSL-F1 of the inference mode without teacher forcing and without beam search. With teacher forcing, the models (except for the Sampling variants) ranged from 0.7 to 0.9.

| | **TSL-F1** | **micro-Slot-F1** | **macro-Slot-F1** |
|---|---|---|---|
| **Baseline-BS** | — | 0.59 | 0.58 |
| **Initial** | 0.38 | 0.11 | 0.09 |
| **Initial-BS** | — | 0.18 | 0.14 |
| **Seq-Shift** | 0.39 | 0.17 | 0.13 |
| **Seq-Shift-BS** | — | 0.24 | 0.18 |
| **Final** | 0.54 | 0.34 | 0.26 |
| **Final-BS** | — | 0.46 | 0.36 |
| **Final-Sampling-Both** | 0.50 | 0.26 | 0.19 |
| **Final-Sampling-Both-BS** | — | 0.33 | 0.25 |
| **Final-Sampling-Slot** | 0.51 | 0.32 | 0.23 |
| **Final-Sampling-Slot-BS** | — | 0.42 | 0.30 |
| **Final-NoEnc** | 0.35 | 0.27 | 0.17 |
| **Final-NoDec** | 0.48 | 0.07 | 0.05 |

**Table 4.2** F1-Score results on the development set. Model names correspond to the architecture presentation in Section 4.2. "Sampling" denotes that pseudo scheduled sampling was used to train the model, "BS" signifies beam search of size 3 during inference (both discussed in Section 4.3).

Unfortunately, none of our proposed architectures managed to surpass the baseline. As we have expected, the `Initial` version performs very poorly. We believe this is due to the sequence misalignment.

Contrary to our initial theory, providing the Slot decoder with all the Whisper's inner representations does not benefit the model. The `Final` version that does not utilize the last Whisper decoder output $WhisperDec_t$ performs substantially better than `Seq-Shift`. We theorize this happens because $SlotFFCL$ in `Seq-Shift` becomes too dependent on the last Whisper decoder output $WhisperDec_t$. Then, during autoregressive decoding, Whisper may make a mistake early on in the decoding, and the following Whisper decoder outputs $WhisperDec_i$ are noisy.

Pseudo-scheduled sampling does not bring any improvements. These models struggled to converge, and the training was very slow. Mismatched tokens due to different decoding lengths discussed in Section 4.4.1 may be part of the reason. However, similar issues may be the reason why good teacher-forcing results do not translate into a good autoregressive performance.

In ablation experiments, we find that the `Final` variant utilizes both the audio encodings and the previous text token embeddings with benefit, as the results of `Final-NoEnc` are poor. Interestingly, the TSL-F1 result of `Final-NoDec` is good and its macro-Slot-F1 is very poor. We knew the TSL-F1 metric was not conclusive, but in the other experiments, it usually aligned with the proper metrics. We hypothesize this is caused by the fact that the `Final-NoDec` has no way to align with the text tokens. Although the TSL-F1 is good, even a little misalignment results in a completely erroneous decoded output.

We chose the `Final-BS` as it got the best results and evaluated it on the test set. The test set results are summarized in Table 4.3. The results are very similar, the baseline still performs better. The macro-Slot-F1 is much closer to the micro-Slot-F1 than on the development set. We believe this is caused by the `via` slot type. The whole dataset contains only tens of examples that contain this slot type. Therefore, it is complicated for the model to learn its properties. For future experiments, we recommend either omitting it completely or evaluating it separately as a measure of few-shot learning.

| | **micro-Slot-F1** | **macro-Slot-F1** |
|---|---|---|
| **Baseline-BS** | 0.60 | 0.59 |
| **Final-BS** | 0.46 | 0.45 |

**Table 4.3** F1-Score results on the test set. Our final architecture with beam search of size 3 during inference was chosen as it achieved the best results on the development set.

### 4.4.3 Error Analysis

In this section, we analyze the errors of our best `Final-BS` architecture on the development set. We will discuss some more granular evaluation metrics and then perform a manual error assessment.

#### Granular Metrics

First, let us explore the metrics of different slot types (discussed in Section 4.1.1), as summarized in Table 4.4. As we can see, the poor result for the `via` slot type draws the macro-Slot-F1 down a lot. The scores for `from` and `to` slots are quite balanced, the result for the `unknown` slot is worse. Values of this slot usually appear in less structured sentences, which probably makes the prediction harder for the model.

| | **Precision** | **Recall** | **F1-Score** |
|---|---|---|---|
| `from` | 0.50 | 0.41 | 0.45 |
| `to` | 0.52 | 0.51 | 0.51 |
| `via` | 1.0 | 0.08 | 0.15 |
| `unknown` | 0.35 | 0.32 | 0.33 |

**Table 4.4** Detailed metrics for different slot types of the `Final-BS` on the validation set.

Next, we focus on an analysis of false positives. In total, we measure that the model produced 3536 false positive predictions. Some of these false positives were partially correct. First, we pair the true positives with the corresponding gold values and exclude these from further inspection. Then, we try to match the false positives to the remaining gold values in three ways:

- Correct lemma, wrong type: The predicted lemma exactly matches some gold lemma. The predicted slot type is wrong.
- Lemma is a proper subset, any type: The predicted lemma is a proper subset of some gold lemma. The predicted type can be right or wrong.
- Lemma is a proper superset, any type: The predicted lemma is a proper superset of some gold lemma. The predicted type can be right or wrong.

Note that we cannot directly assign the false positive prediction to just one of these classes. One prediction lemma could potentially be a subset of one gold lemma and a superset of another. However, this is not probable, as the dataset hardly ever contains more than two gold values in a single utterance. We can thus assume that the overlap among our three classes will be negligible. The overview of false positive predictions can be found in Table 4.5. When the false positive prediction is at least partially correct, it is most often a proper subset of some gold value. However, the amount of false positive predictions that are a proper superset of a gold value is also not negligible.

| | **#predictions** | **% false positives** |
|---|---|---|
| **Correct lemma, wrong type** | 156 | 4.4 % |
| **Lemma proper subset, any type** | 336 | 9.5 % |
| **Lemma proper superset, any type** | 165 | 4.7 % |

**Table 4.5** Numbers and ratios of false positive predictions. Note that the classes are not necessarily disjunct.

### Manual Error Assessment

We manually explore a sample of the model predictions and compare them to the gold labels. We identify certain classes of common mistakes and illustrate them on examples. We always provide the gold transcription, the predicted transcription, and then tokens annotated with token slot labels. We do not consider different slot types and convert IOB to IO8 for better readability.

**Early Prediction** The model predicts the start of slot value one token too early and decodes a wrong value. An example is shown in Figure 4.4.

**Hallucination** The model recognizes further text that does not exist in the recording; see an example in Figure 4.5. Usually, it is acoustically similar to what was uttered. This is a documented behavior in Whisper (Koenecke et al., 2024). It may be amplified by our added decoder, where whenever a start of slot value is predicted, it restricts Whisper from predicting an end-of-sequence token.

```
Gold: Do Brna.
Pred: Dobrná.
Gold: 0Do0 8 Br8 1na1 0.0
Pred: 8D8 1ob1 1r1 1n1 1á1 0.0
```

**Figure 4.4** An example of early predicted token slot label. The slot value decoding then leads to a wrong sequence.

```
Gold: Malostranská.
Pred: Malostranská. Mělice.
Gold: 8M8 1al1 1ost1 1rans1 1k1 1á1 0.0
Pred: 8M8 1al1 1ost1 1rans1 1k1 1á1 0.0 8 M8 1ě1 1l1 1ice1 0.0
```

**Figure 4.5** An example of hallucination in predictions. The model produces a part of the text that does not match the input recording.

**No Slot Predicted** In some cases, the slot predictor does not predict any slot value, and the decoding is unconstrained. An example is shown in Figure 4.6.

```
Gold: Rád bych jel z Kladna města
Pred: Rád bych je skladna města
Gold: 0R0 0ád0 0 by0 0ch0 0 j0 0el0 0 z0 8 K8 1lad1 1na1 0 m0 0ě0 0sta0
Pred: 0R0 0ád0 0 by0 0ch0 0 je0 0 sk0 0lad0 0na0 0 m0 0ě0 0sta0
```

**Figure 4.6** An example of a prediction where the model does not predict any value. The decoding is then not constrained.

**Correct Token Slot Label, Wrong Text** Some errors are caused purely by the speech recognition part. The Slot decoder correctly predicts the beginning of a slot value and starts constrained decoding. However, the speech recognition part assigns the highest probability to a wrong token, even though the set is already limited. We show an example in Figure 4.7.

```
Gold: Ne, Hloubětín.
Pred: Ne, loubí čtyři! Konečná stanice!
Gold: 0Ne0 0,0 8 H8 1l1 1ou1 1b1 1ě1 1t1 1ín1 0.0
Pred: 0N0 0e0 0,0 8 lou8 1b1 1í1 1 č1 1ty1 1ř1 1i1 0!0
```

**Figure 4.7** An example of a correct token slot label and wrong text in predictions. The decoding is correctly constrained, but the text token probabilities are wrong.

# Conclusion

In this thesis, we explored the task of slot filling in spoken language understanding for voice-based task-oriented dialogue systems. In the first three chapters, we summarized the necessary theory, including audio processing, machine learning methods, automatic speech recognition, textual slot filling, and existing methods for spoken slot filling.

In order to target the task, we compiled a dataset for spoken slot filling in the Czech language. To the best of our knowledge, there was no existing open-source dataset that targeted this task and language. The dataset deals with public transport search. It contains roughly 30k utterances where users ask about public transport connections. We identified four main slot types of interest, but we recommend three of them for further use. Although the provided labels are of high quality, we believe a manual inspection may be beneficial.

Our primary research question was:

> How can we utilize prior knowledge of possible slot values (e.g., having them in a database) to achieve robust slot filling in SLU with general off-the-shelf ASR (without fine-tuning the ASR model)?

In an attempt to answer it, we have proposed a novel architecture that integrates SLU by adding a new decoder layer to the Whisper pre-trained ASR. This decoder predicts labels in IOB format for individual tokens of the decoded text sequence. We then constrain the text decoding to achieve only valid slot values using the predicted IOB labels and a trie.

We have shown that even a relatively small Transformer-based decoder with roughly 27M trainable parameters (compared to the 1543M of the underlying Whisper model) can benefit from access to speech encodings. Unfortunately, the best version of our architecture did not surpass the presented baseline.

Nevertheless, this presents an opportunity for future research to improve upon these results. The baseline results are roughly 0.6 F1 score, i.e., there is a substantial gap for improvement. For future work, we suggest altering the training approach (Ruan et al., 2020) or further improving the constrained decoding (P. Wang et al., 2022).

# Bibliography

Abdel-Hamid, O.; Mohamed, A.; Jiang, H.; Deng, L.; Penn, G.; Yu, D., 2014. Convolutional Neural Networks for Speech Recognition. *IEEE ACM Trans. Audio Speech Lang. Process.* Vol. 22, no. 10, pp. 1533–1545. Available from doi: 10.1109/TASLP.2014.2339736.

Allen, J.; Rabiner, L., 1977. A unified approach to short-time Fourier analysis and synthesis. *Proceedings of the IEEE.* Vol. 65, no. 11, pp. 1558–1564. Available from doi: 10.1109/PROC.1977.10770.

Amodei, D.; Ananthanarayanan, S.; Anubhai, R.; Bai, J.; Battenberg, E.; Case, C.; Casper, J.; Catanzaro, B.; Chen, J.; Chrzanowski, M.; Coates, A.; Diamos, G.; Elsen, E.; Engel, J. H.; Fan, L.; Fougner, C.; Hannun, A. Y.; Jun, B.; Han, T.; LeGresley, P.; Li, X.; Lin, L.; Narang, S.; Ng, A. Y.; Ozair, S.; Prenger, R.; Qian, S.; Raiman, J.; Satheesh, S.; Seetapun, D.; Sengupta, S.; Wang, C.; Wang, Y.; Wang, Z.; Xiao, B.; Xie, Y.; Yogatama, D.; Zhan, J.; Zhu, Z., 2016. Deep Speech 2 : End-to-End Speech Recognition in English and Mandarin. In: Balcan, M.; Weinberger, K. Q. (eds.). *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016.* JMLR.org. Vol. 48, pp. 173–182. JMLR Workshop and Conference Proceedings. Available also from: http://proceedings.mlr.press/v48/amodei16.html.

Ardila, R.; Branson, M.; Davis, K.; Kohler, M.; Meyer, J.; Henretty, M.; Morais, R.; Saunders, L.; Tyers, F. M.; Weber, G., 2020. Common Voice: A Massively-Multilingual Speech Corpus. In: Calzolari, N.; Béchet, F.; Blache, P.; Choukri, K.; Cieri, C.; Declerck, T.; Goggi, S.; Isahara, H.; Maegaard, B.; Mariani, J.; Mazo, H.; Moreno, A.; Odijk, J.; Piperidis, S. (eds.). *Proceedings of The 12th Language Resources and Evaluation Conference, LREC 2020, Marseille, France, May 11-16, 2020.* European Language Resources Association, pp. 4218–4222. Available also from: https://aclanthology.org/2020.lrec-1.520/.

Baevski, A.; Zhou, Y.; Mohamed, A.; Auli, M., 2020. wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations. In: Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; Lin, H. (eds.). *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual.* Available also from: https://proceedings.neurips.cc/paper/2020/hash/92d1e1eb1cd6f9fba3227870bb6d7f07-Abstract.html.

Bahdanau, D.; Cho, K.; Bengio, Y., 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In: Bengio, Y.; LeCun, Y. (eds.). *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings.* Available also from: http://arxiv.org/abs/1409.0473.

Basodi, S.; Ji, C.; Zhang, H.; Pan, Y., 2020. Gradient amplification: An efficient way to train deep neural networks. *Big Data Min. Anal.* Vol. 3, no. 3, pp. 196–207. Available from doi: 10.26599/BDMA.2020.9020004.

BASTIANELLI, E.; VANZO, A.; SWIETOJANSKI, P.; RIESER, V., 2020. SLURP: A Spoken Language Understanding Resource Package. In: WEBBER, B.; COHN, T.; HE, Y.; LIU, Y. (eds.). *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*. Association for Computational Linguistics, pp. 7252–7262. Available from DOI: `10.18653/V1/2020.EMNLP-MAIN.588`.

BAUM, L. E.; PETRIE, T.; SOULES, G.; WEISS, N., 1970. A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains. *The Annals of Mathematical Statistics* [online]. Vol. 41, no. 1, pp. 164–171 [visited on 2024-06-26]. ISSN 0003-4851. Available from DOI: `10.1214/aoms/1177697196`.

BELLMAN, R.; DREYFUS, S., 2010. *Dynamic Programming*. Vol. 33 [online]. Princeton University Press [visited on 2024-07-14]. ISBN 978-0-691-14668-3. Available from DOI: `10.2307/j.ctv1nxcw0f`.

BENGIO, S.; VINYALS, O.; JAITLY, N.; SHAZEER, N., 2015. Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks. In: CORTES, C.; LAWRENCE, N. D.; LEE, D. D.; SUGIYAMA, M.; GARNETT, R. (eds.). *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pp. 1171–1179. Available also from: `https://proceedings.neurips.cc/paper/2015/hash/e995f98d56967d946471af29d7bf99f1-Abstract.html`.

BISHOP, C. M.; NASRABADI, N. M., 2006. *Pattern recognition and machine learning*. Vol. 4. Springer. Number: 4.

BODON, F.; RÓNYAI, L., 2003. Trie: an alternative data structure for data mining algorithms. *Mathematical and Computer Modelling*. Vol. 38, no. 7, pp. 739–751. Publisher: Elsevier.

BOJANOWSKI, P.; GRAVE, E.; JOULIN, A.; MIKOLOV, T., 2017. Enriching Word Vectors with Subword Information. *Trans. Assoc. Comput. Linguistics*. Vol. 5, pp. 135–146. Available from DOI: `10.1162/TACL\_A\_00051`.

BROWN, T. B.; MANN, B.; RYDER, N.; SUBBIAH, M.; KAPLAN, J.; DHARIWAL, P.; NEELAKANTAN, A.; SHYAM, P.; SASTRY, G.; ASKELL, A.; AGARWAL, S.; HERBERT-VOSS, A.; KRUEGER, G.; HENIGHAN, T.; CHILD, R.; RAMESH, A.; ZIEGLER, D. M.; WU, J.; WINTER, C.; HESSE, C.; CHEN, M.; SIGLER, E.; LITWIN, M.; GRAY, S.; CHESS, B.; CLARK, J.; BERNER, C.; MCCANDLISH, S.; RADFORD, A.; SUTSKEVER, I.; AMODEI, D., 2020. Language Models are Few-Shot Learners. In: LAROCHELLE, H.; RANZATO, M.; HADSELL, R.; BALCAN, M.; LIN, H. (eds.). *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Available also from: `https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html`.

CAO, L., 2024. *DiagGPT: An LLM-based and Multi-agent Dialogue System with Automatic Topic Management for Flexible Task-Oriented Dialogue* [online]. arXiv [visited on 2024-07-04]. No. arXiv:2308.08043. Available from arXiv: `2308.08043[cs]`.

CHAO, G.; LANE, I. R., 2019. BERT-DST: Scalable End-to-End Dialogue State Tracking with Bidirectional Encoder Representations from Transformer. In: KUBIN, G.; KACIC, Z. (eds.). *20th Annual Conference of the International Speech Communication Association, Interspeech 2019, Graz, Austria, September 15-19, 2019*. ISCA, pp. 1468–1472. Available from DOI: `10.21437/INTERSPEECH.2019-1355`.

CHEN, Q.; ZHUO, Z.; WANG, W., 2019. BERT for Joint Intent Classification and Slot Filling. *CoRR*. Vol. abs/1902.10909. Available from arXiv: `1902.10909`.

CHEN, S. F.; GOODMAN, J., 1999. An empirical study of smoothing techniques for language modeling. *Comput. Speech Lang.* Vol. 13, no. 4, pp. 359–393. Available from DOI: `10.1006/CSLA.1999.0128`.

CHO, K.; MERRIENBOER, B. van; GÜLÇEHRE, Ç.; BAHDANAU, D.; BOUGARES, F.; SCHWENK, H.; BENGIO, Y., 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In: MOSCHITTI, A.; PANG, B.; DAELEMANS, W. (eds.). *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL, pp. 1724–1734. Available from DOI: `10.3115/V1/D14-1179`.

CHOROWSKI, J.; BAHDANAU, D.; CHO, K.; BENGIO, Y., 2014. End-to-end Continuous Speech Recognition using Attention-based Recurrent NN: First Results. *CoRR*. Vol. abs/1412.1602. Available from arXiv: `1412.1602`.

CHUNG, J.; GÜLÇEHRE, Ç.; CHO, K.; BENGIO, Y., 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *CoRR*. Vol. abs/1412.3555. Available from arXiv: `1412.3555`.

DAVIS, S.; MERMELSTEIN, P., 1980. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*. Vol. 28, no. 4, pp. 357–366. Available from DOI: `10.1109/TASSP.1980.1163420`.

DENISOV, P.; VU, N. T., 2020. Pretrained Semantic Speech Embeddings for End-to-End Spoken Language Understanding via Cross-Modal Teacher-Student Learning. In: MENG, H.; XU, B.; ZHENG, T. F. (eds.). *21st Annual Conference of the International Speech Communication Association, Interspeech 2020, Virtual Event, Shanghai, China, October 25-29, 2020*. ISCA, pp. 881–885. Available from DOI: `10.21437/INTERSPEECH.2020-2456`.

DESOT, T.; PORTET, F.; VACHER, M., 2022. End-to-End Spoken Language Understanding: Performance analyses of a voice command task in a low resource setting. *Comput. Speech Lang.* Vol. 75, p. 101369. Available from DOI: `10.1016/J.CSL.2022.101369`.

DEVLIN, J.; CHANG, M.; LEE, K.; TOUTANOVA, K., 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: BURSTEIN, J.; DORAN, C.; SOLORIO, T. (eds.). *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, pp. 4171–4186. Available from DOI: `10.18653/V1/N19-1423`.

Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; Uszkoreit, J.; Houlsby, N., 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. Available also from: `https://openreview.net/forum?id=YicbFdNTTy`.

Drugman, T.; Pylkkönen, J.; Kneser, R., 2016. Active and Semi-Supervised Learning in ASR: Benefits on the Acoustic and Language Models. In: Morgan, N. (ed.). *17th Annual Conference of the International Speech Communication Association, Interspeech 2016, San Francisco, CA, USA, September 8-12, 2016*. ISCA, pp. 2318–2322. Available from doi: `10.21437/INTERSPEECH.2016-1382`.

Du, K.-L.; Swamy, M., 2013. *Neural networks and statistical learning*. isbn 978-1-4471-5570-6. Available from doi: `10.1007/978-1-4471-5571-3`.

Dusek, O.; Plátek, O.; Zilka, L.; Jurcícek, F., 2014. Alex: Bootstrapping a Spoken Dialogue System for a New Domain by Real Users. In: *Proceedings of the SIGDIAL 2014 Conference, The 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue, 18-20 June 2014, Philadelphia, PA, USA*. The Association for Computer Linguistics, pp. 79–83. Available from doi: `10.3115/V1/W14-4311`.

Fang, A.; Filice, S.; Limsopatham, N.; Rokhlenko, O., 2020. Using Phoneme Representations to Build Predictive Models Robust to ASR Errors. In: Huang, J. X.; Chang, Y.; Cheng, X.; Kamps, J.; Murdock, V.; Wen, J.; Liu, Y. (eds.). *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*. ACM, pp. 699–708. Available from doi: `10.1145/3397271.3401050`.

Feng, L.; Yu, J.; Cai, D.; Liu, S.; Zheng, H.; Wang, Y., 2021. ASR-GLUE: A New Multi-task Benchmark for ASR-Robust Natural Language Understanding. *CoRR*. Vol. abs/2108.13048. Available from arXiv: `2108.13048`.

Filice, S.; Castellucci, G.; Collins, M. D.; Agichtein, E.; Rokhlenko, O., 2021. VoiSeR: A New Benchmark for Voice-Based Search Refinement. In: Merlo, P.; Tiedemann, J.; Tsarfaty, R. (eds.). *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, EACL 2021, Online, April 19 - 23, 2021*. Association for Computational Linguistics, pp. 2321–2329. Available from doi: `10.18653/V1/2021.EACL-MAIN.197`.

Gage, P., 1994. A new algorithm for data compression. *The C Users Journal archive*. Vol. 12, pp. 23–38. Available also from: `https://api.semanticscholar.org/CorpusID:59804030`.

Gaido, M.; Tang, Y.; Kulikov, I.; Huang, R.; Gong, H.; Inaguma, H., 2023. Named Entity Detection and Injection for Direct Speech Translation. In: *IEEE International Conference on Acoustics, Speech and Signal Processing ICASSP 2023, Rhodes Island, Greece, June 4-10, 2023*. IEEE, pp. 1–5. Available from doi: `10.1109/ICASSP49357.2023.10094689`.

GAUVAIN, J.; LEE, C., 1994. Maximum a posteriori estimation for multivariate Gaussian mixture observations of Markov chains. *IEEE Trans. Speech Audio Process.* Vol. 2, no. 2, pp. 291–298. Available from DOI: 10.1109/89.279278.

GEHRING, J.; AULI, M.; GRANGIER, D.; YARATS, D.; DAUPHIN, Y. N., 2017. Convolutional Sequence to Sequence Learning. In: PRECUP, D.; TEH, Y. W. (eds.). *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017.* PMLR. Vol. 70, pp. 1243–1252. Proceedings of Machine Learning Research. Available also from: http://proceedings.mlr.press/v70/gehring17a.html.

GOLD, B.; MORGAN, N.; ELLIS, D., 2011. *Speech and audio signal processing: Processing and perception of speech and music.* Wiley. ISBN 978-0-470-19536-9. Available also from: https://books.google.cz/books?id=M1TM8-GA_YkC. tex.lccn: 2011293787.

GOO, C.; GAO, G.; HSU, Y.; HUO, C.; CHEN, T.; HSU, K.; CHEN, Y., 2018. Slot-Gated Modeling for Joint Slot Filling and Intent Prediction. In: WALKER, M. A.; JI, H.; STENT, A. (eds.). *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 2 (Short Papers).* Association for Computational Linguistics, pp. 753–757. Available from DOI: 10.18653/V1/N18-2118.

GOODFELLOW, I. J.; BENGIO, Y.; COURVILLE, A. C., 2016. *Deep Learning.* MIT Press. Adaptive computation and machine learning. ISBN 978-0-262-03561-3. Available also from: http://www.deeplearningbook.org/.

GRAVES, A.; FERNÁNDEZ, S.; GOMEZ, F. J.; SCHMIDHUBER, J., 2006. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In: COHEN, W. W.; MOORE, A. W. (eds.). *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006.* ACM. Vol. 148, pp. 369–376. ACM International Conference Proceeding Series. Available from DOI: 10.1145/1143844.1143891.

GRAVES, A.; MOHAMED, A.; HINTON, G. E., 2013. Speech recognition with deep recurrent neural networks. In: *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013.* IEEE, pp. 6645–6649. Available from DOI: 10.1109/ICASSP.2013.6638947.

GRAVES, A.; SCHMIDHUBER, J., 2005. Framewise phoneme classification with bidirectional LSTM networks. *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* Vol. 4, 2047–2052 vol. 4. Available also from: https://api.semanticscholar.org/CorpusID:9594328.

GULATI, A.; QIN, J.; CHIU, C.; PARMAR, N.; ZHANG, Y.; YU, J.; HAN, W.; WANG, S.; ZHANG, Z.; WU, Y.; PANG, R., 2020. Conformer: Convolution-augmented Transformer for Speech Recognition. In: MENG, H.; XU, B.; ZHENG, T. F. (eds.). *21st Annual Conference of the International Speech Communication Association, Interspeech 2020, Virtual Event, Shanghai, China, October 25-29, 2020.* ISCA, pp. 5036–5040. Available from DOI: 10.21437/INTERSPEECH.2020-3015.

Hannun, A., 2017. Sequence modeling with CTC. *Distill*. Available from doi: `10.23915/distill.00008`.

Hastie, T.; Friedman, J. H.; Tibshirani, R., 2001. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer. Springer Series in Statistics. isbn 978-1-4899-0519-2. Available from doi: `10.1007/978-0-387-21606-5`.

He, K.; Zhang, X.; Ren, S.; Sun, J., 2016. Deep Residual Learning for Image Recognition. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, pp. 770–778. Available from doi: `10.1109/CVPR.2016.90`.

He, M.; Garner, P. N., 2023. Can ChatGPT Detect Intent? Evaluating Large Language Models for Spoken Language Understanding. In: Harte, N.; Carson-Berndsen, J.; Jones, G. (eds.). *24th Annual Conference of the International Speech Communication Association, Interspeech 2023, Dublin, Ireland, August 20-24, 2023*. ISCA, pp. 1109–1113. Available from doi: `10.21437/INTERSPEECH.2023-1799`.

He, Y.; Young, S. J., 2005. Semantic processing using the Hidden Vector State model. *Comput. Speech Lang.* Vol. 19, no. 1, pp. 85–106. Available from doi: `10.1016/J.CSL.2004.03.001`.

Heck, M.; Lubis, N.; Ruppik, B. M.; Vukovic, R.; Feng, S.; Geishauser, C.; Lin, H.; Niekerk, C. van; Gasic, M., 2023. ChatGPT for Zero-shot Dialogue State Tracking: A Solution or an Opportunity? In: Rogers, A.; Boyd-Graber, J. L.; Okazaki, N. (eds.). *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*. Association for Computational Linguistics, pp. 936–950. Available from doi: `10.18653/V1/2023.ACL-SHORT.81`.

Heideman, M.; Johnson, D.; Burrus, C., 1984. Gauss and the history of the fast fourier transform. *IEEE ASSP Magazine* [online]. Vol. 1, no. 4, pp. 14–21 [visited on 2024-06-24]. issn 0740-7467. Available from doi: `10.1109/MASSP.1984.1162257`.

Hemphill, C. T.; Godfrey, J. J.; Doddington, G. R., 1990. The ATIS Spoken Language Systems Pilot Corpus. In: *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, USA, June 24-27, 1990*. Morgan Kaufmann. Available also from: `https://aclanthology.org/H90-1021/`.

Henderson, M.; Thomson, B.; Williams, J. D., 2014. The Second Dialog State Tracking Challenge. In: *Proceedings of the SIGDIAL 2014 Conference, The 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue, 18-20 June 2014, Philadelphia, PA, USA*. The Association for Computer Linguistics, pp. 263–272. Available from doi: `10.3115/V1/W14-4337`.

Hinton, G.; Deng, L.; Yu, D.; Dahl, G.; Mohamed, A.-r.; Jaitly, N.; Senior, A.; Vanhoucke, V.; Nguyen, P.; Sainath, T.; Kingsbury, B., 2012. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine* [online]. Vol. 29, no. 6, pp. 82–97 [visited on 2024-06-27]. issn 1053-5888. Available from doi: 10.1109/MSP.2012.2205597.

Hochreiter, S., 1998. The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. *Int. J. Uncertain. Fuzziness Knowl. Based Syst.* Vol. 6, no. 2, pp. 107–116. Available from doi: 10.1142/S0218488598000094.

Hochreiter, S.; Schmidhuber, J., 1997. Long Short-Term Memory. *Neural Comput.* Vol. 9, no. 8, pp. 1735–1780. Available from doi: 10.1162/NECO.1997.9.8.1735.

Hokamp, C.; Liu, Q., 2017. Lexically Constrained Decoding for Sequence Generation Using Grid Beam Search. In: Barzilay, R.; Kan, M. (eds.). *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers.* Association for Computational Linguistics, pp. 1535–1546. Available from doi: 10.18653/V1/P17-1141.

Hono, Y.; Mitsuda, K.; Zhao, T.; Mitsui, K.; Wakatsuki, T.; Sawada, K., 2024. *Integrating Pre-Trained Speech and Language Models for End-to-End Speech Recognition* [online]. arXiv [visited on 2024-07-17]. No. arXiv:2312.03668. Available from doi: 10.48550/arXiv.2312.03668.

Hopfield, J. J., 1982. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences.* Vol. 79, no. 8, pp. 2554–2558. Available from doi: 10.1073/pnas.79.8.2554. tex.eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.79.8.2554.

Hossan, M. A.; Memon, S.; Gregory, M. A., 2010. A novel approach for MFCC feature extraction. In: *2010 4th International Conference on Signal Processing and Communication Systems* [online]. Gold Coast, Australia: IEEE, pp. 1–5 [visited on 2024-06-24]. isbn 978-1-4244-7908-5. Available from doi: 10.1109/ICSPCS.2010.5709752.

Huang, C.; Chen, Y., 2020a. Learning Asr-Robust Contextualized Embeddings for Spoken Language Understanding. In: *2020 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2020, Barcelona, Spain, May 4-8, 2020.* IEEE, pp. 8009–8013. Available from doi: 10.1109/ICASSP40776.2020.9054689.

Huang, C.; Chen, Y., 2020b. Learning Spoken Language Representations with Neural Lattice Language Modeling. In: Jurafsky, D.; Chai, J.; Schluter, N.; Tetreault, J. R. (eds.). *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020.* Association for Computational Linguistics, pp. 3764–3769. Available from doi: 10.18653/V1/2020.ACL-MAIN.347.

Huang, Z.; Xu, W.; Yu, K., 2015. Bidirectional LSTM-CRF Models for Sequence Tagging. *CoRR.* Vol. abs/1508.01991. Available from arXiv: 1508.01991.

Hudecek, V.; Dusek, O., 2023. Are LLMs All You Need for Task-Oriented Dialogue? *CoRR*. Vol. abs/2304.06556. Available from doi: `10.48550/ARXIV.2304.06556`.

Jelinek, F., 1998. *Statistical methods for speech recognition*. MIT Press. Language, speech, and communication. isbn 978-0-262-10066-3. Available also from: `https://books.google.cz/books?id=1C9dzcJTWowC`. tex.lccn: 97034860.

Jurafsky, D.; Martin, J., 2024. *Speech and language processing*. 3rd Edition Draft. Available also from: `https://web.stanford.edu/~jurafsky/slp3/`.

Kim, M.; Kim, G.; Lee, S.; Ha, J., 2021. St-Bert: Cross-Modal Language Model Pre-Training for End-to-End Spoken Language Understanding. In: *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2021, Toronto, ON, Canada, June 6-11, 2021*. IEEE, pp. 7478–7482. Available from doi: `10.1109/ICASSP39728.2021.9414558`.

Kocmi, T.; Bojar, O., 2016. SubGram: Extending Skip-Gram Word Representation with Substrings. In: Sojka, P.; Horák, A.; Kopecek, I.; Pala, K. (eds.). *Text, Speech, and Dialogue - 19th International Conference, TSD 2016, Brno, Czech Republic, September 12-16, 2016, Proceedings*. Springer. Vol. 9924, pp. 182–189. Lecture Notes in Computer Science. Available from doi: `10.1007/978-3-319-45510-5\_21`.

Koenecke, A.; Choi, A. S. G.; Mei, K. X.; Schellmann, H.; Sloane, M., 2024. Careless Whisper: Speech-to-text hallucination harms. In: *Proceedings of the 2024 ACM conference on fairness, accountability, and transparency*. New York, NY, USA: Association for Computing Machinery, pp. 1672–1681. FAccT '24. isbn 9798400704505. Available from doi: `10.1145/3630106.3658996`. Number of pages: 10 Place: Rio de Janeiro, Brazil.

Ladhak, F.; Gandhe, A.; Dreyer, M.; Mathias, L.; Rastrow, A.; Hoffmeister, B., 2016. LatticeRnn: Recurrent Neural Networks Over Lattices. In: Morgan, N. (ed.). *17th Annual Conference of the International Speech Communication Association, Interspeech 2016, San Francisco, CA, USA, September 8-12, 2016*. ISCA, pp. 695–699. Available from doi: `10.21437/INTERSPEECH.2016-1583`.

Lafferty, J. D.; McCallum, A.; Pereira, F. C. N., 2001. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In: Brodley, C. E.; Danyluk, A. P. (eds.). *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001), Williams College, Williamstown, MA, USA, June 28 - July 1, 2001*. Morgan Kaufmann, pp. 282–289.

Levenshtein, V. I., 1965. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics. Doklady*. Vol. 10, pp. 707–710. Available also from: `https://api.semanticscholar.org/CorpusID:60827152`.

Li, M.; Ruan, W.; Liu, X.; Soldaini, L.; Hamza, W.; Su, C., 2020. Improving Spoken Language Understanding By Exploiting ASR N-best Hypotheses. *CoRR*. Vol. abs/2001.05284. Available from arXiv: `2001.05284`.

Li, X.; Wang, L.; Dong, G.; He, K.; Zhao, J.; Lei, H.; Liu, J.; Xu, W., 2023. Generative Zero-Shot Prompt Learning for Cross-Domain Slot Filling with Inverse Prompting. In: Rogers, A.; Boyd-Graber, J. L.; Okazaki, N. (eds.). *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023.* Association for Computational Linguistics, pp. 825–834. Available from doi: `10.18653/V1/2023.FINDINGS-ACL.52`.

Liu, B.; Lane, I. R., 2016. Attention-Based Recurrent Neural Network Models for Joint Intent Detection and Slot Filling. In: Morgan, N. (ed.). *17th Annual Conference of the International Speech Communication Association, Interspeech 2016, San Francisco, CA, USA, September 8-12, 2016.* ISCA, pp. 685–689. Available from doi: `10.21437/INTERSPEECH.2016-1352`.

Liu, X.; Zhang, F.; Hou, Z.; Mian, L.; Wang, Z.; Zhang, J.; Tang, J., 2023. Self-Supervised Learning: Generative or Contrastive. *IEEE Trans. Knowl. Data Eng.* Vol. 35, no. 1, pp. 857–876. Available from doi: `10.1109/TKDE.2021.3090866`.

Liu, X.; Li, M.; Chen, L.; Wanigasekara, P.; Ruan, W.; Khan, H.; Hamza, W.; Su, C., 2021. ASR N-Best Fusion Nets. In: *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2021, Toronto, ON, Canada, June 6-11, 2021.* IEEE, pp. 7618–7622. Available from doi: `10.1109/ICASSP39728.2021.9414806`.

Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; Stoyanov, V., 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR.* Vol. abs/1907.11692. Available from arXiv: `1907.11692`.

Louvan, S.; Magnini, B., 2020. Recent Neural Methods on Slot Filling and Intent Classification for Task-Oriented Dialogue Systems: A Survey. In: Scott, D.; Bel, N.; Zong, C. (eds.). *Proceedings of the 28th International Conference on Computational Linguistics, COLING 2020, Barcelona, Spain (Online), December 8-13, 2020.* International Committee on Computational Linguistics, pp. 480–496. Available from doi: `10.18653/V1/2020.COLING-MAIN.42`.

Lugosch, L.; Ravanelli, M.; Ignoto, P.; Tomar, V. S.; Bengio, Y., 2019. Speech Model Pre-Training for End-to-End Spoken Language Understanding. In: Kubin, G.; Kacic, Z. (eds.). *20th Annual Conference of the International Speech Communication Association, Interspeech 2019, Graz, Austria, September 15-19, 2019.* ISCA, pp. 814–818. Available from doi: `10.21437/INTERSPEECH.2019-2396`.

Manning, C.; Schutze, H., 1999. *Foundations of statistical natural language processing.* MIT Press. Foundations of statistical natural language processing. isbn 978-0-262-13360-9. Available also from: `https://books.google.cz/books?id=YiFDxbEX3SUC`. tex.lccn: 99021137.

McCulloch, W. S.; Pitts, W. H., 1990. A Logical Calculus of the Ideas Immanent in Nervous Activity. In: Boden, M. A. (ed.). *The Philosophy of Artificial Intelligence.* Oxford University Press, pp. 22–39. Oxford readings in philosophy.

Mikolov, T.; Chen, K.; Corrado, G.; Dean, J., 2013. Efficient Estimation of Word Representations in Vector Space. In: Bengio, Y.; LeCun, Y. (eds.). *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*. Available also from: `http://arxiv.org/abs/1301.3781`.

Morbini, F.; Audhkhasi, K.; Artstein, R.; Segbroeck, M. V.; Sagae, K.; Georgiou, P. G.; Traum, D. R.; Narayanan, S. S., 2012. A reranking approach for recognition and classification of speech input in conversational dialogue systems. In: *2012 IEEE Spoken Language Technology Workshop (SLT), Miami, FL, USA, December 2-5, 2012*. IEEE, pp. 49–54. Available from doi: `10.1109/SLT.2012.6424196`.

Murphy, K. P., 2012. *Machine learning: a probabilistic perspective*. MIT press.

Nyquist, H., 2002. Certain topics in telegraph transmission theory. *Proc. IEEE*. Vol. 90, no. 2, pp. 280–305. Available from doi: `10.1109/5.989875`.

Ogawa, A.; Delcroix, M.; Karita, S.; Nakatani, T., 2018. Rescoring N-Best Speech Recognition List Based on One-on-One Hypothesis Comparison Using Encoder-Classifier Model. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2018, Calgary, AB, Canada, April 15-20, 2018*. IEEE, pp. 6099–6103. Available from doi: `10.1109/ICASSP.2018.8461405`.

Ogawa, A.; Delcroix, M.; Karita, S.; Nakatani, T., 2019. Improved Deep Duel Model for Rescoring N-Best Speech Recognition List Using Backward LSTMLM and Ensemble Encoders. In: Kubin, G.; Kacic, Z. (eds.). *20th Annual Conference of the International Speech Communication Association, Interspeech 2019, Graz, Austria, September 15-19, 2019*. ISCA, pp. 3900–3904. Available from doi: `10.21437/INTERSPEECH.2019-1949`.

Oppenheim, A., 1999. *Discrete-time signal processing*. Pearson Education. Pearson education signal processing series. isbn 978-81-317-0492-9. Available also from: `https://books.google.cz/books?id=geTn5W47KEsC`.

Panayotov, V.; Chen, G.; Povey, D.; Khudanpur, S., 2015. Librispeech: An ASR corpus based on public domain audio books. In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2015, South Brisbane, Queensland, Australia, April 19-24, 2015*. IEEE, pp. 5206–5210. Available from doi: `10.1109/ICASSP.2015.7178964`.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Köpf, A.; Yang, E. Z.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; Chintala, S., 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Wallach, H. M.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E. B.; Garnett, R. (eds.). *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 8024–8035. Available also from: `https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html`.

Pennington, J.; Socher, R.; Manning, C. D., 2014. Glove: Global Vectors for Word Representation. In: Moschitti, A.; Pang, B.; Daelemans, W. (eds.). *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL, pp. 1532–1543. Available from DOI: 10.3115/V1/D14-1162.

Peris, C.; Oz, G.; Abboud, K.; Varada, V. sai; Wanigasekara, P.; Khan, H., 2020. Using multiple ASR hypotheses to boost i18n NLU performance. In: Bhattacharyya, P.; Sharma, D. M.; Sangal, R. (eds.). *Proceedings of the 17th International Conference on Natural Language Processing, ICON 2020, Indian Institute of Technology Patna, Patna, India, December 18-21, 2020*. NLP Association of India (NLPAI), pp. 30–39. Available also from: https://aclanthology.org/2020.icon-main.5.

Peters, M. E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; Zettlemoyer, L., 2018. Deep Contextualized Word Representations. In: Walker, M. A.; Ji, H.; Stent, A. (eds.). *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*. Association for Computational Linguistics, pp. 2227–2237. Available from DOI: 10.18653/V1/N18-1202.

Plátek, O.; Dušek, O.; Jurčíček, F., 2016. *Vystadial 2016 – czech data*. Available also from: http://hdl.handle.net/11234/1-1740. tex.copyright: Creative Commons - Attribution-ShareAlike 4.0 International (CC BY-SA 4.0).

Rabiner, L., 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE* [online]. Vol. 77, no. 2, pp. 257–286 [visited on 2024-06-26]. ISSN 00189219. Available from DOI: 10.1109/5.18626.

Radford, A.; Kim, J. W.; Xu, T.; Brockman, G.; McLeavey, C.; Sutskever, I., 2023. Robust Speech Recognition via Large-Scale Weak Supervision. In: Krause, A.; Brunskill, E.; Cho, K.; Engelhardt, B.; Sabato, S.; Scarlett, J. (eds.). *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*. PMLR. Vol. 202, pp. 28492–28518. Proceedings of Machine Learning Research. Available also from: https://proceedings.mlr.press/v202/radford23a.html.

Radford, A.; Narasimhan, K., 2018. Improving language understanding by generative pre-training. In: available also from: https://api.semanticscholar.org/CorpusID:49313245.

Ramshaw, L. A.; Marcus, M., 1995. Text Chunking using Transformation-Based Learning. In: Yarowsky, D.; Church, K. (eds.). *Third Workshop on Very Large Corpora, VLC at ACL 1995, Cambridge, Massachusetts, USA, June 30, 1995*. Available also from: https://aclanthology.org/W95-0107/.

Robbins, H.; Monro, S., 1951. A stochastic approximation method. *The Annals of Mathematical Statistics* [online]. Vol. 22, no. 3, pp. 400–407 [visited on 2024-06-27]. ISSN 00034851. Available from: http://www.jstor.org/stable/2236626. Publisher: Institute of Mathematical Statistics.

Roodschild, M.; Gotay Sardiñas, J.; Will, A., 2020. A new approach for the vanishing gradient problem on sigmoid activation. *Progress in Artificial Intelligence*. Vol. 9, no. 4, pp. 351–360. Publisher: Springer.

Rosenblatt, F., 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*. Vol. 65, no. 6, pp. 386–408. issn 1939-1471(Electronic), issn 0033-295X(Print). Available from doi: `10.1037/h0042519`. Place: US Publisher: American Psychological Association.

Ruan, W.; Nechaev, Y.; Chen, L.; Su, C.; Kiss, I., 2020. Towards an ASR Error Robust Spoken Language Understanding System. In: Meng, H.; Xu, B.; Zheng, T. F. (eds.). *21st Annual Conference of the International Speech Communication Association, Interspeech 2020, Virtual Event, Shanghai, China, October 25-29, 2020*. ISCA, pp. 901–905. Available from doi: `10.21437/INTERSPEECH.2020-2844`.

Rumelhart, D. E.; Hinton, G. E.; Williams, R. J., 1986. Learning representations by back-propagating errors. *Nature*. Vol. 323, pp. 533–536. Available also from: `https://api.semanticscholar.org/CorpusID:205001834`.

Russell, S.; Norvig, P., 2021. *Artificial Intelligence, Global Edition : A Modern Approach*. Pearson Deutschland. isbn ISBN 9781292401133. Available also from: `https://elibrary.pearson.de/book/99.150005/9781292401171`.

Schumann, R.; Angkititrakul, P., 2018. Incorporating ASR Errors with Attention-Based, Jointly Trained RNN for Intent Detection and Slot Filling. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2018, Calgary, AB, Canada, April 15-20, 2018*. IEEE, pp. 6059–6063. Available from doi: `10.1109/ICASSP.2018.8461598`.

Sennrich, R.; Haddow, B.; Birch, A., 2016. Neural Machine Translation of Rare Words with Subword Units. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics. Available from doi: `10.18653/V1/P16-1162`.

Serdyuk, D.; Wang, Y.; Fuegen, C.; Kumar, A.; Liu, B.; Bengio, Y., 2018. Towards End-to-end Spoken Language Understanding. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2018, Calgary, AB, Canada, April 15-20, 2018*. IEEE, pp. 5754–5758. Available from doi: `10.1109/ICASSP.2018.8461785`.

Shivakumar, P. G.; Yang, M.; Georgiou, P. G., 2019. Spoken Language Intent Detection Using Confusion2Vec. In: Kubin, G.; Kacic, Z. (eds.). *20th Annual Conference of the International Speech Communication Association, Interspeech 2019, Graz, Austria, September 15-19, 2019*. ISCA, pp. 819–823. Available from doi: `10.21437/INTERSPEECH.2019-2226`.

Siegel, R. J., 1965. A Replication of the Mel Scale of Pitch. *The American Journal of Psychology* [online]. Vol. 78, no. 4, pp. 615–620 [visited on 2024-07-04]. issn 00029556. Available from doi: `10.2307/1420924`. Publisher: University of Illinois Press.

STEVENS, S. S.; VOLKMANN, J., 1940. The relation of pitch to frequency: a revised scale. *The American Journal of Psychology*. Vol. 53, pp. 329–353. ISSN 1939-8298(Electronic), ISSN 0002-9556(Print). Available from DOI: 10.2307/1417526. Place: US Publisher: Univ of Illinois Press.

STEVENS, S. S.; VOLKMANN, J.; NEWMAN, E. B., 1937. A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America*. Vol. 8, no. 3, pp. 185–190. ISSN 0001-4966. Available from DOI: 10.1121/1.1915893. tex.eprint: https://pubs.aip.org/asa/jasa/article-pdf/8/3/185/18723823/185_1_online.pdf.

STRAKOVÁ, J.; STRAKA, M.; HAJIC, J., 2014. Open-Source Tools for Morphology, Lemmatization, POS Tagging and Named Entity Recognition. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, System Demonstrations*. The Association for Computer Linguistics, pp. 13–18. Available from DOI: 10.3115/V1/P14-5003.

SUN, G.; ZHANG, C.; VULIC, I.; BUDZIANOWSKI, P.; WOODLAND, P. C., 2023. Knowledge-Aware Audio-Grounded Generative Slot Filling for Limited Annotated Data. *CoRR*. Vol. abs/2307.01764. Available from DOI: 10.48550/ARXIV.2307.01764.

SUN, G.; ZHANG, C.; WOODLAND, P. C., 2023. End-to-End Spoken Language Understanding with Tree-Constrained Pointer Generator. In: *IEEE International Conference on Acoustics, Speech and Signal Processing ICASSP 2023, Rhodes Island, Greece, June 4-10, 2023*. IEEE, pp. 1–5. Available from DOI: 10.1109/ICASSP49357.2023.10096541.

SUNDARARAMAN, M. N.; KUMAR, A.; VEPA, J., 2021. Phoneme-BERT: Joint Language Modelling of Phoneme Sequence and ASR Transcript. *CoRR*. Available from arXiv: 2102.00804.

SUTSKEVER, I.; VINYALS, O.; LE, Q. V., 2014. Sequence to Sequence Learning with Neural Networks. In: GHAHRAMANI, Z.; WELLING, M.; CORTES, C.; LAWRENCE, N. D.; WEINBERGER, K. Q. (eds.). *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pp. 3104–3112. Available also from: https://proceedings.neurips.cc/paper/2014/hash/a14ac55a4f27472c5d894ec1c3c743d2-Abstract.html.

TAM, Y.; LEI, Y.; ZHENG, J.; WANG, W., 2014. ASR error detection using recurrent neural network language model and complementary ASR. In: *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2014, Florence, Italy, May 4-9, 2014*. IEEE, pp. 2312–2316. Available from DOI: 10.1109/ICASSP.2014.6854012.

TUR, G.; DE MORI, R., 2011. *Spoken language understanding: Systems for extracting semantic information from speech*. John Wiley & Sons.

TÜR, G.; DEORAS, A.; HAKKANI-TÜR, D., 2013. Semantic parsing using word confusion networks with conditional random fields. In: BIMBOT, F.; CERISARA, C.; FOUGERON, C.; GRAVIER, G.; LAMEL, L.; PELLEGRINO, F.; PERRIER, P. (eds.). *14th Annual Conference of the International Speech Communication Association, INTERSPEECH 2013, Lyon, France, August 25-29, 2013*. ISCA, pp. 2579–2583. Available from DOI: 10.21437/INTERSPEECH.2013-580.

TÜR, G.; HAKKANI-TÜR, D.; HECK, L. P., 2010. What is left to be understood in ATIS? In: HAKKANI-TÜR, D.; OSTENDORF, M. (eds.). *2010 IEEE Spoken Language Technology Workshop, SLT 2010, Berkeley, California, USA, December 12-15, 2010*. IEEE, pp. 19–24. Available from DOI: 10.1109/SLT.2010.5700816.

VASWANI, A.; SHAZEER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L.; GOMEZ, A. N.; KAISER, L.; POLOSUKHIN, I., 2017. Attention is All you Need. In: GUYON, I.; LUXBURG, U. von; BENGIO, S.; WALLACH, H. M.; FERGUS, R.; VISHWANATHAN, S. V. N.; GARNETT, R. (eds.). *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5998–6008. Available also from: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

VITERBI, A., 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory* [online]. Vol. 13, no. 2, pp. 260–269 [visited on 2024-06-26]. ISSN 0018-9448, ISSN 1557-9654. Available from DOI: 10.1109/TIT.1967.1054010.

WANG, C.; RIVIÈRE, M.; LEE, A.; WU, A.; TALNIKAR, C.; HAZIZA, D.; WILLIAMSON, M.; PINO, J. M.; DUPOUX, E., 2021. VoxPopuli: A Large-Scale Multilingual Speech Corpus for Representation Learning, Semi-Supervised Learning and Interpretation. In: ZONG, C.; XIA, F.; LI, W.; NAVIGLI, R. (eds.). *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*. Association for Computational Linguistics, pp. 993–1003. Available from DOI: 10.18653/V1/2021.ACL-LONG.80.

WANG, P.; SU, Y.; ZHOU, X.; YE, X.; WEI, L.; LIU, M.; YOU, Y.; JIANG, F., 2022. Speech2Slot: A Limited Generation Framework with Boundary Detection for Slot Filling from Speech. In: KO, H.; HANSEN, J. H. L. (eds.). *23rd Annual Conference of the International Speech Communication Association, Interspeech 2022, Incheon, Korea, September 18-22, 2022*. ISCA, pp. 2748–2752. Available from DOI: 10.21437/INTERSPEECH.2022-11347.

WANG, P.; WEI, L.; CAO, Y.; XIE, J.; NIE, Z., 2020. Large-Scale Unsupervised Pre-Training for End-to-End Spoken Language Understanding. In: *2020 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2020, Barcelona, Spain, May 4-8, 2020*. IEEE, pp. 7999–8003. Available from DOI: 10.1109/ICASSP40776.2020.9053163.

Wang, Z.; Le, Y.; Zhu, Y.; Zhao, Y.; Feng, M.; Chen, M.; He, X., 2022. Building Robust Spoken Language Understanding by Cross Attention Between Phoneme Sequence and ASR Hypothesis. In: *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2022, Virtual and Singapore, 23-27 May 2022*. IEEE, pp. 7147–7151. Available from doi: 10.1109/ICASSP43922.2022.9747198.

Weld, H.; Huang, X.; Long, S.; Poon, J.; Han, S. C., 2023. A Survey of Joint Intent Detection and Slot Filling Models in Natural Language Understanding. *ACM Comput. Surv.* Vol. 55, no. 8, 156:1–156:38. Available from doi: 10.1145/3547138.

Weng, Y.; Miryala, S. S.; Khatri, C.; Wang, R.; Zheng, H.; Molino, P.; Namazifar, M.; Papangelis, A.; Williams, H.; Bell, F.; Tür, G., 2020. Joint Contextual Modeling for ASR Correction and Language Understanding. In: *2020 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2020, Barcelona, Spain, May 4-8, 2020*. IEEE, pp. 6349–6353. Available from doi: 10.1109/ICASSP40776.2020.9053213.

Wieting, J.; Bansal, M.; Gimpel, K.; Livescu, K., 2016. Charagram: Embedding Words and Sentences via Character n-grams. In: Su, J.; Carreras, X.; Duh, K. (eds.). *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*. The Association for Computational Linguistics, pp. 1504–1515. Available from doi: 10.18653/V1/D16-1157.

Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; Moi, A.; Cistac, P.; Rault, T.; Louf, R.; Funtowicz, M.; Brew, J., 2019. HuggingFace's Transformers: State-of-the-art Natural Language Processing. *CoRR*. Vol. abs/1910.03771. Available from arXiv: 1910.03771.

Xu, M.; Duan, L.; Cai, J.; Chia, L.; Xu, C.; Tian, Q., 2004. HMM-Based Audio Keyword Generation. In: Aizawa, K.; Nakamura, Y.; Satoh, S. (eds.). *Advances in Multimedia Information Processing - PCM 2004, 5th Pacific Rim Conference on Multimedia, Tokyo, Japan, November 30 - December 3, 2004, Proceedings, Part III*. Springer. Vol. 3333, pp. 566–574. Lecture Notes in Computer Science. Available from doi: 10.1007/978-3-540-30543-9_71.

Yenigalla, P.; Kumar, A.; Tripathi, S.; Singh, C.; Kar, S.; Vepa, J., 2018. Speech Emotion Recognition Using Spectrogram & Phoneme Embedding. In: Yegnanarayana, B. (ed.). *19th Annual Conference of the International Speech Communication Association, Interspeech 2018, Hyderabad, India, September 2-6, 2018*. ISCA, pp. 3688–3692. Available from doi: 10.21437/INTERSPEECH.2018-1811.

Young, S.; Evermann, G.; Gales, M.; Hain, T.; Kershaw, D.; Liu, X.; Moore, G.; Odell, J.; Ollason, D.; Povey, D., et al., 1999. *The HTK book*.

Zailan, A. S. M.; Teo, N. H. I.; Abdullah, N. A. S.; Joy, M., 2023. State of the Art in Intent Detection and Slot Filling for Question Answering System: A Systematic Literature Review. *International Journal of Advanced Computer Science and Applications* [online]. Vol. 14, no. 11 [visited on 2024-07-02].

ISSN 21565570, ISSN 2158107X. Available from DOI: 10.14569/IJACSA.2023.0141103.

ZHANG, X.; WANG, H., 2016. A Joint Model of Intent Determination and Slot Filling for Spoken Language Understanding. In: KAMBHAMPATI, S. (ed.). *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016.* IJCAI/AAAI Press, pp. 2993–2999. Available also from: http://www.ijcai.org/Abstract/16/425.

ZHAO, L.; FENG, Z., 2018. Improving Slot Filling in Spoken Language Understanding with Joint Pointer and Attention. In: GUREVYCH, I.; MIYAO, Y. (eds.). *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 2: Short Papers.* Association for Computational Linguistics, pp. 426–431. Available from DOI: 10.18653/V1/P18-2068.

ZHOU, Z.-H., 2018. A brief introduction to weakly supervised learning. *National Science Review* [online]. Vol. 5, no. 1, pp. 44–53 [visited on 2024-07-16]. ISSN 2095-5138. Available from DOI: 10.1093/nsr/nwx106.

# List of Figures

# List of Tables

# A Attachments

## A.1 First Attachment

We attach a ZIP archive with the dataset annotations; the audio recordings were too large and are available in the original archive (Plátek et al., 2016). We also share our model source code and other scripts. Consult the provided README for details.