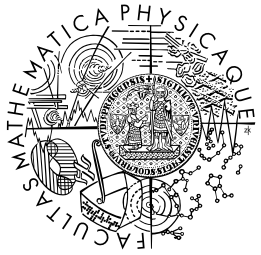

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Jonáš Kulhánek

End-to-end dialogue systems with pretrained language models

Institute of Formal and Applied Linguistics

Supervisor of the master thesis: Mgr. Ondřej Dušek, Ph.D.

Study programme: Computer Science (N1801)

Study branch: IUI (1801T036)

Prague 2021

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I would like to thank Mgr. Ondřej Dušek, Ph.D., for his invaluable advice and support with writing this thesis. I would also like to express my thanks to Mgr. Vojtěch Hudeček and Mgr. Tomáš Nekvida, who provided inspiring ideas and helped me with implementation. I want to thank MFF, Charles University and CIIRC, Czech Technical University, for providing computational resources, without which this research would not be possible.

Title: End-to-end dialogue systems with pretrained language models
Author: Jonáš Kulhánek
Institute: Institute of Formal and Applied Linguistics
Supervisor: Mgr. Ondřej Dušek, Ph.D.

Abstract: Current dialogue systems typically consist of separate components, which are manually engineered to a large part and need extensive annotation. End-to-end trainable systems exist but produce lower-quality, unreliable outputs. The recent transformer-based pre-trained language models such as GPT-2 brought considerable progress to language modelling, but they rely on huge amounts of textual data, which are not available for common dialogue domains. Therefore, training these models runs a high risk of overfitting. To overcome these obstacles, we propose a novel end-to-end dialogue system called AuGPT. We add auxiliary training objectives to use training data more efficiently, and we use massive data augmentation via back-translation and pretraining on multiple datasets to increase data volume and diversity. We evaluate our system using automatic methods (corpus-based metrics, user simulation), human evaluation as part of the DSTC 9 shared task challenge (where our system placed 3rd out of 10), as well as extensive manual error analysis. Our method substantially outperforms the baseline on the MultiWOZ benchmark and shows competitive results with state-of-the-art end-to-end dialogue systems.

Keywords: machine learning, dialogue systems, deep learning, pretrained language models

Název práce: End-to-end dialogové systémy s předtrénovanými jazykovými modely
Autor: Jonáš Kulhánek
Katedra (ústav): Ústav formální a aplikované lingvistiky
Vedoucí diplomové práce: Mgr. Ondřej Dušek, Ph.D.

Abstrakt: Současné dialogové systémy se obvykle skládají ze samostatných komponent, které jsou z velké části vytvořeny ručně a vyžadují rozsáhlé anotace dat. Existují end-to-end trénovatelné systémy, které jsou ale méně spolehlivé a produkují méně kvalitní výstupy. Současné předtrénované jazykové modely založené na transformer architektuře, jako je GPT-2, přinesly do modelování jazyka značný pokrok, ale současně vyžadují velké množství textových dat, která nejsou pro běžné dialogové domény k dispozici. Proto je při trénování těchto modelů vysoké nebezpečí přeučení. Abychom tyto překážky překonali, navrhujeme nový end-to-end dialogový systém nazvaný AuGPT. Abychom efektivněji využili trénovací data, rozšiřujeme architekturu o pomocné moduly, a abychom zvýšili množství a rozmanitost dat, využíváme rozsáhlé augmentace dat pomocí zpětného překladu a předtrénování na více datových sadách. Náš systém vyhodnocujeme pomocí automatických metod (korpusové metriky, simulace uživatele), lidského vyhodnocení v rámci soutěže DSTC 9 shared task challenge (kde se náš systém umístil na třetím místě z 10) a také rozsáhlé manuální analýzy chyb. Naše metoda podstatně překonává baseline na benchmarku MultiWOZ a vykazuje výsledky konkurenceschopné s nejmodernějšími end-to-end dialogovými systémy.

Klíčová slova: strojové učení, dialogové systémy, hluboké učení, předtrénované jazykové modely

Contents

1. Introduction

Even before computers were invented, people already thought about how they can eventually behave similarly to humans and communicate in natural language (Turing, 1950). Unfortunately, this task is immensely difficult because the systems would have to understand all the things humans understand and would have to be able to infer from this knowledge. Essentially, the machines would have to think in the same way humans do. The systems that communicate with the user in natural language are called *dialogue systems*, and while they cannot behave in the same way as humans, in a restricted form, they are being applied to a variety of problems. The reason is that in some cases, it is sufficient if the systems can communicate in natural language about a few fixed topics. For example, a lot of phone users use *task-oriented* dialogue systems on a daily basis as an alternative interface for the phone, allowing them to use voice commands to play favourite music or find train connections. Unfortunately, currently deployed systems are engineered by humans in labour-intensive process and not able to learn from data. Trainable alternatives were already proposed, but their quality and reliability are nowhere near the quality of manually designed systems. However, recent advances in deep learning reduced the gap substantially.

Traditionally, task-oriented dialogue systems were based on modularized pipelines (Young et al., 2013; Gao et al., 2019a). The idea was that the individual components would be transferrable between different systems, and the design would simplify. Also, since the computational resources were limited at the time of designing the dialogue systems, decomposing them made it easier to use machine learning. Unfortunately, in order to make the system trainable, data had to be engineered and labelled for each component. Another problem with these approaches was that errors accumulated as the data propagated through the pipelines.

End-to-end dialogue systems, on the other hand, do not need the explicit intermediate representation in order to be able to learn from the dialogues. Nearly all functionality required to hold the conversation is integrated into a single deep neural network (Wen et al., 2017; Eric et al., 2017; Lei et al., 2018). These systems are still in the early stages of development and are not yet ready to be deployed to production. However, in recent years thanks to the increase of computational power and the mass expansion of deep neural networks, they made a considerable leap forward. The problem with dialogue modelling is that dialogue datasets are much smaller compared to other natural language processing domains. Training neural networks has a high risk of overfitting. Therefore, the models have to be efficient in the amount of data they need for training. After their recent advent, large pre-trained transformer-based language models (Devlin et al., 2019; Radford et al., 2018, 2019; Zhang et al., 2020c) soon became widely adopted in the dialogue modelling domain. The recipe for their success was not their efficiency in terms of the number of training samples they need for training, but in the powerfull prior knowledge they retained from pre-training on vast ammounts of data. On the other hand, one could argue that these models are much more prone to overfitting than recurrent neural networks due to the massive number of their parameters. The pre-training enables these models to be be fine-tuned with relatively few dialogue

data. For example, when a language model is fine-tuned for generating responses in a dialogue system, it can already generate grammatically correct sentences; we just need to bias it towards the dialogue domain. A demonstration of this transfer is a system based on GPT-2 (Radford et al., 2019), which was introduced by Budzianowski and Vulić (2019), who showed that a system trained on a large number of open-domain dialogues with no annotations could be fine-tuned to a specific task-oriented dialogue domain relatively easily with only a small amount of data required.

Using these powerful language models, however, causes other problems. When applied to response generation, due to their capacity and also due to their ability to generate valid sentences, they can sometimes generate a text that does not correlate with the input to the model. In other words, the model *hallucinates* words without any connection to the knowledge base. This problem is called *lack of grounding* (Huang et al., 2020). The second problem is that although the models were pre-trained on large amounts of text data, it is not guaranteed that this knowledge will be preserved. In fact, it is quite the opposite. Fine-tuning the models, especially on small datasets, may cause catastrophic forgetting (Greco et al., 2019) and the model can lose its ability to generate diverse responses. In extreme cases, it will overfit the training dataset to such an extent that it will memorize it completely.

In this thesis, we address the above problems with pre-trained language models. We follow the research in the field of end-to-end task-orinted dialogue systems with pre-trained language models. We aim to fulfil the following objectives:

1. Reimplement one of the currently best end-to-end dialogue system based on GPT-2 language model, SOLOIST (Peng et al., 2020).

2. Propose and experiment with different improvements of the system in terms of the architecture and training of the deep neural network (DNN), and also with pre-processing of the data.

3. Evaluate the system and its variants on MultiWOZ benchmark and compare it to the original SOLOIST (Peng et al., 2020) method, as well as other state-of-the-art systems.

We propose an end-to-end task-oriented dialogue system called AuGPT. The model is based on the GPT-2 language model (Radford et al., 2019) and extends a prior method called SOLOIST, which was proposed by Peng et al. (2020). The contributions can be summarized as follows:

- AuGPT is pre-trained on multiple different datasets, and the final dataset, which is used for fine-tuning, is massively augmented by paraphrasing. The paraphrases are automatically generated by back-translation – all texts are translated to multiple intermediate languages and back to English (Edunov et al., 2018; Sennrich et al., 2016).

- We propose a novel dialogue consistency detection auxiliary task. Similarly to Peng et al. (2020), we corrupt half of the training samples by either replacing the belief state or response with a randomly chosen one or resampling the belief state's values. A binary classifier is attached to the main network,

and it is trained to detect corrupted dialogues. The task is used only during optimization to help guide the gradients in the correct direction.

- To increase the diversity of the generated responses, the model uses token unlikelihood loss proposed by Welleck et al. (2020). Also, different decoding strategies are used for the belief state and for the response. The belief state decoding, where precision is prefered, uses the greedy search, whereas nucleus sampling (Holtzman et al., 2020) is used for the response generation to further promote diversity.

- Traditionally, end-to-end task-oriented dialogue systems are not trained on the final responses but on *delexicalized responses*, obtained by replacing concrete words like names of hotels with placeholders like `[name]`. In Section 4.4.2, we propose an alternative approach to traditional imperfect lexicalization, which lets the system generate the final response. This alternative approach is carefully compared with the baseline in Section 6.5 and Section 6.7.3. The results are discussed in Section 7.3.

- The model is carefully evaluated and compared with state-of-the-art methods using MultiWOZ (Budzianowski et al., 2018) and ConvLab 2 (Zhu et al., 2020) automatic evaluations (see Chapter 5). Each choice in the process of designing the system is validated by an ablation study. We also include a detailed manual error analysis in Section 5.7.

- Finally, a variant of the dialogue system took part in the DSTC 9 challenge (Gunasekara et al., 2020), where the system placed third out of ten. The details of the competition are given in Section 5.5 and the results are displayed in Section 6.6.

In Chapter 2, the reader is first introduced to concepts used throughout the thesis. The following Chapter 3 gives details on similar methods and draws parallels between them to help the reader understand the importance of the approach. Next, in Chapter 4, the method itself is described. We have designed experiments to evaluate the quality of the dialogue system, and the details are given in Chapter 5. The following Chapter 6 presents the results of the experiments, and the results are discussed in Chapter 7.

2. Theoretical background

In this chapter, the reader is introduced to concepts used throughout the rest of the thesis. The chapter starts with language modelling (Section 2.1). A language model is formally defined, and different decoding strategies are explained. Section 2.2 continues with a description of deep neural networks. They are formally defined, and we explain how these models are trained. Then, we introduce different neural network architectures such as recurrent neural networks and encoder-decoder models. In Section 2.3, an important architecture called *transformers* (Vaswani et al., 2017) is introduced. We explain how attention and positional encoding are used in these models. We also summarize recent advances in transformer-based language models. Finally, in Section 2.4, we define task-oriented dialogue systems and describe the components from which the dialogue systems were traditionally composed.

2.1 Generative language modelling

Language modelling is applied across many natural language processing (NLP) areas, including dialogue systems. Essentially, a *language model (LM)* represents a probability distribution over natural language texts in a chosen language (or multiple languages). A text is generally represented as sequences of *tokens*, e.g., words, word pieces, or individual letters. We will denote the sequence of tokens by $\boldsymbol{x}$. The probability represented by the language model can formaly be denoted as follows:

$$p(\boldsymbol{x}) = p(x_1, x_2, ..., x_t)\,. \tag{2.1}$$

Let us consider a category of LMs, where we model the probability of the next token in the sequence based on the previous tokens. We will call these models *causal language models*. Formaly, they model the following probability distribution:

$$p_t(x_t|\boldsymbol{x}_{<t})\,, \tag{2.2}$$

which can be used to factorize the LM probability distribution, given in (2.1), as follows:

$$p(\boldsymbol{x}) = \prod_{i=1}^{t} p_i(x_i|\boldsymbol{x}_{<i})\,. \tag{2.3}$$

These models allow us not only to model the probability of sequences of tokens, but they can also be used to greedily generate texts because we are able to repeatedly sample the next token in the sequence from a relatively small distribution, as opposed to the original LM probability distribution, where sampling from the distribution over all texts would not be tractable. Furthermore, we are able to generate multiple sequences of tokens by either sampling multiple tokens at each generation step for the direct sampling or by keeping a fixed number of hypothesis with the highest probability. Generating multiple sequences of tokens is called *beam search*, where the number of generated sequences is usually called the *width* of the beam search.

Another alternative for generating sequences of tokens from modern DNN-based causal LMs is called the *nucleus sampling* or the *top-p sampling* (Holtzman

et al., 2020), which performs exceptionally well on transformer-based architecture such as GPT (Radford et al., 2018, 2019). The nucleus sampling is similar to greedy decoding, but the next token is sampled from a truncated distribution instead of the full one. At each timestep, the nucleus sampling choses a minimal set of tokens with the total probability higher than a fixed constant p. This can be expressed as follows:

$$V_p(\boldsymbol{x}_{<t}) = \underset{\substack{V \in 2^W \\ \sum_{x_t \in V} p(x_t|\boldsymbol{x}_{<t}) \geq p}}{\arg\min} |V| \,, \tag{2.4}$$

where W is the set of all possible tokens and 2^W denotes the set of all subsets of W. The probabilities of the tokens from the V_p set are rescaled to sum to one and the next token is sampled from this distribution.

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial L}{\partial \theta_i} \,, \tag{2.5}$$

2.2 Deep neural networks

In recent years, thanks to the rise of computational power, models called deep neural networks (DNNs) raised a lot of attention. They are applied to a variety of domains ranging from image recognition to automatic speech recognition. Perhaps the main benefit of these models is that we have efficient optimization algorithms to train these models effectively.

We will define a class of functions called DNNs, whose goal is to approximate an unknown function f^* (Goodfellow et al., 2016). Let a DNN architecture be a directed acyclic graph, where each node represents a function on a tuple of tensors, e.g., affine transformation, ReLU.[1] The in-degree of each node is the same as the arity (the number of arguments) of the function associated with the graph node. We will call each node with its associated function an *operation*. Let o_1, o_2, ..., o_n be a topological ordering of the operations, which clearly exists since the graph is acyclic. Each operation o_i takes as the input the output of operations from which there exists a directed edge to o_i. These operations already computed their results because of the topological ordering. We are, therefore, able to compute the output of all operations in this graph. We also have a set of input nodes and a set of output nodes. Input nodes will have no incoming edges, whereas output nodes will have no outgoing edges. Furthermore, an ordered set of input nodes will correspond to the input to the function f^*, where the output of each input node is the input to the DNN. An ordered set of output nodes will correspond to the output of function f^*, where the output of the DNN is the result of the operations represented by these nodes. Each operation in the graph could be parametrized by a set of parameters θ_i, and the set of all parameters of all operations will be denoted by θ. A DNN is parametrized by the DNN architecture and the set of its parameters. When we talk about a DNN in this thesis, we will usually assume a fixed architecture and describe DNN as a function $f(\boldsymbol{x}|\theta)$, where the DNN architecture is fixed and determines the class of functions f.

[1] ReLU (Rectified Linear Unit) is a function mapping x to $\max(0, x)$

2.2.1 Deep neural network optimization

Suppose there is a probability distribution over the input space of the DNN and we have a sample from this distribution called the *dataset*. Usually, we do not have access to the unknown function f^* or it may not exist; however, in the case of *supervised learning* we have a sample from a distribution over input-output pairs. In order to find parameters that make the DNN with a fixed architecture a better approximator of the unknown function f^*, we define a *loss function* L which takes as the input the output of the DNN and the true value and measures the difference between these two. The *expected loss* (risk function) is then given as the expected value of the loss function over the entire dataset:

$$L(\mathcal{D}|\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}[L(f(\boldsymbol{x}|\theta), \boldsymbol{y})] \tag{2.6}$$

Some of the most frequently used loss functions are *mean squared error*, which is used when we assume that $p(y|x)$ is a Gaussian distribution, and *cross-entropy* when $p(y|x)$ is categorical distribution. Since the latter is used in language modelling, we will describe it in more detail. The cross-entropy is given as follows:

$$L(\hat{\boldsymbol{y}}, y) = -\sum_c \log \hat{y}_c^{\mathbb{1}_{y=c}}\,, \tag{2.7}$$

where y is the true category, $\hat{y}_c$ is the probability predicted by our model for class c, and $\mathbb{1}$ is the indicator function.

We cannot compute the expected loss over the unknown probability distribution, and therefore, we use our sampled dataset instead. The building blocks of DNNs and loss functions are always differentiable almost everywhere, and by construction, this also holds for the expected loss. This allows us to compute the gradients of the expected loss on the dataset w.r.t. the parameters $\boldsymbol{\theta}$, and apply gradient descent methods to decrease the loss function. The simplest posible parameter optimization step is given as follows:

$$\theta_i \leftarrow \theta_i - \alpha\frac{\partial L}{\partial \theta_i}\,, \tag{2.8}$$

where α is called the *learning rate*, which can be either constant or can decrease during training. There are other optimizers such as RMSprop (Graves, 2013), Adam (Kingma and Ba, 2014), AdamW (Loshchilov and Hutter, 2017), which track first and second moments of the gradient (independently for each coordinate) and use them to adjust the parameter update.

2.2.2 Recurrent neural networks

If the data have a natural temporal ordering (i.e. sequences), we can use a class of DNNs called the *recurrent neural networks (RNNs)* to model the temporal dependencies. Intuitively, an RNN is a DNN which is applied to the input sequentially, updating its *hidden state* at each time step. After all of the input was processed by the RNN, its internal state will contain a representation of the whole input sequence and can be used as an input to a classifier, for example.

Let the input to the network be a sequence $\{\boldsymbol{x}\}_{t=1}^T$. An RNN is a function $f(\boldsymbol{x}_t, \boldsymbol{s}_{t-1}|\theta) \to (\boldsymbol{y}_t, \boldsymbol{s}_t)$ where $\boldsymbol{s}_t \in \mathcal{S}$ is the *state* of the RNN at time step t, s_0 is

an *initial state*, and $\boldsymbol{y}_t$ is the RNN output at time t. The final output of the RNN is both the sequence $\{\boldsymbol{y}\}_{t=1}^{T}$ and the final state $\boldsymbol{s}_T$. During training, in order to compute the gradients of the loss function w.r.t. the parameters of the network, the gradient of the loss function is backpropagated through time in reversed order:

$$\frac{\partial \mathcal{L}}{\partial \theta_i} = \sum_{t=1}^{T} \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_t} \frac{\partial \boldsymbol{y}_t}{\partial \theta_i} + \frac{\partial \mathcal{L}}{\partial \boldsymbol{s}_t} \frac{\partial \boldsymbol{s}_t}{\partial \theta_i} \right) \tag{2.9}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{s}_t} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{s}_{t+1}} \frac{\partial \boldsymbol{s}_{t+1}}{\partial \boldsymbol{s}_t} + \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_{t+1}} \frac{\partial \boldsymbol{y}_{t+1}}{\partial \boldsymbol{s}_t} \tag{2.10}$$

2.2.3 Encoder-decoder architecture

In many NLP applications such as translation or summarization, we need to model a mapping from sequences to other sequences where the length generally differs. In order to achieve that, we can use the *encoder-decoder* (seq2seq) architecture (Cho et al., 2014; Bahdanau et al., 2015; Sutskever et al., 2014), which consists of two RNNs – the encoder and the decoder. The encoder reads the input sequence sequentially and outputs its final state. The decoder takes the generated state as its initial state and a special `<bos>` token as its input and generates the first token of the output sequence. The generated token is passed as the input to the decoder, and this is repeated until `<eos>` token is generated by the decoder. The decoding is visualized in Figure 2.1. We say that the generator generates the output sequence *autoregressively*, which means that it uses the generated tokens as the input to the next time step. As discussed in Section 2.1, different *decoding strategies* for generating the output sequence of tokens such as beam search, nucleus sampling (Holtzman et al., 2020), or top-k sampling can be employed.

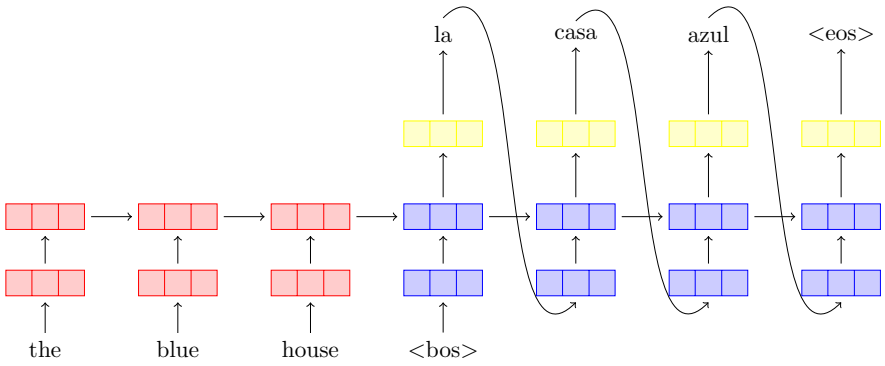

Figure 2.1: Encoder-decoder sequence generation. The encoder (red) generates the initial state for the decoder (blue), which autoregressively generates the output sequence.

One problem with RNNs is that the hidden size with a fixed length has to capture all the information from all previous time steps regardless of the sequence size. In practice, the models tend to propagate the signal well for shorter sequences but have difficulties capturing longer input. Also, the longer the distance between two time steps, the more difficult it is for the gradient to propagate through the sequence, and at the end, the signal is usually weak. In order to tackle this issue, *attention mechanisms* were proposed (Bahdanau et al., 2015; Luong et al., 2015),

which allow the decoder to read (*attend* to) states at any time step generated by the encoder. At each time step of the generation, the decoder also outputs an *query* vector, which is used to find the time step the decoder wants to attend to. Then, the *alignment* between the query and each of the generated states is computed, and finally, the resulting vector is computed as a weighted sum of the generated states, where the weights are given by alignment scores. The output of the attention is used as another input to the decoder. A question is whether RNNs are still needed or if the attention alone is able to represent the input sequence. Models called *transformers*, described in the following section, try to use only attention mechanisms to achieve the same goal as encoder-decoder models.

2.3 Transformers

While RNNs enable us to train sequence classifiers or even sequence-to-sequence mappings in the case of encoder-decoder architectures, the training cannot be done efficiently due to their sequential nature – the time needed grows linearly with the length of the input or output sequence. Alternative models called transformers were proposed to resolve this issue. Transformers have a constant time complexity (with the assumption that matrix multiplication is an elementary operation). The transformer is a DNN architecture proposed by Vaswani et al. (2017), which does not use any recurrent connections, but only attention mechanisms in combination with feedforward neural networks.

A typical transformer-based neural network consists of encoder and decoder blocks. The overview of the architecture is given in Figure 2.2. The encoder consists of multiple identical layers, which are composed of two sub-layers: the multi-head self-attention layer (described in Section 2.3.1) followed by a simple feed-forward neural network, which is applied independently to each time step. Residual connections are employed around both sub-layers, followed by a layer normalization operation (Ba et al., 2016). The decoder layers are similar to encoder layers, but another multi-head attention sub-layer is introduced after the multi-head self-attention layer, which attends over all outputs of the individual encoder layer. This is visualized in Figure 2.2. Also, the self-attention layer is modified in such a way that at each time step, the model can attend only to previous time steps – future ones are zeroed out.

Since the attention is invariant to the sequential order of the data and the feed-forward neural network is applied independently for each time step, we need to introduce the order information to the input to the model. To achieve that, Vaswani et al. (2017) add *positional encoding* to each input token embedding. More details are given in Section 2.3.2

The output of the model is a softmax-activated affine transformation of the last decoder layer's output. The model is usually trained using the cross-entropy loss, where the output tokens are used as the target. The same embedding matrix is used for both the input tokens and the output tokens to reduce the number of parameters. Furthermore, the same weight matrix is also used in the last affine transformation. During inference, autoregressive decoding is used – the same as in the encoder-decoder RNN architectures described in Section 2.2.3.

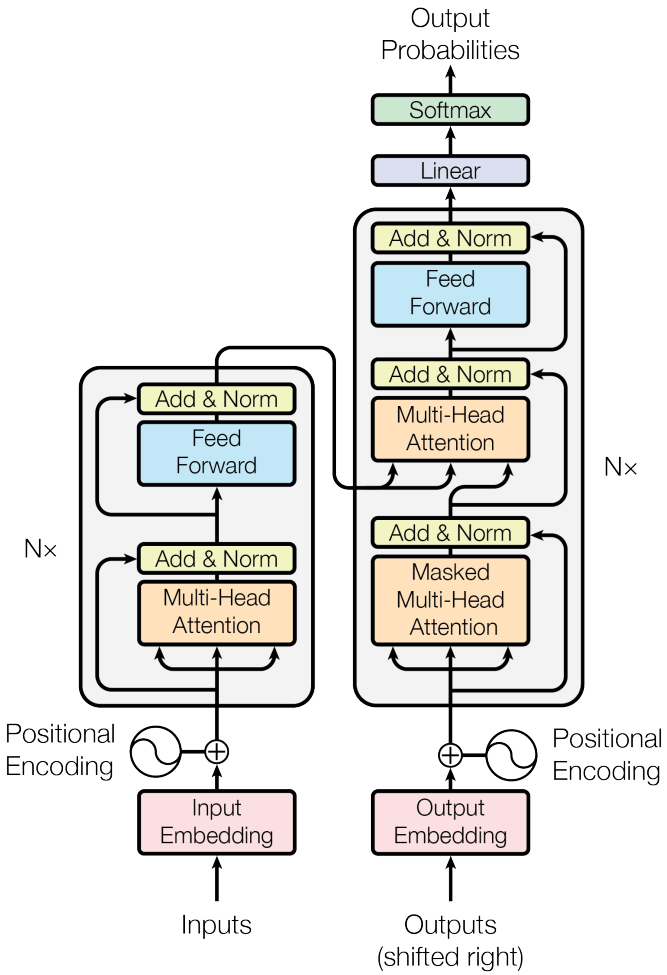

Figure 2.2: The transformer encoder and decoder block overview. Source: Vaswani et al. (2017).

2.3.1 Attention

The multi-head attention layer is an important part of the architecture. It maps a query and a set of key-value pairs, each of which is a vector, to a vector output.

Each *head* of the multi-head attention layer is represented by the *scaled dot-product attention*, which itself maps a query $\boldsymbol{q}$, a key $\boldsymbol{k}$, and a value $\boldsymbol{v}$ to a vector output. The function output is usually computed for each time steps at the same time. Let us denote the queries, keys, and values for all time steps by matrices $Q \in \mathbf{R}^{t \times d_k}$, $K \in \mathbf{R}^{t \times d_k}$, and $V \in \mathbf{R}^{t \times d_v}$, where t denotes the total time steps, d_k denotes the dimensionality of both $\boldsymbol{k}$ and $\boldsymbol{q}$, and d_v denotes the dimensionality of $\boldsymbol{v}$. The result of the attention is then computed as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{2.11}$$

The output itself is a matrix of shape $t \times d_v$. The term $\frac{1}{\sqrt{d_k}}$ was introduced by Vaswani et al. (2017), who argued, that for large values of d_k, the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients. Deviding the dot products by $\sqrt{d_k}$ counteracts this effect.

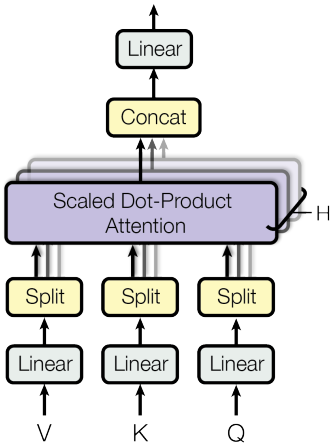

Figure 2.3: The transformer multihead attention. Source: Vaswani et al. (2017).

In the multi-head attention, an affine transformation is applied first to the query, the key, and the value. Then, these vectors are split into H pieces of identical sizes, where H is the total number of heads. For each head h of the multi-head layer, the input is the h-th part of the query, the key, and the value, resulting in different $\boldsymbol{q}_h$, $\boldsymbol{k}_h$, and $\boldsymbol{v}_h$. These vectors are passed through the h's scaled dot-product attention layer. The results from each head are then concatenated, and an affine transformation is applied to the result. This is visualized in Figure 2.3.

2.3.2 Positional encoding

Attention alone is invariant to the sequential order of the tokens. Therefore, to make use of the order, Vaswani et al. (2017) use *positional encoding*. The *positional encoding* vectors are simply summed with the input and output tokens embeddings at the bottom of both the encoder and the decoder stacks. Therefore,

the length of a *positional encoding* vector has to match with the length of the token embedding vector. Sine and cosine functions of different frequencies are used for different components of the positional embedding vector. The positional embeddings $\boldsymbol{\phi}^{(t)}$ for the time step t are computed as follows:

$$\begin{aligned} \boldsymbol{\phi}_{2i}^{(t)} &= \sin(t \cdot 10000^{-2i/d_m}) \\ \boldsymbol{\phi}_{2i+1}^{(t)} &= \cos(t \cdot 10000^{-2i/d_m}) \,, \end{aligned} \tag{2.12}$$

where the index $2i$ and $2i+1$ indexes the vector $\boldsymbol{\phi}^{(t)}$ and d_m is the dimension of the input and the output token embeddings.

2.3.3 Transformer-based language models

The transformer architecture was designed for modelling sequences, and therefore, was used as an LM (Vaswani et al., 2017; Radford et al., 2018, 2019; Devlin et al., 2019) with outstanding success. Radford et al. (2018) used a decoder-only architecture trained on large corpora in an autoregressive manner, i.e. the model predicted the next token based on the history. Instead of a fixed positional encoding, Radford et al. (2018) used trained positional embeddings. Later, Radford et al. (2019) proposed a slightly altered method where the layer normalization was placed before the residual, and the weights were initialized differently. The model was trained on even more data and showed exceptional text generation capabilities.

A different line of research uses encoder-only architectures. Devlin et al. (2019) designed a model called BERT which is able to capture the bidirectional context as opposed to leftward context used in autoregressive architectures (Vaswani et al., 2017; Radford et al., 2018). In order to train the model, the input sequence is masked with unknown *[mask]* tokens, and the model is trained to recover the original tokens. This is conceptually similar to word-to-vec models (Mikolov et al., 2013) for pre-training word embeddings by predicting a word from its surroundings. The model also used different auxiliary tasks, e.g., next sentence prediction, where a classifier is trained to classify if a sentence follows another in the original corpus. Other models based on text masking and corruption were proposed (Lewis et al., 2019; Liu et al., 2019), outperforming the original BERT.

These models are pre-trained on large text corpora in an unsupervised fashion. Later, they can be fine-tuned to perform domain-specific tasks like question answering or sentiment analysis. In order to train the original LM for a specific classification task, usually, a feed-forward neural network is attached to the last hidden representation of the model.

While decoder-only architectures (Radford et al., 2018, 2019) can be applied directly to natural language generation problems, encoder-based approaches can be extended with a decoder to be able to generate texts (Lewis et al., 2019).

2.4 Task-oriented dialogue systems

A *dialogue system* or a *conversational agent* is a computer program which communicates with *users* in natural language. The definition covers different forms of input – text, speech, or a combination of the two. We can, however, use speech

recognition (Baevski et al., 2020) and speech synthesis (Oord et al., 2018; Wang et al., 2017) systems to recognize the speech and translate it to text, and to generate the speech from text. Both speech recognition and speech synthesis could be considered independently from dialogue systems, and therefore, in this thesis, we will restrict our attention to text-based dialogue systems.

We can categorize dialogue systems into two categories: *task-oriented dialogue systems*, and *chatbots*. Task-oriented dialogue systems use a conversation with users to achieve a predefined task. Digital assistants such as Siri, Alexa, or Google Home, are some examples of task-oriented dialogue systems. Chatbots, on the other hand, are designed to attract user's attention through conversation. Their goal is to mimic natural conversation and keep user attracted to it. Chatbots (Gao et al., 2019b; Jiang and de Rijke, 2018) are generally harder to objectively evaluate and can be considered an instance of language modelling. In this thesis, we focus on task-oriented systems only.

A task-oriented dialogue system communicates with the user, responds to the user input and, through the conversation, fulfils a fixed goal. The goal of the dialogue system could range from searching and returning records from a database to making reservations and booking flight tickets. We can abstract these use cases in the form of an external system, which responds to *queries* and returns a list of results as a response, possibly with side-effects in the external world. Information retrieval systems, centered around a read-only database, do not have any side effects in the external world, whereas an ordering system is expected to have side effects.

We will use the term *utterance* for each continuous text – user input or system response. In a task-oriented dialogue, the system and the user take *turns*, i.e., they respond to each other's utterances. A dialogue is then a sequence of interleaved system's and user's utterances. Each utterance is generated based on all previous utterances in the dialogue called the *dialogue context*. Examples of dialogues can be seen in Section 6.7.3.

Traditionally, dialogue systems were composed of several modules: natural language understanding (NLU), dialogue state tracking (DST), dialogue policy, and natural language generation (NLG). Figure 2.4 shows how user input passes through each component until the system response is generated.

Natural language is ambiguous and may contain redundant and unimportant information. Therefore, NLU is responsible for parsing the natural language user input into a more structured representation called *dialogue acts* which are unambiguous and contain only the information required for the dialogue system. Each dialogue act assigns a *value* to a predefined *slot*. We can see an example of dialogue acts in Figure 2.4 under the NLU component, where NLU assigns value 'Italian' to a slot called 'food'. Each user input can be categorized into several types. These may include informing the system about something, asking for clarification, etc. This category is called *user intent* and is usually contained in the dialogue acts under special 'intent' slot.

DST is then applied to incorporate the dialogue history from past utterances. Internally, it keeps and updates a *belief state*, which summarizes the dialogue history. In Figure 2.4, we can see how DST updated the prior belief state: `food=French, price=cheap`, by replacing the slot 'food' with a new value 'Italian' from the dialogue acts. This resulted in a new belief state:

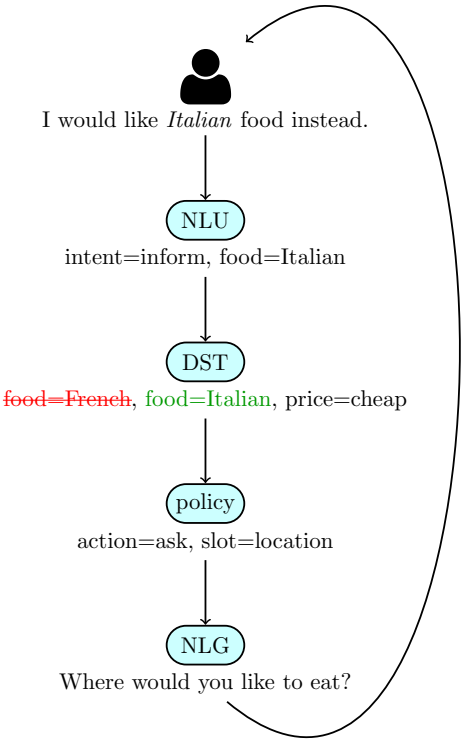

Figure 2.4: Traditional dialogue system pipeline. The user input passes through several subsystems: natural language understanding (NLU), dialogue state tracking (DST), dialogue policy, and natural language generation (NLG).

`food=Italian, price=cheap`.

The *dialogue policy* then uses the belief state to decide what to output to the user. It has to keep track of what information is required from the user and ask for missing information. The dialogue policy choses a *system action*, which specifies the type of response the dialogue system will return to the user. The *system action* is has similar function as *user intent*, but whereas the *user intent* specifies the type of user input the *system action* is the type of system response. The *system action* is returned by the policy as one of several dialogue acts. In Figure 2.4, we can see how the *dialogue policy* decided it needed more information about the restaurant and outputted a new dialogue acts, where the action was to ask the user to fill the slot value for the slot called 'location'.

Finally, NLG is the inverse of NLU – it takes the dialogue act entities generated by the dialogue policy and transforms them into natural language text, which can be returned to the user.

2.4.1 Natural language understanding

The NLU component is responsible for removing the ambiguity in the natural language text by translating it into dialogue acts entities. Simplest approaches used rule based systems and combinatory categorial grammar (CCG) (Artzi and Zettlemoyer, 2011; Zettlemoyer and Collins, 2007). Classifiers such as support vector machines (SVMs) and string kernels were also used (Kate and Mooney, 2006; Le Nguyen et al., 2006). More advanced classical methods used conditional random fields (CRFs) (Huang et al., 2015; Lafferty et al., 2001).

Modern approaches are mostly based on DNNs (Hakkani-Tür et al., 2016; Liu and Lane, 2016; Jiao et al., 2020; Eberts and Ulges, 2019). Encoder-decoder architectures are employed often enhanced with attention mechanisms (Hakkani-Tür et al., 2016; Liu and Lane, 2016). Recently, transformers were also successfully applied (Jiao et al., 2020; Eberts and Ulges, 2019).

2.4.2 Natural language generation

In practice, natural language generation is often rule-based or template-based (Reiter and Dale, 1997; Busemann and Horacek, 1998). A template is a string with special placeholders, which are later replaced with slot values. Language generation could be considered an instance of language modelling where we explicitly condition on the dialogue acts (Mairesse and Young, 2014; Wen et al., 2015c, 2016b). RNNs were also used in modern systems (Dušek and Jurčíček, 2016; Dušek and Jurcicek, 2019; Wen et al., 2015c, 2016b). Dušek and Jurčíček (2016) use encoder-decoder architecture to generate the output text sequentially. Most approaches do not generate the final text, but only the *delexicalized text*, which is basically a template of the final text, where the concrete names and values are replaced with placeholders. The dialogue acts are used to replace the placeholders in a post-processing step to obtain the final text. An alternative is to use copy mechanisms to force the RNN to copy text from the input (Gehrmann et al., 2018).

3. Related work

Traditionally, dialogue systems were based on the *dialogue-state architecture*, which consists of several components: NLU, DST, policy, NLG. Although more complex, the architecture was inspired by the GUS system, which was introduced by Bobrow et al. (1977). Using the belief state (see Section 2.4) for modelling the uncertainty in the dialogue was suggested in early attempts by Pulman (1996); Horvitz and Paek (1999); Meng et al. (2003). Young et al. (2010) used the dialogue-state architecture in their restaurant recommendation system.

Early research often considered each component of the dialogue system individually. Often, many of these components were manually engineered. For the NLU component, however, Suendermann et al. (2009) showed that trainable classifiers could often outperform the original rule-based systems. Recurrent neural networks were used by Kim et al. (2017); Lee et al. (2019). Recently, pre-trained transformer-based language models such as BERT (Devlin et al., 2019) were also used (Zhu et al., 2020). In their work, Zhu et al. (2020) attached two feed-forward neural networks to the top of the pre-trained BERT model. These classifiers were trained for intent classification and slot tagging.

Other components are often rule-based. Wu et al. (2019); Ramadan et al. (2018), however, proposed a combined NLU and DST component which maps the utterances directly to belief states. Wu et al. (2019) achieved state-of-the-art results on MultiWOZ 2.1 (Zhu et al., 2020). Also, while, NLG is often template-based (Dhingra et al., 2017; Williams et al., 2017; Henderson et al., 2014) – the for each response, a template is selected from a set of manually engineered templates and placeholders are replaced with correct values (see Section 2.4.2) – an alternative is to use recurrent neural networks in the form of a decoder network (Wen et al., 2015b,a; Dušek and Jurčíček, 2016). In their approach, Dušek and Jurčíček (2016) use a seq2seq architecture to produce natural language texts as well as deep syntax dependency trees from input dialogue acts. The generated hypotheses are then re-ranked based on how closely they match the dialogue acts, and the best hypothesis is selected as the final response.

End-to-end task-oriented systems aim to merge the different components into one system. The first systems tried to mimic the traditional pipelines by explicit modelling of the individual components (Wen et al., 2017). Later, a different strategy was adopted, which used a two-stage setup for the sentence prediction. In their Sequicity dialogue system, Lei et al. (2018) propose to use the seq2seq (encoder-decoder) architecture (Sutskever et al., 2014) for generating the belief state and the system response. At each turn, the previous belief state, the previous response, and the user input are passed to a seq2seq model, which generates the next belief state (in a string representation) and the next system response. This process is illustrated in Figure 3.1. A key observation is that the system does not use the full dialogue context, as recent dialogue systems (Peng et al., 2020; Hosseini-Asl et al., 2020; Ham et al., 2020), but only the last response and the last belief state. This relies on an assumption that the belief state captures the full history of the dialogue. The system is trained on delexicalized responses, i.e., any concrete slot values are replaced by placeholders (Wen et al., 2017; Lei et al., 2018). During inference, the placeholders in the generated delexicalized responses

are replaced back with the slot values retrieved from the database. At its core, Sequicity uses an encoder-decoder architecture called *CopyNet* proposed by Gu et al. (2016). *CopyNet* follows (Bahdanau et al., 2015) (see Section 2.3.1) and modifies the decoder to allow it to copy subsequences of the input sequence to the decoder output.

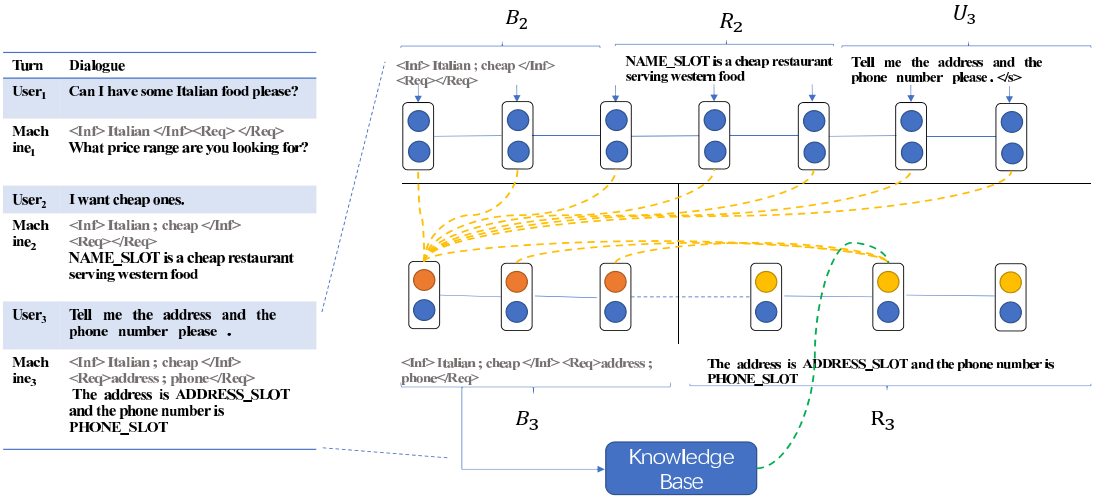

Figure 3.1: Sequicity overview. An example of a dialogue can be seen on the left. The belief states are represented as a strings (see B_2 and B_3 in the figure). In order to decode the next belief state and the next response, the previous belief state and the previous response are passed as the input to a seq2seq model. The model is trained on delexicalized responses, where concrete slot values are replaced with placeholders (see R_2 and R_3). During inference, this process is inversed and the placeholders are replaced with slot values from the database results. Source: Lei et al. (2018).

Recently, large-scale task-oriented datasets were published (Budzianowski et al., 2018; Eric et al., 2020; Byrne et al., 2019; Rastogi et al., 2020), which motivated research on data-driven multi-domain dialogue systems. Unfortunately, earlier approaches that performed well on single-domain datasets (Wen et al., 2016a) struggled with multi-domain scenarios (Zhang et al., 2020b). To address the issue, Zhang et al. (2020a) introduce the LABES-S2S architecture based on Sequicity. Zhang et al. (2020a) identified the need for turn-level annotations as one of the main problems, which makes it difficult for the dialogue systems to scale well to new domains, where these annotations could be prohibitively expensive. Therefore, in LABES-S2S, the belief states – and consequently the database results – are random. This is visualized in the probabilistic graphical model of LABES-S2S in Figure 3.2. Two models are used to generate the belief states: the conditional generative model, which models the probability distribution over belief states given dialogue context, and the approximate posterior model, which models the probability distribution over belief states given dialogue context and the response. Training assumes that a part of the data is annotated, and supervised learning is used as well as unsupervised learning, where the variational evidence lower bound (ELBO) is optimized. Finally, the loss from the response is backpropagated to the belief state using a simple Straight-Through estimator (Bengio et al., 2013). Zhang et al. (2020b) present DAMD – a three-stage sequence-

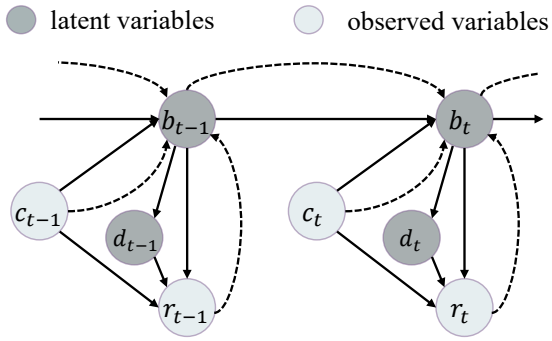

Figure 3.2: The probabilistic graphical model of LABES-S2S (Zhang et al., 2020a). The turn number is denoted as t, and b_t, d_t, c_t, r_t are the belief state, the database results, the dialogue context, and the response in turn t. In LABES-S2S, only the dialogue context and the response are fixed (denoted by lighter gray), whereas the belief state and the database results are random. Solid arrows denote the conditional generative model, and dash arrows denote the approximate posterior used in LABES-S2S. Source: Zhang et al. (2020a).

to-sequence architecture that explicitly decodes the system action, which is later used in the generation of a delexicalized response. Mehri et al. (2019) suggest a pre-trained modular system and Madotto et al. (2020) propose to use meta-learning for parameter selection for different domains. Qin et al. (2020) introduce shared-private neural networks to learn knowledge shared between domains and domain specific knowledge. The approach is evaluated in low-resource settings and even on unseen domains. Reinforcement learning is used in the LAVA model (Lubis et al., 2020) to learn a policy over latent system actions that are initialized using a variational auto-encoder.

Pre-trained transformer-based language models (see Section 2.3.3) were successfully applied to dialogue modelling. The BERT language model was applied to task-oriented dialogue modelling as NLU by Wu et al. (2020). DialoGPT – an open-domain chatbot, which uses the same architecture as GPT-2 language model, was designed by Zhang et al. (2020c). The model was trained on 147M conversation-like exchanges extracted from Reddit comment chains.

The transformer-based language models were also applied to task-oriented dialogue modelling. Budzianowski and Vulić (2019); Golovanov et al. (2019); Wolf et al. (2019b) used GPT-2 to model multi-domain task-oriented dialogues. Golovanov et al. (2019) explored the possibility of applying a language model to open-domain dialogues. Wolf et al. (2019a) published an open-domain dialogue model architecture called TransferTransfo. The training of TransferTransfo differs from GPT-2. It uses a weighted sum of two losses: the original cross-entropy loss used by GPT-2, and a next-utterance classification loss not unlike Devlin et al. (2019) (see Section 2.3.3). The final model showed strong improvements over seq2seq models. Later, Budzianowski and Vulić (2019) applied TransferTransfo to response generation in task-oriented dialogue modelling.

Recently, three similar approaches (Peng et al., 2020; Hosseini-Asl et al., 2020; Ham et al., 2020) used a pre-trained GPT-2 language model for end-to-end task-oriented dialogue modelling. The systems use two-stage decoding, where the belief state is decoded first, and then after passing the generated belief state and

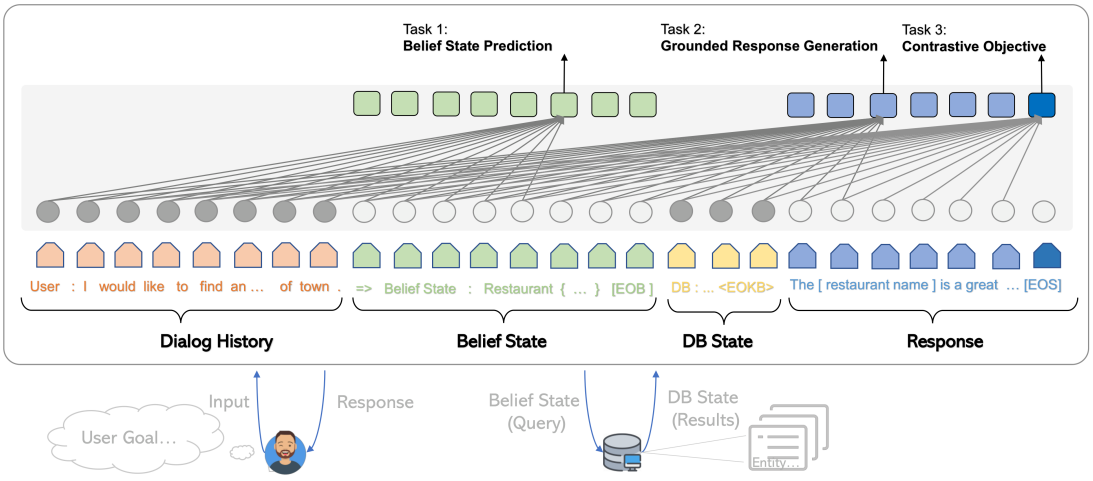

Figure 3.3: The architecture of the SOLOIST dialogue system. The system uses two-stage decoding. First, the dialogue context is passed as the input to the GPT-2 language model and the belief state is decoded in a string representation. Then the belief state is parsed and used as a query for the database. Finally, the dialogue context, the belief state, and the number of database results are passed as the input to the same GPT-2 LM, and the delexicalized response is decoded. The belief state prediction ('Task 1') and the response prediction ('Task 2') are trained using the *cross-entropy*. SOLOIST also uses an auxiliary task called *consistency detection* ('Task 3'). During the training, half of the samples are corrupted, and a binary classifier is trained to detect if the sample was corrupted. Source: Peng et al. (2020).

database results to the model, the final delexicalized response is generated. In order to be able to pass the belief state into the GPT-2 model, in their SOLOIST model, Peng et al. (2020) encoded the belief state into a string representation. Since the generated responses were delexicalized, the language model did not need the database results as its input. However, the number of database results had to be passes as the input to the model. In Figure 3.3, we can see the input and the output of the SOLOIST model. When the belief state is decoded, only the dialogue history is passed as the input. Then, for the final response, we can see, how the dialogue history, the belief state, and the number of database results are all concatenated and passed to the same language model to generate the delexicalized response. Peng et al. (2020) trained the belief state and response prediction using *cross-entropy* (see Section 2.2.1). The *cross-entropy* losses for the belief state and response prediction are denoted as 'Task 1' and 'Task 2' respectively in the figure. Furthermore, Peng et al. (2020) used an auxiliary task called the *consistency detection* in order to improve the training. The idea is similar to Devlin et al. (2019) (see Section 2.3.3). In the SOLOIST case, half of the training samples were corrupted by using a different randomly sampled belief state or response. A binary classifier, which shares the parameters with the rest of the network, was trained to detect corrupted samples. In Figure 3.3, this is denoted as 'Task 3'.

In this thesis, these prior approaches are extended, and data augmentation strategies based on paraphrasing are used. Paraphrasing by back-translations (Edunov et al., 2018; Federmann et al., 2019) showed success in the neural machine translation field (Edunov et al., 2018). In their work, Edunov et al. (2018) used a transformer-based model to back-translate the training corpus. Augmenting the training dataset by paraphrasing was already explored in the context of dialogue modelling (Jin et al., 2018; Einolghozati et al., 2019). Einolghozati et al. (2019) focused on improving robustness of NLU. The training corpus was translated to Spanish and Czech and back to English using an encoder-decoder model. Then, an RNN-based neural network was trained using the augmented data.

4. Method

In this chapter, we introduce a multi-domain task-oriented dialogue system called *AuGPT*. *AuGPT* extends an end-to-end dialogue system called SOLOIST (Peng et al., 2020). The same GPT-2 (Radford et al., 2019) language model is used to generate the *belief state* and the *response*. We start this chapter by formulating the dialogue modelling objective in Section 4.1.

In Section 4.2 we introduce the system's architecture and describe how the LM is trained and how the final response is generated. To be able to generate the belief state from an LM we need a string representation of the belief state, which is described in Section 4.3. When generating the response, the system has to incorporate the database results in the response. In Section 4.4, we propose two approaches to tackle the problem. The first one (used also by SOLOIST) follows Wen et al. (2015b) and trains the LM on *delexicalized responses* (see Section 4.4.1). The other approach, described in Section 4.4.2, passes the database results as the input to the language model.

In Section 4.5, Section 4.6, and Section 4.8 we give details on how the pre-trained GPT-2 model is fine-tuned. Whereas SOLOIST uses simple cross-entropy loss in the LM, we use the unlikelihood loss (Welleck et al., 2020; Li et al., 2020) for the response. To help the model train, auxiliary tasks are used (see Section 4.7). We use a *consistency detection* auxiliary task with some modifications from Peng et al. (2020). We also experiment with two novel auxiliary tasks: *user intent prediction* and *system action prediction*. The summarize, the main architectural differences between SOLOIST and our full AuGPT system are different *consistency detection* and other auxiliary tasks (described in Section 4.7, and different loss function and decoding strategies described in Section 4.5 and Section 4.8.

Finally, in Section 4.9 we describe how paraphrasing through back-translation is used to improve the system performance by augmenting the training datasets.

4.1 Dialogue modelling

A task-oriented dialogue system communicates with the *user*, responds to user input in order to fulfill a goal by using an external system, e.g., a database (DB) or an external API. In this work, we will use a database as the external system because it allows us to use automatic evaluation; however, the method applies to broader range of external systems including systems with side effects.

In a probabilistic dialogue system, we want to model the probability distribution $p(r|c)$, where r is the system response and c is the dialogue context, i.e., a concatenation of all previous utterances in the dialogue – both system's and user's. The training instances for an LM-based task-oriented dialogue system are tuples (c, b, d, r), where b is the system's belief state, which is also used for querying the database, and d are the database results.

The belief state represents the system's belief about the user's preferences (see Section 2.4). In the case of a multi-domain dialogue system, the belief state is a set of pairs (*domain name*, *domain belief*), where the *domain belief* is an assignment of values into slots, i.e., a set of pairs (*slot name*, *value*). An example of a belief state is given in Figure 4.2. Similarly, the database results d is a set of

pairs (*domain name*, *domain database results*), where the *domain database results* is an ordered list of entities returned by the database. We also define the *database result counts* d_c as the number of results in d for each domain.

First, let us rewrite the probability distribution $p(r|c)$ to model the interaction with an external database. The distribution can be factorized as follows:

$$\begin{aligned} p(r|c) &= \sum_d p(r|d,c)p(d|c) \\ &= \sum_d \sum_b p(r|d,b,c)p(d|b)p(b|c) \\ &= \sum_b p(r|\mathit{Query}(b),b,c)p(b|c)\,, \end{aligned} \tag{4.1}$$

where $p(d|b)$ is a deterministic 'one-hot' distribution over the database results, and *Query* is a function returning database results. Therefore, if we model both distributions $p(r|d,b,c)$ and $p(b|c)$, we get a dialogue system which is able to query an external database and incorporate the results in its reponses.

4.2 AuGPT system architecture

The essential part of our dialogue system is a pre-trained GPT-2 (Radford et al., 2019), which is used as an LM for generating both the belief state and the response. To be able to generate the belief state from an LM, we encode the belief state into a string representation. Similarly, encoding the database results into a string representation allows us to pass it directly into the language model. Let $\hat{p}$ be our LM. Formally, we use the same $\hat{p}$ to approximate the belief state prediction and the response prediction distributions:

$$p(\bar{r}|d_c,b,c) \approx \hat{p}(\bar{r}|d_c,b,c,\theta) \tag{4.2}$$

$$p(b|c) \approx \hat{p}(b|\emptyset,\emptyset,c,\theta)\,, \tag{4.3}$$

where we denote the model's parameters as θ.

The main variant of our dialogue system called *AuGPT* does not generate the final responses directly, but generates *delexicalized responses* where we replace all concrete slot values with placeholders. After the delexicalized response is generated, we can replace the placeholders with correct slot values from the database. The motivation is that the language model does not need the database results as its input, but it is enough to pass the number of database results per each domain – *database result counts* (denoted as d_c) – as a part of the input. More details are given in Section 4.4.

For our LM, we use the GPT-2-small configuration. First, the previous dialogue context is passed to the language model, then the LM autoregressively generates the string representation of the belief state, which is parsed and passed as the query to the database. The database returns the database results, and the database result counts string is concatenated with dialogue context and belief state string and passed again to the same LM (see Section 4.3). Finally, the delexicalized response is autoregressively generated by the language model and lexicalized using the database results from the previous step. The whole process is visualized in Figure 4.1.

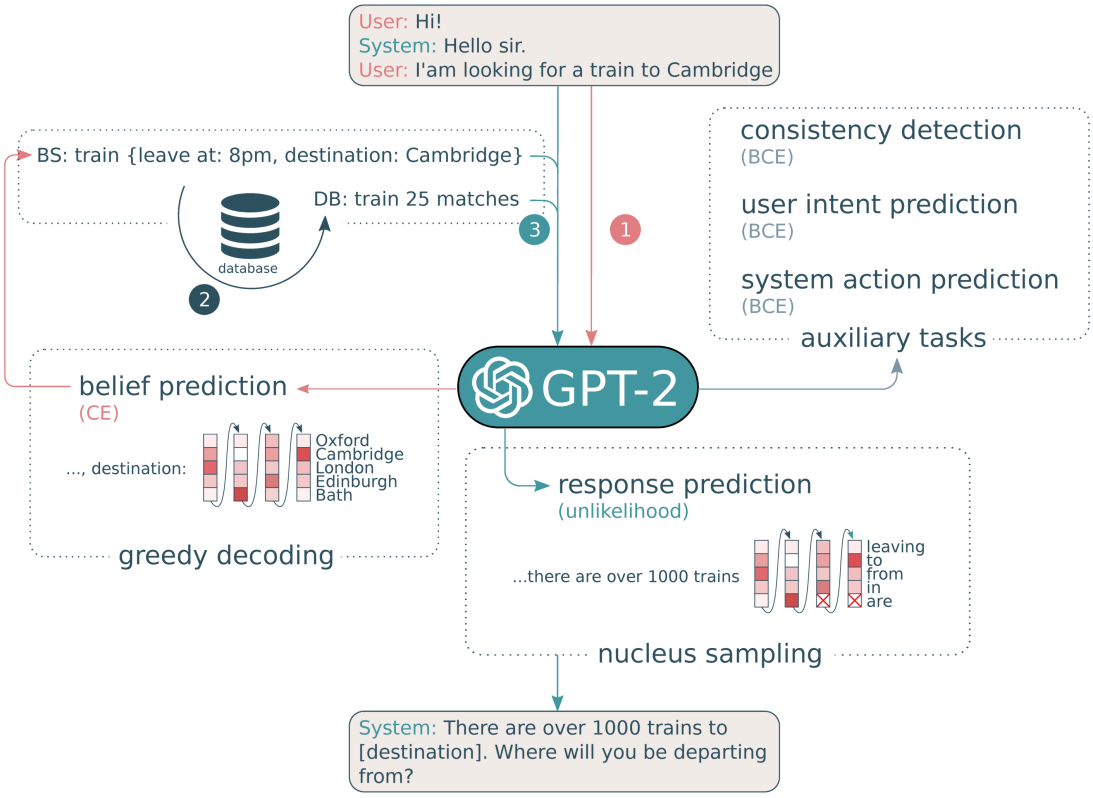

Figure 4.1: The architecture of AuGPT. The pipeline runs in two stages. First, a fine-tuned GPT-2 LM is used to predict the belief state. This is denoted as step **1** in the figure. Then the database results are obtained (step **2**) and everything is passed to the GPT-2 again to predict the final delexicalized response (step **3**). Auxiliary tasks are used to improve the training. The final AuGPT model uses only the belief state consistency detection auxiliary task, however, we also experiment with the user intent prediction and the system action prediction. Unlikelihood training is used for the response prediction, whereas cross-entropy is used for the belief state prediction. The pictures in 'belief prediction' and 'response prediction' boxes denote the autoregressive decoding, where the probability distribution over each token uses previously generated tokens. In unlikelihood training, token repeating is discouraged by the loss function, which is denoted by crosses.

Belief state:
train { leave at=15:30, arrive by=17:15 }, hotel { price range = cheap }
DB: train 23 matches, hotel no match

Figure 4.2: String format for AuGPT's belief state and database result counts.

4.3 Belief state & database result counts

Since the same model is used for both the response and belief state generation, we want to reuse as many tokens as possible, i.e., make the string representation of the belief state and database result counts close to the natural language. Furthermore, our original language model was pre-trained on large natural language text corpora and choosing this representation makes it easier for the model to learn to generate the belief state.

The representation of the belief state and the database result counts is illustrated in Figure 4.2. The string representation of the belief state is generated as follows:

1. The belief state is sorted. The active domain is the first, followed by other domains in lexicographical order.
2. The string representation of each slot-value pair for each domain is generated as `[slot] = [value]`, where both the slot and the value is chosen as a natural language text. E.g., instead of `leaveAt` we use `leave at`.
3. All slot-value strings for each domain are sorted lexicographically by the slot name.
4. All slot-value strings for each domains are concatenated with `,` used as the separator.
5. The string representation of each domain is generated as: `[domain name] = { [concatenated slot-value strings] }`.
6. The final representation is generated by concatenating all domain string representations with `,` used as the separator.

For the database result counts, we generate the string representation as follows:

1. The database result counts are ordered with the same order as the belief state.
2. The string representation of each domain-count pair is generated based on the count, where for zero results, we use `[domain] no match`, for a single result, we use `[domain] 1 match`, and for everything else, we use `[domain] [count] matches`.
3. The final representation is generated by concatenating all domain string representations with `,` used as the separator.

The reason for the first rule used to generate the string representation of the belief state is that we need to sort the belief state to make the representation unique. We also need the model to output the active domain for the response (see Section 4.4). Therefore, we order the domains of the belief state so that the active domain is the first one, followed by other domains in lexicographical order. During the evaluation, we generate the belief state and use the first outputted domain as the active domain for the response. The disadvantage of this approach is that we cannot determine the active domain if the belief state is empty. However, in such a case, the lexicalization would fail anyway due to the database results being empty, and, therefore, the system performance is not affected by this decision.

4.4 Lexicalization

When generating the response, it is essential that the system uses the database results. Unfortunately, this is not always guaranteed when a powerful language model is used. Especially on a small dataset, the LM can learn to generate the final responses as if they were returned from the database. We provide two orthogonal approaches to the problem.

In Section 4.4.1 we describe the same technique as other end-to-end task-oriented dialogue systems (Peng et al., 2020; Hosseini-Asl et al., 2020; Ham et al., 2020). Before training, the slot values, e.g., hotel names, in the response are replaced with placeholders such as `[name]`. We call this process *delexicalization* and the resulting responses *delexicalized responses*. In inference mode, we let the model generate the delexicalized response and an inverse process called *lexicalization* is used to replace the placeholders with slot values from the database. This approach is used in the system that we call *AuGPT*.

Unfortunately, the rule-based lexicalization is not perfect, and the generated responses are sometimes grammatically or factually incorrect. An alternative approach, described in Section 4.4.2 lets the model generate the final responses without using rule-based *lexicalization* and *delexicalization*. In order to be able to incorporate the database results into its response, the database results have to be passed as input to the language model.

4.4.1 Using delexicalized responses

If we train an LM for the $p(r|d, b, c)$ distribution, it is difficult for the model to generalize past the training set. Task-oriented dialogue datasets are usually small (compared to general text datasets used to train LMs), and responses often contain underrepresented, sometimes unique words, such as reference numbers, restaurant names, etc. Ideally, the model would need to learn to copy the database results into the response. Imagine, for example, that the database contains names of restaurants, and the dialogue system is asked to pick a restaurant. The names of restaurants could be arbitrary, and lots of them will not be included in the training set. This is a problem for the language model because if it maximizes the likelihood on the training set, it may never learn to copy the name of the restaurant from the database results.

In order to resolve this problem and force our dialogue system to copy the database results into its responses, we use *delexicalized responses* (Wen et al.,

delexicalized response:
There are multiple restaurants in the area. We recommend `[name]` or `[name]` with postcodes `[postcode]` and `[poscode]` respectively.

lexicalized response:
There are multiple restaurants in the area. We recommend *Curry Garden*[name] or *Gastroplace*[name] with postcodes *CB259AQ*[postcode] and *CB23JX*[poscode] respectively.

Figure 4.3: Example of a delexicalized response, where the lexicalization fails. Note multiple occurences of placeholders `name` and `price`. The *lexicalize* function is not able to correctly pair the database results with placeholders.

2015b), where all strings which correspond to parts of database results (slot values) are replaced with placeholders such as `[name]`, `[price]`. This forces the dialogue system to copy the database results into its responses because, during lexicalization, the placeholders are replaced with slot values from the database results. Furthermore, the sparsity of the training dataset is decreased because underrepresented and unique words occur much less often. We define two functions: *lexicalize* and *delexicalize*. *Delexicalize* takes as the input a system response r from the training dataset and deterministically replaces all slot values with corresponding tokens. The resulting delexicalized response $\bar{r}$ does not depend on the database results, but only on their counts d_c. *Lexicalize*, on the other hand, takes as the input the delexicalized response $\bar{r}$ and the database results d and deterministically produces ideally the original response r. If we assume perfect lexicalization, i.e., by lexicalizing a delexicalized response, we obtain the original response, we have $p(r|c) = p(\bar{r}|c)$ and $p(r|d, b, c) = p(\bar{r}|d_c, b, c)$ and we can apply our original dialogue objective unchanged. In fact, this assumption holds in most cases on the MultiWOZ 2.1 dataset (see Chapter 5). However, there are some cases, where this assumption fails, one of which is demonstrated in Figure 4.3.

Unlike Budzianowski et al. (2018) and Eric et al. (2020), our placeholders used in the delexicalized responses do not contain domain names. In order to lexicalize the generated response back, we need to know the domain for each placeholder to use the correct domain from the database results. While dialogues usually span multiple domains, in a single response, only one domain is used, e.g., the system never responds with prices of restaurants and train connections in a single response. We will call this single domain the *active domain* for the turn. For our lexicalization to work, we need to know the active domain in order to associate the placeholders with correct database results domain. In the Section 4.3, we will describe how the active domain is inferred from the belief state.

4.4.2 Generating lexicalized responses

While using delexicalized responses works in most cases, we have identified rule-based lexicalization as one of the main cause of errors (see Section 6.7). Furthermore, the process of designing the rule-based delexicalization and lexicalization is labour-intensive and prone to errors. Therefore, we decided to explore an alternative approach and let the language model generate the final responses. The

example 1:
train (78) {arrive by = 20:23, 19:23, 18:23, ...; id = TR4977, TR7883, ...; leave at = 17:40, 16:40, 15:40, ...; price = 375.50 pounds}, hotel (23)

example 2:
restaurant (33) address = 106 Regent Street City Centre, ...; area = centre (33); food = indian (6); name = Curry Garden, Stazione Restaurant and Coffee Bar, ...; phone = 01223302330; postcode = CB21DP; price range = expensive (33); type = restaurant (33)

Figure 4.4: Examples of formatted database results used when generating the lexicalized responses directly by the language model. The database results in **example 1** contain two domains: train (active) and hotel. In this example, all slots are of the *text* type. In **example 2**, we can see a mix of *categorical* and *textual* slots demonstrated on the restaurant domain. While *price range*, *type*, *food*, and *area* slots are categorical, the rest of the slots are textual.

model itself is expected to copy the slot values from its input to the generated response.

In order for the model to be able to generate the responses, the model has to have access to the full database results instead of using only the counts. The results, therefore, have to be formatted as a string before they are passed into the model. Unfortunately, the transformer architecture has restrictions on the string length, and full database results would be too long for the model. The formatted database results have to be relatively short because otherwise, a big part of the dialogue context or even the belief state would have to be cropped, and the system would not be able to read it. Since only a limited number of results can be displayed, we have to sort the database results first. The simplest approach is to use the order in which the results are returned from the database. One exception has to be made in the case of the train domain. When the user specifies the *arrive by* slot, the results are expected to be sorted from the latest train, which arrives right before the specified time, to the earliest one.

Since the system uses exclusively the results from the *active domain*, only the *active domain* has to be displayed in more detail. For all other domains in the database results, it is enough to return the number of entities. We split the columns in the database into two categories: categorical and text. The categorical assigns the entity into a category such as food type, star rating, etc. A text type could be the name of the restaurant, its address, etc. For the categorical columns, we return the first n most frequent categories with their count. The number of the categories displayed (n) was designed so that the language model would have all the information it needs in most responses, while the representation is kept relatively short. The number is different for each column, but is usually less than three. More details on the formatting of different columns are given in Appendix A.1. An example of the formatted database results can be seen in Figure 4.4.

4.5 Language model losses

GPT-2 is a *causal* LM, which means that it sequentially generates the next token in the sequence based on the history. During training, *teacher forcing* is used to train the model, i.e., the ground-truth labels from prior steps are used as the input to the model. For the belief state predictor, we use cross-entropy loss defined as follows:

$$\mathcal{L}_{bs} = -\mathbb{E}_{(c,b)\sim\mathcal{D}}\Big[\frac{1}{|b|}\sum_{i=1}^{|b|}\log \hat{p}(b_i|\boldsymbol{b}_{<i}, \boldsymbol{c}, \boldsymbol{\theta})\Big] \tag{4.4}$$

For the response predictor, we also use the cross-entropy loss, but we also add the unlikelihood penalty (Welleck et al., 2020; Li et al., 2020) to the total training objective:

$$\mathcal{L}_{res} = -\mathbb{E}_{(c,b,d,r)\sim\mathcal{D}}\Big[\frac{1}{|\bar{r}|}\sum_{i=1}^{|\bar{r}|}\log \hat{p}(\bar{r}_i|\bar{\boldsymbol{r}}_{<i}, \boldsymbol{d_c}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{\theta})\Big] \tag{4.5}$$

$$\mathcal{L}_{unlike} = -\mathbb{E}_{(c,b,d,r)\sim\mathcal{D}}\Big[\frac{1}{|\bar{r}|}\sum_{i=1}^{|\bar{r}|}\sum_{c\in\{\bar{r}_1,\ldots,\bar{r}_{i-1}\}}\log(1-\hat{p}(c|\bar{\boldsymbol{r}}_{<i}, \boldsymbol{d_c}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{\theta}))\Big] \tag{4.6}$$

The unlikelihood term penalizes previously seen tokens either generated by the model or part of the input. This increases the diversity of the generated output and should help with repeated tokens (Welleck et al., 2020).

4.6 Using pre-trained GPT-2

In order to effectively transfer the pre-trained GPT-2 language model (Radford et al., 2019) to our domain, we needed to make our input representation as close as possible to the GPT-2's representation. We reuse the GPT-2's token embeddings, but we add additional tokens to the tokenizer's dictionary and add randomly initialized rows to the embedding matrix for the newly added tokens. In particular, we add tokens: `<eob>`, `<eokb>`, that separate the dialogue context from the belief state and the belief state from the database result counts, respectively. To separate the dialogue context from the belief state, we use the token `=>`, which is already present in the tokenizer's dictionary. The belief state and the database result counts consists of natural language tokens, and therefore, can be passed as the input to the model and share the same token embeddings with dialogue context. This means that the same token, e.g., `no`, will use the same embedding vector regardless of whether it is in the belief state or in the dialogue context.

4.7 Auxiliary tasks

We also employ additional auxiliary tasks to help the model learn a better internal representation from the dataset, similarly to Devlin et al. (2019) and Peng et al. (2020). We have experimented with three auxiliary tasks: *consistency detection*, *user intent prediction* and *system action prediction*. In our experiments (see Chapter 5), however, user intent and system action prediction did not improve the performance much and, therefore, only the consistency detection is included in the final AuGPT model.

4.7.1 Consistency detection

In the consistency detection auxiliary task, we corrupt half of the training samples and train a binary classifier to detect whether a sample was corrupted or not. The binary classifier uses the hidden representation of the last response token of GPT-2 backbone (output of the last transformer layer) and applies an affine transformation of these features with a weight matrix $W^{(c)} \in \boldsymbol{\theta}$ and bias $b^{(c)} \in \boldsymbol{\theta}$. Let $f_c(\boldsymbol{x}|\boldsymbol{\theta})$ be the last hidden features of the last response token for a sample $\boldsymbol{x}$. The classifier uses the binary cross entropy loss given as follows:

$$\begin{aligned}\mathcal{L}_c = &-\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\Big[\log\sigma(W^{(c)}f_c(\boldsymbol{x}|\boldsymbol{\theta}) + b^{(c)})\Big] \\ &-\mathbb{E}_{\boldsymbol{x}\sim\tilde{\mathcal{D}}}\Big[\log(1-\sigma(W^{(c)}f_c(\boldsymbol{x}|\boldsymbol{\theta}) + b^{(c)}))\Big]\,,\end{aligned} \tag{4.7}$$

where $\sigma(\cdot)$ is the sigmoid function and $\tilde{\mathcal{D}}$ is the corrupted dataset. In each training batch, we corrupt half of the samples by randomly applying one or more of the following changes with the same probability:

1. The belief state b is replaced with another belief state, sampled uniformly randomly from the training dataset.

2. The delexicalized response $\bar{r}$ is replaced with a different one sampled uniformly randomly from the training dataset. If this change is applied in combination with the belief state resampling, the delexicalized response and the belief state are taken from the same random sample.

3. Each value in all slot-value pairs of the belief state is resampled uniformly randomly from a set of valid values for the given domain-slot pair – obtained from the training data. In this case, the domain names and domains order are unchanged and the active domain stays the same.

Note that Peng et al. (2020) uses the first two changes. The third one is novel, and we found it to be very useful in the context of multiple domains. The consistency detection task as proposed by Peng et al. (2020) is easy for the model to learn since, in most cases, it is sufficient to detect if the domains in the belief state agree with the dialogue context. It implies that the original task provides good gradients only at the beginning of the training, but after that, it saturates. Our consistency detection task, on the other hand, is much more difficult for the model to learn and provides better gradients throughout the training process. Since the model has to learn if the dialogue context agrees with the values in the belief state, it has to attend to the relevant information in the dialogue context, which helps the belief state predictor.

4.7.2 User intent & system action prediction

We experiment with two additional classifiers predicting the user intent and the system action. For each dialogue context, multiple user intents are possible, as well as multiple system actions. Therefore, the user intent and system action classifiers are trained as binary classifiers predicting each user intent or system action independently. The input to the user intent classifier is the last hidden representation of the last dialogue context token, and the input to the system

action classifier is the last hidden representation of the last database counts token. Both classifiers apply an affine transformation of the input features and sigmoid function as in logistic regression similarly to the consistency detection auxiliary task described in Section 4.7.1.

Let $W^{(u)}$ and $W^{(a)}$ be weight matrices of user intent and system action classifiers and $\boldsymbol{b^{(u)}}$ and $\boldsymbol{b^{(a)}}$ their biases. Let $f_u(\boldsymbol{x}|\boldsymbol{\theta})$ and $f_a(\boldsymbol{x}|\boldsymbol{\theta})$ be the last hidden features of the last dialogue context token and the last database counts token for a sample $\boldsymbol{x}$. Also, let $\mathcal{A}_u$ and $\mathcal{A}_a$ be the sets of all possible user intents and all system actions, and $A_u \subseteq \mathcal{A}_u$, $A_a \subseteq \mathcal{A}_a$ be the sets of user intents and system actions for the sample $\boldsymbol{x}$. Then the user intent and the system action auxiliary losses $\mathcal{L}_u$ and $\mathcal{L}_a$ are given as follows:

$$\begin{aligned}\mathcal{L}_u = -\mathbb{E}_{(A_u,\boldsymbol{x})\sim\mathcal{D}}\Big[&\sum_{i\in A_u} \log\sigma(W^{(u)}_{i,\cdot} f_u(\boldsymbol{x}|\boldsymbol{\theta}) + b^{(u)}_i) \\ &- \sum_{i\in\mathcal{A}_u\setminus A_u} \log(1-\sigma(W^{(u)}_{i,\cdot} f_u(\boldsymbol{x}|\boldsymbol{\theta}) + b^{(u)}_i))\Big]\end{aligned} \tag{4.8}$$

$$\begin{aligned}\mathcal{L}_a = -\mathbb{E}_{(A_a,\boldsymbol{x})\sim\mathcal{D}}\Big[&\sum_{i\in A_a} \log\sigma(W^{(a)}_{i,\cdot} f_a(\boldsymbol{x}|\boldsymbol{\theta}) + b^{(a)}_i) \\ &- \sum_{i\in\mathcal{A}_a\setminus A_a} \log(1-\sigma(W^{(a)}_{i,\cdot} f_a(\boldsymbol{x}|\boldsymbol{\theta}) + b^{(a)}_i))\Big]\end{aligned} \tag{4.9}$$

4.8 Model training & inference

In order to train the model, the total loss is computed as the sum of all individual losses:

$$\mathcal{L} = \mathcal{L}_{bs} + \mathcal{L}_{res} + \mathcal{L}_{unlike} + \mathcal{L}_c\,, \tag{4.10}$$

optionally adding also $\mathcal{L}_u$ and $\mathcal{L}_a$. The Adam optimizer (Kingma and Ba, 2014) is used to update the model's parameters $\boldsymbol{\theta}$.

When the model is used in inference mode, first, the dialogue context is passed to the model, and the belief state string is autoregressively decoded. We use greedy decoding for the belief state. The belief state string is concatenated with the dialogue context, and database result counts are added at the end, and the whole string is again passed to the same model. This time, the delexicalized response is autoregressively decoded. We use nucleus sampling (Holtzman et al., 2020) for decoding the delexicalized response. We found nucleus sampling useful for generating the response since it increases diversity, but greedy decoding is preferred for the belief state with a fixed structure. The whole training and inference pipeline is shown in Figure 4.1.

4.9 Augmenting the training dataset

Transformer-based large language models (Radford et al., 2019; Devlin et al., 2019) have achieved extraordinary success in the NLP domain, possibly because of the large ammounts of training data they use for training. Since the number of parameters of these models is very high, there is an increased risk of overfitting the training data if a smaller dataset was used. Unfortunately, in the dialogue

modelling domain, the ammount of training data available is much smaller compared to general language modelling. Therefore, it is more challenging to apply large transformer-based models, which require a lot of data.

To tackle the issue, we propose to increase the size of the training dataset by paraphrasing all user's and systems's utterances. To generate the paraphrases automatically, we use back-translation – each utterance in the training dataset is translated to another language and then translated back to English, generating different surface forms. Using back-translation was motivated by successful usage of synthetic data in a lot of NLP tasks (Sennrich et al., 2016; Konstas et al., 2017; Elder et al., 2020). In our setup, multiple paraphrases are generated for each training utterance. During training, the dialogue context is built by independently sampling each utterance uniformly at random from the set of all variants, including the original one. This process effectively increases the variability of the data.

We use a trained multilingual machine translation model (Macháček et al., 2020; Edunov et al., 2018) to paraphrase the data. More specifically, we used the translation system available at our department (Macháček et al., 2020) as a part of the ELITR project.[1] The original utterance is translated into ten different intermediate languages to obtain a set of different paraphrases. In particular, the following languages were chosen: Arabic, Bulgarian, Bosnian, German, Spanish, French, Russian, Slovakian, Swedish, Albanian. The languages were chosen from the set of 40 languages for which the training data were available, and the chosen languages were the ones with the least errors judged on few sampled utterances. Note that the accuracy of the back-translation is far from optimal. Sometimes the sentence has a different or even opposite meaning. Also, the names of hotels, restaurants, etc., are prone to errors in the back-translation process. Nevertheless, using these noisy training data increases the system performance, which we show experimentally in Section 6.3.

[1] https://elitr.eu/

5. Experiments

A series of experiments were conducted to evaluate the proposed dialogue system. In this chapter, we give details on training and experimental setup as well as details on the evaluation methods. The results of all experiments are presented in Chapter 6.

In Section 5.1, we describe the datasets that were used for training and evaluation of AuGPT. The main focus of this thesis was to use the MultiWOZ dataset (Budzianowski et al., 2018). Unfortunately, the dataset annotation is rather noisy and, therefore, in Section 5.1.1 we explain how we filtered the training dataset. The resulting subset is called *clean samples* in the rest of the thesis. Section 5.2 gives some details on hyperparameters used for the training.

In order to measure the performance of our AuGPT dialogue system, we used automatic as well as human evaluations. One approach to evaluating end-to-end dialogue systems is to use an user simulator – another engineered dialogue system that emulates the user. Section 5.3 describes how the ConvLab 2 (Zhu et al., 2020) user simulator was used to evaluate the system. An alternative, described in Section 5.4, is to use a corpus-based evaluation, where we generate each individual response conditioned on the ground-truth dialogue context – each response is considered individually. First, in Section 5.4.1, we use the metrics proposed by Budzianowski et al. (2018) to be able to compare with state-of-the-art approaches. Then, we use an alternative evaluation (in Section 5.4.2), which measures the quality of generated belief states (see Section 4.2), as well as the quality of generated responses using the BLEU score (Papineni et al., 2002) and the ROUGE metric (Lin, 2004). We experiment with different variants of the system and carefully evaluate each contribution through a series of ablation experiments described in Section 5.6. Furthermore, we conduct a detailed error analysis with both quantitative and qualitative results in Section 5.7.

A variant of the AuGPT dialogue system competed in the DSTC 9 end-to-end dialogue system challenge (Gunasekara et al., 2020). The details of the competition are given in Section 5.5. After the competition was over, we continued the work on the dialogue system. Instead of relying on the error-prone delexicalization, we proposed to decode the lexicalized responses directly (see Section 4.4.2). The details of these experiments are given in Section 5.8.

The author of this thesis implemented main parts of the training and evaluation code: the model, training scripts, the rest of the pipeline, including the lexicalization and delexicalization, and the ConvLab 2 and MultiWOZ evaluations (see Section 4.2). His colleagues helped with writing the code for data pre-processing and dataset loading. In total, the author of this thesis authored 93% of the code.[1]

5.1 Datasets

We used multiple datasets for the training of the AuGPT dialogue system and for the evaluation and comparisons. However, we focused mainly on the MultiWOZ 2.1 dataset (Eric et al., 2020). The MultiWOZ dataset (Budzianowski et al., 2018;

[1]The source code is publicly available at `https://github.com/ufal/augpt`. It is also attached as a part of the thesis (see Appendix A.2).

Eric et al., 2020) contain seven distinct domains related to tourist information: hotel information, restaurant booking, train connections, bus connections, hospital information, police station information, taxi booking. The dataset was collected using the Wizard of Oz method (Kelley, 1984) – crowdsourced workers played the roles of the user and the system and interacted with each other to build the dialogues. The workers playing the role of the user were given a goal, e.g., to book a train with a specific set of constraints such as the time of departure, and by interacting with other worker playing the role of the system, they tried to fulfil the goal. An example of the dialogue goal can be seen in Section 6.7.3. The workers playing the role of the system had access to the database and responded accordingly. Later, crowdsourced workers also annotated each turn of the dialogue with appropriate dialogue acts. In total, there are 10 438 dialogues in the dataset, and 7 032 of these dialogues are multi-domain.[2] MultiWOZ 2.1 is an improved version of the original MultiWOZ 2.0 dataset (Budzianowski et al., 2018). The main difference is that in the newer version, there are span annotations, and some errors in the labels were corrected.

Since the MultiWOZ dataset is relatively small, we pre-train our large model on other task-oriented dialogue datasets. We believe that this task-adaptive pre-training on a dataset more similar to the final one is beneficial for the speed of convergence and the final accuracy. This claim is supported by experiments shown in Section 6.3. We combine two datasets for the pre-training – Taskmaster-1 (Byrne et al., 2019) and Schema-Guided Dialogue (Rastogi et al., 2020). Both datasets are multi-domain, task-oriented, dialogue datasets consisting of 12 215 and 22 825 dialogues, respectively. Two procedures were used to create the Taskmaster-1 dataset: Wizard of Oz (Kelley, 1984) and self-dialogue methods (Byrne et al., 2019). In the Wizard of Oz approach, crowdsourced workers interacted with trained agents, whereas in the self-dialogue approach, the dialogues were written entirely by crowdsourced workers. Machine-generated utterances were used in the collection Schema-Guided Dialogue, which were later paraphrased by human annotators to obtain more realistic dialogues.

5.1.1 Dataset cleaning

Although the MultiWOZ 2.1 dataset was generated by human-to-human interactions and, therefore, should contain more realistic dialogues than datasets such as Schema-Guided Dialogue, it has a lot of errors in its labels and contains a lot of inconsistencies. Some of the most frequent errors are the following:

- The dialogue goal annotation is often not aligned with the real goal of the dialogue. During the collection process, the crowdsourced worker playing the role of the user was asked to book an entity with a set of constraints (see Section 5.1); however, sometimes, not all constraints were fulfilled.
- Even if all constraints were fulfilled, the belief state sometimes does not reflect all constraints because the worker playing the role of the user did not ensure that all constraints were fulfilled. For example, the worker-user was supposed to find a hotel with a free wifi. They asked the system for a hotel, and the system (another worker) returned a hotel that had a free wifi, but

[2]The numbers of dialogues are given for the MultiWOZ 2.1 dataset (Eric et al., 2020).

the user-worker did not ensure that this was the case – did not ask if the hotel had a free wifi.

- The entity returned by the worker playing the role of the system does not correspond to the belief state due to the worker's mistake in interacting with the database. The worker-system sometimes does not query the database correctly and reports different results.

- There are typos in the belief state. Particularly, the names of restaurants and hotels sometimes contain errors.

Therefore, it could be beneficial to use only a portion of the original dataset which does not contain errors. The increase in the quality of the training samples might outweigh the loss in quantity. To verify this hypothesis, we compared the model's performance on the original unfiltered dataset with a model trained only on a subset of the dataset called *clean samples*. The results of the experiment can be seen in Section 6.3. We have chosen *clean samples* as the dialogues where the dialogue goal corresponds to the turn-level annotated data. We used the following rule to determine this: if the database results obtained by using the final annotated belief state were consistent with the dialogue goal entity and if all information in the goal was also included in the belief state at some point in time,[3] the dialogue was added to clean samples. Note that the filtering process uses the same set of rules as the MultiWOZ evaluation (Budzianowski et al., 2018), i.e., the *clean samples* dialogue subset would have a perfect performance in terms of MultiWOZ metrics – the inform and the success rates (described in Section 5.4) equal to one. The size of the clean samples subset is about 70% the size of the original dataset.

5.1.2 Combining training datasets

Although GPT-2 model (Radford et al., 2019), which is used as the backbone architecture, generates high-quality natural language texts, due to its large size, we need a lot of training data to prevent overfitting. As motivated in Section 5.1, due to the relatively small size of the MultiWOZ dataset, we decided to first pre-train the GPT-2 model[4] on combined Taskmaster-1 and Schema-Guided Dialogue datasets. However, in order for the pre-training to be effective, the pre-training datasets had to be made as close as possible to the fine-tuning dataset – MultiWOZ 2.0/2.1. Therefore, we changed the belief state representations by unifying domain and slot names to make them consistent across datasets. In addition, the same slot names were used in the delexicalization (see Section 4.4). In the case of both Taskmaster-1 and Schema-Guided Dialogue, there were a lot of domains which we decided to unify into a single domain. For example we unified the following domains:

- `flight_search`
- `flight1_detail`

[3] We also require the presence of the *name* slot even if it is not in the dialogue goal, which is a part of the MultiWOZ evaluation code.

[4] We did not initialize GPT-2 from scratch but used the model pre-trained on English texts (Radford et al., 2019).

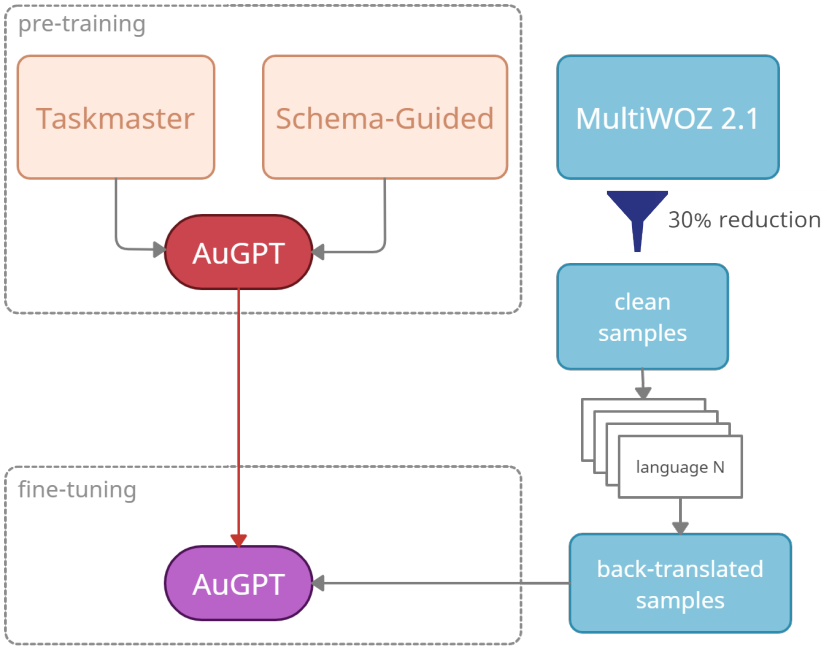

Figure 5.1: This diagram visualizes how the final AuGPT model was obtained. First the pre-trained GPT-2 model was trained on the concatenation of Taskmaster-1 and Schema-Guided Dialogue datasets. Then it was finetuned on clean samples from the MultiWOZ 2.1 dataset, that were augmented via backtranslations.

- `flight2_detail`
- `flight3_detail`
- `flight4_detail`
- `flight_booked`

into a single domain called `flight`. We also renamed slot names to have the same meaning as in the MultiWOZ dataset. For example, we used the following transformations:

- `destination1` → `destination`
- `destination2` → `destination`
- `origin` → `departure`
- `from` → `leave at`
- `to` → `arrive by`
- ...

The final ontology that unifies all three datasets contains 22 domains and 135 slots. When belief state annotation was not available, as was the case with Taskmaster-1, we used accumulated domain-slot-value NLU annotations from previous turns.

5.2 Training the model

In this section, we give more details on the model architecture described in Section 4.2. In all experiments, the HuggingFace Transformers (Wolf et al., 2019a)

implementation is used for GPT-2 model. All code is written for the PyTorch framework (Paszke et al., 2019). Our model extends the *small* variant of the GPT-2 model (Peng et al., 2020) and consists of 12 transformer blocks with a layer size of 768. The number of heads in each attention layer is 12. In total, the model has appropriately 124 million parameters. A dropout rate of 0.1 is used (Srivastava et al., 2014), and, furthermore, for auxiliary tasks, we use a label smoothing of 0.1 (Müller et al., 2019). The model is optimized using the Adam optimizer (Kingma and Ba, 2014). To find the optimal set of hyperparameters, we have conducted a limited exploratory search starting from sensible defaults (Peng et al., 2020). In order to increase the training efficiency, we use mixed-precision training (Micikevicius et al., 2018), which uses the float16 precision for intermediate computed values and the float32 precision for the model parameters. We use PyTorch AMP, which rescales the loss to keep gradients in valid ranges.

During training, we crop the input sequence to the maximum length of 512 tokens. The models are pre-trained for eight epochs on the combined Schema-Guided Dialogue and Taskmaster-1 dataset and fine-tuned on eight epochs on the MultiWOZ dataset. If clean samples are used instead of the full dataset, the corresponding number of epochs is used to keep the total number of gradient update steps the same. We use the Adam optimizer (Kingma and Ba, 2014) with a constant learning rate 5×10^{-4} and no weight decay. The model is trained on four NVIDIA V100 GPUs with a total batch size of 16, and the training takes less than one day.

5.3 Simulated user evaluation

For automatic evaluation with a simulated user (Schatzmann et al., 2006), we use the ConvLab 2 platform (Zhu et al., 2020). To represent a simulated user, the platform uses a classical dialogue pipeline consisting of NLU, DST, policy, and NLG. NLU component uses fine-tuned BERT language model (Devlin et al., 2019). All other components are rule-based.

To evaluate our system, we simulate 1000 conversations between the system and the simulated user agent. The ConvLab 2 agent mimics the user's behaviour and interacts with the system in order to complete a randomly sampled goal. Furthermore, it evaluates the system performance and computes multiple metrics such as *complete rate*, *success rate*, and *book rate*. The *complete rate* measures the number of successfully completed dialogues, where all user's requests have been met. The *success rate* then computes the percentage of dialogues which are *complete* and, furthermore, in which all user's constraints are captured by the system, and a valid booking number is provided by the system if requested. The *book rate* is then the percentage of dialogues where the system booked the correct entity, e.g., a hotel or a restaurant, if booking an entity was part of the agent's goal. Apart from these metrics, we also compute *precision*, *recall*, and *F1 score*. There metrics are computed on generated *domain-slot-value* tuples in the belief state over all turns. Finally, the average number of turns per dialogue are also reported.

5.4 Automated corpus evaluation

In order to evaluate the system performance, in the automated corpus evaluation, the evaluation process generates each individual response conditioned on the ground-truth dialogue context. Therefore, no user simulator is needed for the evaluation process. On the other hand, the computed performance does not properly measure the true quality of the system. The reason is that in the corpus-based evaluation, only one turn at a time is considered, whereas, in real dialogue, the error accumulates – if the dialogue system generates one response wrong, the whole dialogue may fail. There is another problem with the corpus-based evaluation. Imagine, for example, a system that needs to fill multiple slot values before it applies an action. Theoretically, for this system, the order in which it collects the information is not important for the success of the dialogue. Unfortunately, for the corpus-based evaluation, the order is constant, given by the ground-truth dialogue.

First, in Section 5.4.1, we describe the original metrics proposed by (Budzianowski et al., 2018) to measure the performance of task-oriented dialogue systems. Then, in Section 5.4.2, we explain a similar set of metrics that allow us to analyze the performance of different parts of our dialogue system.

5.4.1 MultiWOZ evaluation

In order to compare our approach with prior work on the MultiWOZ dataset, we use the MultiWOZ evaluation proposed by Budzianowski et al. (2018). It consists of the following automated metrics: *inform rate*, *success rate* (Budzianowski et al., 2018), and *BLEU* (Papineni et al., 2002). While the former two metrics evaluate the quality of the dialogue state tracking component, the *BLEU* metric evaluates the performance of the NLG component.

The *inform rate* is the percentage of dialogues in which the system provided an appropriate entity – an entity that fulfils all the required constraints. Furthermore, the system has to output the name of the entity explicitly. If the system makes a booking with the correct entity and outputs all required information, it could still have the inform rate 0 because the name was not outputted by the system. The *success rate* is the percentage of dialogues in which the inform rate is 1, and in which the system outputted all the requested information. Finally, the BLEU score (Papineni et al., 2002) is computed on the generated delexicalized responses. The BLEU score measures the n-gram overlap between the generated and ground-truth responses. Note that for the computation of the BLEU score, each placeholder in the response, e.g., `[leave at]`, is considered to be a single word. Also, the BLEU score is computed on the text converted to lowercase.

5.4.2 Individual component evaluation

A classical modular dialogue system can be split into several components such as DST and policy. In our end-to-end dialogue system, we can no longer split the system into these components. We can, however, evaluate the performance of the belief state generation and the response generation that correspond to the classical DST and NLG components.

To evaluate DST, we use the joint accuracy, slot accuracy and the F1 score. The joint accuracy computes the percentage of correctly generated belief states, i.e., states where all domain-slot pairs have the correct values, and there are no missing or superfluous domain-slot pairs. The slot accuracy measures the percentage of correctly classified values for each domain-slot pairs averaged over all domain-slot pairs. Finally, the F1 score is defined as follows:

$$F_1 = \frac{\#\text{tp}}{\#\text{tp} + \frac{1}{2}(\#\text{fp} + \#\text{fn})}, \tag{5.1}$$

where #tp is the number of domain-slot-value tuples shared between generated and ground-truth belief states. #fp is the number of superfluous domain-slot-value tuples in the generated belief state and #fn is the number of these tuples in the ground-truth belief state which are missing from the dialogue system output.

For the evaluation of the NLG part of the dialogue system, we use the BLEU score (Papineni et al., 2002) and the ROUGE metric (Lin, 2004) The reference responses are pairwise compared with the generated ones, and the metrics are averaged over the whole testing dataset. We use the ROUGE-L variant of the family of ROUGE metrics, which is based on the longest common subsequence. To be able to better isolate the NLG from the rest of the system, we also evaluate the performance of the system with ground-truth belief states and ground-truth database result counts. Before the final response is generated, we substitute the generated belief state or both the belief state and database result count, with the ground-truth values.

5.5 DSTC 9 challenge

The model was submitted to the Ninth Dialog System Technology Challenge – DSTC 9 (Gunasekara et al., 2020). More specifically, the DSTC 9 challenge consists of several tracks, and one of them evaluates the performance of end-to-end task-oriented dialogue systems. It uses both automatic and human evaluation. The participants were allowed to submit up to five models, and one was selected based on an automatic evaluation using the ConvLab 2 user simulator. The best model per team was then evaluated by humans on the Amazon Mechanical Turk platform. The judges were given a set of goals, and they conducted a series of dialogues in natural language with the system in order to fulfil the goals. Afterwards, they rated the system based on whether the dialogue was successful using a binary decision. They also reported the language understanding score and response appropriateness on a five-point Likert scale. Information provided by the system was additionally checked for consistency with the database, and the average of success rates given by the judges and by database grounding was used as the main metric. Furher details are provided by Gunasekara et al. (2020).

5.6 Ablation study

In order to evaluate which contribution is the most important for the system performance, we conduct a series of experiments, separately disabling extentions

over baseline (Peng et al., 2020). Specifically, the following contributions were evaluated:

1. the unlikelihood loss described in Section 4.5
2. additional auxiliary tasks – the user intent prediction and system action prediction (see Section 4.7)
3. the data augmentation via paraphrasing (see Section 4.9)
4. the modified consistency task described in Section 4.7
5. unclean data filtering (see Section 5.1.1)

To evaluate the system, two sets of metrics were used. Simulated user evaluation (Section 5.3) and automated corpus evaluation (Section 5.4). Both evaluations use the MultiWOZ 2.1 dataset, and while each has its own advantages and disadvantages (as discussed in Section 5.3 and Section 5.4), we consider the simulated user evaluation more important because it evaluates the capabilities of the system to conduct the whole dialogue, and it allows the system to use phrasing different from the MultiWOZ dataset to achieve the same goals. Perhaps the main reason is that the DSTC 9 challenge, which is the primary focus of our dialogue system, uses the ConvLab 2 simulated user evaluation.

5.7 Human analysis

type	source	description
hallucinated values	DST/policy	system used a slot value in the reply that is not grounded in the DB nor in the context
wrong lexicalization	policy	system repeats the same value in a list of choices during lexicalization
missing information	policy	system makes booking while not all information is specified
ignored input	DST	system keeps asking for information that was provided
bad domain	DST	system fails to focus on the correct domain
false response	policy	system states a different value of a slot than the value stored in DB
repeated output	policy	system repeats the same slot twice on the output
failed booking	DB/policy	booking was unsuccessful due to DB mismatch
other	–	other rare errors

Table 5.1: The categories of errors used by expert annotators to evaluate the performance of the dialogue system. The *source* column denotes the likely source of the failure.

To better understand the behaviour of the dialogue system, we performed a detailed error analysis based on human interactions with the system. The purpose of the analysis was to categorize and quantize the kinds of errors that occur during the interaction with a real user. Random goals were sampled from the MultiWOZ test set, and expert annotators[5] were tasked to chat with the system in order to fulfil the goals. They also evaluated the quality of the system and classified the

[5]The author and three colleagues from the department.

errors into several categories, which are given in Table 5.1. The annotators were familiar with the architecture of the system and, therefore, were able to identify in detail which part of the system caused the error based on the internal system's state.

Note that the system is end-to-end and cannot be split into different components such as NLU or NLG. However, since the model uses two-stage decoding, we can separate the errors caused by the belief state decoding (denoted as *DST* in Table 5.1) from the errors caused by the response decoding (denoted as *policy* errors in the table). Also, some errors of the dialogue system are caused by a mismatch between the database and the dataset (denoted as *DB* errors in the table). For example, some dialogues contain typos in names of hotels in the belief state annotation. The database implementation cannot find the same entity as the crowdsourced worker did during the collection of the dataset.

5.8 Generating lexicalized responses

We have identified rule-based lexicalization as one of the main cause of errors (see Section 6.7). Therefore, in this section, we explore an alternative approach that lets the language model generate the final response. For more details, please refer to Section 4.4.2. To be able to generate the responses, the model has access to full database results instead of using only the counts. It is expected to copy the slot values from its input to the decoded responses. The ConvLab 2 evaluation is used to compare different variants because MultiWOZ evaluation evaluates only the delexicalized responses and cannot be applied to systems that return fully lexicalized responses.

The original method corresponds to delexicalizing all slots. An alternative is to delexicalize only the `[reference]` token. This token should always be delexicalized because in the database, this token is generated on the fly, and the database results would not be consistent with responses.

We also try to delexicalize only some slots while leaving the rest. More specifically, the following slots were chosen: `[phone]`, `[address]`, `[postcode]`, `[reference]`, `[id]`, because they are present in the dialogue goals in the MultiWOZ dataset (Budzianowski et al., 2018). Therefore, this variant of the model could theoretically be evaluated using the MultiWOZ evaluation.[6]

[6]In fact, the MultiWOZ evaluation also uses the `[name]` token, which is not a part of dialogue goals. The same code could be rewritten to use string matching to verify if the entity name is present in the string.

6. Results

In this chapter, we present the results of experiments described in Chapter 5. We compare the full AuGPT model (see Section 4.2) with state-of-the-art approaches in Section 6.1 using the ConvLab 2 simulated user evaluation (Zhu et al., 2020) and in Section 6.2 using the MultiWOZ evaluation (Budzianowski et al., 2018) – the details of the experiments were given in Section 5.3 and Section 5.4, respectively. To evaluate which of the proposed contributions described in Chapter 4 had the most significant impact on the overall performance, in Section 6.3, we present the results of the ablation study described in Section 5.6. In Section 6.4, we evaluate different parts of our system separately. In particular, we are interested in the quality of the belief state and response generation. While for the belief state prediction, we use the joint accuracy metric (see Section 5.4.2), for the response, we use the BLEU score (Papineni et al., 2002) and the ROUGE metric (Lin, 2004). In Section 4.4.2, we have proposed an alternative approach to the error-prone delexicalization, and in Section 6.5 we report the results of its analysis. All presented results are discussed in Chapter 7.

A variant of the AuGPT dialogue system competed in the DSTC 9 challenge (Gunasekara et al., 2020), and it placed third out of ten. For the competition, we have submitted five best-performing variants of our model according to the ablation study (Section 6.3). Based on the ConvLab 2 simulated user evaluation, one of the submitted models was chosen for a human evaluation. In Section 5.5 we give more detailed results of the competition.

To gain further insight into the quality of the dialogue system, we have conducted in-house system analysis (see Section 5.7), where expert annotators (the author of this thesis and three colleagues from the department) communicated with the system and evaluated its performance. The results of the analysis are presented in Section 6.7.1. We also demonstrate the most frequently occurring errors on authentic dialogues in Section 6.7.2. In Section 6.7.3, we display four conversations with the same randomly chosen goal using four selected variants of our dialogue system.

The author of the thesis trained all models together with one of his colleagues. In total, we trained 154 models with different hyperparameters and the author of the thesis trained 127 models (82%).

6.1 ConvLab 2 evaluation

Table 6.1 shows a comparison of two versions of our system, *AuGPT* and *AuGPT-b*, with two baselines in the ConvLab evaluation scheme with a simulated user. While AuGPT was trained only on *clean samples* and uses the unlikelihood loss for the response (see Chapter 4), *AuGPT-b* is a simpler version without these contributions. Both models were pre-trained on the Schema-Guided Dialogue and Taskmaster-1 datasets. For further details, please refer to Section 5.6. These variants were chosen for comparison based on the ablation study presented in Section 6.3, where *AuGPT* and *AuGPT-b* achieved the best performance in the corpus-based and simulated user evaluations, respectively. The compared systems – DAMD (Zhang et al., 2020b) and MD-Sequicity (Lei et al., 2018) – were chosen

because they are both fully trainable end-to-end methods (see Chapter 3) and their results for the ConvLab-2 evaluation were available. We also report the results of the best hand-crafted rule-based dialogue system engineered by Zhu et al. (2020).

				inform			turn	
method	complete	success	book	P	R	F1	succ	all
hand-crafted	90.5	81.3	91.1	79.7	92.6	83.5	11.6	12.3
AuGPT	89.4	60.1	85.7	64.5	82.1	70.3	12.7	14.6
AuGPT-b	85.9	58.4	81.3	62.2	79.8	67.5	12.6	14.1
DAMD	39.5	34.3	51.4	60.4	59.8	56.3	15.8	29.8
MD-Sequicity	23.1	9.8	4.1	33.0	32.7	29.9	12.2	32.6

Table 6.1: The table shows the ConvLab-2 evaluation with an user simulator. For the description of the metrics, please refer to the Section 5.3.

We can see that our dialogue system outperformed both compared methods by a wide margin in all metrics. The performance even approached the performance of the best handcrafted rule-based dialogue system, which was engineered specifically for the ConvLab 2 evaluation.

6.2 MultiWOZ results

In this section, we present the results of the corpus evaluation on MultiWOZ 2.0 and MultiWOZ 2.1 using the original metrics suggested by Budzianowski et al. (2018) (see Section 5.4). We compare two versions of our system, *AuGPT* and *AuGPT-b*, with state-of-the-art systems. Details on the compared systems are given in Chapter 3.

The results can be seen in Table 6.2. Unfortunatelly, some models did not provide results for the newer version of the dataset, and therefore, to be able to compare with other approaches, the system was also evaluated on the MultiWOZ 2.0 dataset. The following systems were compared: SOLOIST (Peng et al., 2020), SimpleTOD (Hosseini-Asl et al., 2020), LABES-S2S (Zhang et al., 2020a), DAMD (Zhang et al., 2020b), MD-Sequicity (Zhang et al., 2020b), LAVA (Lubis et al., 2020). MD-Sequicity is a variant of Lei et al. (2018)'s model, extended for a multi-domain setting.

From the results, we can see that on the MultiWOZ 2.1 dataset, the AuGPT-b variant of the model outperforms the AuGPT system by a relatively wide margin in terms of inform and success rates. The BLEU score for both variants is, however, comparable. While AuGPT-b outperforms the SimpleTOD system slightly in terms of the inform rate and has a better BLEU score, it has a lower success rate. Also, both variants of AuGPT outperform LABES-S2S in terms of inform and success rates but have lower BLEU.

On the MultiWOZ 2.0 dataset, both AuGPT variants had comparable performance, with full AuGPT outperforming the AuGPT-b variant slightly. From all compared systems, ours reached the highest BLEU score. On the other hand, in terms of inform and success rates, many compared systems outperformed AuGPT.

method	MultiWOZ 2.0 inform	success	BLEU	MultiWOZ 2.1 inform	success	BLEU
Human	91.0	82.7	–	86.3	79.1	–
AuGPT	83.1	70.1	17.2	83.5	67.3	17.2
AuGPT-b	82.3	68.3	17.3	86.5	69.1	17.5
SOLOIST	85.5	72.9	16.5	–	–	–
SimpleTOD	84.4	70.1	15.1	85.0	70.5	15.2
LABES-S2S	–	–	–	78.1	67.1	18.3
DAMD	76.3	60.4	16.6	–	–	–
MD-Sequicity	86.6	71.6	16.8	–	–	–
LAVA	91.8	81.8	12.0	–	–	–

Table 6.2: Comparison with previous works on the MultiWOZ 2.0 and 2.1 datasets. Please refer to Section 5.4 for a description of the metrics.

method	inform	success	BLEU
AuGPT	83.5	67.3	17.2
AuGPT-b	**86.5**	**69.1**	**17.5**
w/o. unlikelihood	84.1	66.9	17.1
w/o. clean	81.9	64.0	15.8
w. all auxiliary	83.1	66.2	17.0
w/o. pre-training	81.0	62.7	15.1
w/o. back-translations	79.8	61.7	15.2
w. old consistency	81.4	65.8	17.0
w/o. consistency	81.9	64.5	16.3

Table 6.3: Ablation study results on the MultiWOZ 2.1 dataset. See Section 5.4 for a description of the metrics.

We discuss the results and also give some insight into why our model did not outperform other approaches in Chapter 7.

6.3 Ablation study results

Many variants of the proposed dialogue system were compared in order to evaluate the importance of each of the contributions. In Table 6.4, we present the results of the ablation study when ConvLab 2 user simulator was used for the evaluation. Table 6.3 then displays the results of the MultiWOZ 2.1 evaluation of the same variants of our model. The following variants of the model were evaluated:

1. *AuGPT* – the full model
2. *AuGPT-b* – a variant of the model without unlikelihood loss trained on the full dataset
3. *w/o. unlikelihood* – without the unlikelihood loss
4. *w/o. clean* – a variant of the model trained on the full dataset (no filtering was used)
5. *w. all auxiliary* – a variant with additional auxiliary tasks

method	comp	suc	book	inform P	inform R	inform F1	turn suc	turn all
AuGPT*	**89.4**	**60.1**	85.7	64.5	**82.1**	70.3	12.7	14.6
AuGPT-b*	85.9	58.4	81.3	62.2	79.8	67.5	12.6	**14.1**
w/o. unlikelihood*	89.2	59.3	**90.8**	63.9	81.6	69.5	12.8	14.6
w/o. clean	85.0	57.7	85.6	65.6	79.1	69.6	12.7	14.5
w. all auxiliary*	88.7	59.2	86.0	64.6	81.1	69.9	**12.6**	14.4
w/o. pre-training*†	88.1	59.8	83.7	**68.1**	80.9	72.1	13.5	15.6
w/o. back-translations	88.9	58.2	87.4	68.0	81.6	**72.2**	12.9	14.9
w. old consistency	85.5	57.8	86.0	65.2	80.0	69.8	12.7	14.6
w/o. consistency	86.4	57.1	84.1	66.3	81.2	70.9	13.1	14.6

Table 6.4: Ablation study on the ConvLab 2 user simulator platform. In the table, *comp*, *suc*, and *book* denote the complete, success, and book rates, respectively. See Section 5.3 for a description of the metrics. The variants with '*' denote the five selected variants for the DSTC 9 competition and '†' denotes the variant selected for human evaluation in DSTC 9 (see Section 5.5 and Section 6.6).

6. *w/o. pre-training* – a variant of the model where no pre-training on larger datasets was used
7. *w/o. back-translations* – a variant trained without the data augmentation via paraphrasing
8. *w. old consistency* – a variant with the same consistency detection task as described in SOLOIST method (Peng et al., 2020)
9. *w/o. consistency* – a variant without the consistency detection auxiliary task

We can see that all proposed contributions, which are a part of our final AuGPT, have a positive effect on the system performance with respect to the primary metrics on the ConvLab 2 dataset. The smallest decrease in performance occurred when the unlikelihood loss was not used for the response, suggesting that this contribution has the lowest importance of all. If we look at the results of the *w. old consistency* and the *w/o. consistency* variants, we can see that the original SOLOIST's consistency detection auxiliary task is better than no consistency detection. However, our modified consistency detection outperforms the SOLOIST's by a wide margin. One of the biggest decreases in performance was caused when the model was trained on the full dataset instead of using only the clean samples. Furthermore, the added user intent and system action prediction auxiliary tasks did not improve the performance further, as can be seen from the results of the *w. all auxiliary* variant. An important thing to notice is that the *AuGPT-b* variant of the model performed rather poorly in this evaluation.

Unfortunately, the results of the MultiWOZ 2.1 evaluation (Table 6.3) are not consistent with the ConvLab 2 evaluation. While in the ConvLab 2 evaluation, the best performing model was the full system with all the contributions enabled (except for additional auxiliary tasks), in the case of the MultiWOZ evaluation, the best-performing model is the *AuGPT-b* variant without the unlikelihood loss and trained on the full dataset. Some insight into why this was the case

will be given in Section 7.2. In any case, we can see that removing either the pre-training or the back-translations decreases the BLEU score substantially and, more importantly, the success rates. Furthermore, we notice the positive effect of using our improved consistency detection task over the one used in SOLOIST (Peng et al., 2020), which in turn scores better than no consistency detection in terms of both the BLEU score and the success rate. Using the user intent and system action prediction auxiliary tasks (see Section 4.7) made the system performance slightly worse. This is consistent with the ConvLab 2 evaluation, where using all auxiliary tasks did not help either. Finally, by removing either the unlikelihood loss or by training the model on all training data (not just clean samples), the performance drops in terms of both the BLEU score and the success rate. Out of these two contributions, using the clean samples only has more impact on the performance.

To conclude, using back-translation improves the performance substantially regardless of the evaluation method. Similarly, pre-training the model on larger dialogue datasets and using the proposed consistency detection auxiliary task increases performance in both evaluations. Unfortunately, adding the *user intent* and *system action* auxiliary tasks does not increase the quality of the trained model. Training only on *clean samples* seems to increase the performance in both evaluations; however, without the unlikelihood loss, we get mixed results in the MultiWOZ evaluation. Using the unlikelihood loss has possibly the lowest impact of all proposed contributions.

We have used the results from the ablation study to identify the best dialogue system configurations. The two selected variants, *AuGPT* and *AuGTP-b* were chosen because they have the best performance in the ConvLab 2 and MultiWOZ evaluations, respectively. Therefore, when comparing with other dialogue systems, we use these two configurations. In Section 6.7.3, we give examples of conversations generated by these variants.

6.4 Individual component analysis

To gain further insight into the performance of different parts of the system, we evaluate the belief state generation and the response generation individually. They correspond to the DST and NLG components in classical dialogue systems (see Section 2.4). The results are presented in Table 6.5. We train AuGPT on both MultiWOZ 2.0 and MultiWOZ 2.1 datasets. In the DST evaluation, the joint and slot accuracies are reported together with the F1 metric (see Section 5.4.2). For NLG, we try to isolate the DST component by using ground-truth belief states or ground-truth database result counts. These variants are called *oracle bs* and *oracle db* in the table. We report the BLEU score and the ROUGE-L metric.

From the results, we can see that for both datasets, both the BLEU score and ROUGE-L metric increase only slightly when we use ground-truth values for the belief state and the database result counts. It should be noted, however, that delexicalized responses are compared, and the number of cases where the belief state or database result counts were bad enough to cause an NLG error is not that big. This error could occur, for example, when the actual number of results per domain is zero but the NLG uses a value greater than 0. In that case, a response such as 'There are 5 hotels' is generated instead of 'I was not able to find a hotel'.

fine-tuned on	oracle bs	oracle db	DST joint acc.	DST slot acc.	DST F1	NLG BLEU	NLG ROUGE-L
MW 2.0	✗	✗	54.1	97.2	90.0	17.2	39.0
	✗	✓				17.4	39.3
	✓	✓				17.4	39.2
MW 2.1	✗	✗	56.5	97.2	90.6	17.4	38.6
	✗	✓				17.6	38.8
	✓	✓				17.6	38.8

Table 6.5: Performance of DST and NLG components. Joint and slot accuracies, as well as slot values F1 score, are used to evaluate DST. For NLG, BLEU and ROUGE-L metrics are used. Apart from using the generated belief states and database counts, we also evaluate the components with oracle values.

To conclude, the NLG component is relatively insensitive to errors in DST.

6.5 Generating lexicalized responses

In this section, we experiment with an alternative approach to the lexicalization problem. We compare two variants – *no-delex* and *basic-delex* – both of which do not use the fully delexicalized responses, but let the model copy the values from the database results passed as a string to the model. The *no-delex* variant uses the lexicalization only for the `[reference]` placeholder. The *basic-delex* variant lexicalizes the following placeholders (see Section 5.8 for details):
`[phone]`, `[address]`, `[postcode]`, `[reference]`, `[id]`.
The ConvLab 2 user simulator evaluation was used for comparing the two methods and the results can be seen in Table 6.6. The set of metrics is the same as in Section 6.1.

method	comp	suc	book	inform P	inform R	inform F1	turn succ	turn all
hand-crafted	90.5	81.3	91.1	79.7	92.6	83.5	11.6	12.3
AuGPT	89.4	60.1	85.7	64.5	82.1	70.3	12.7	14.6
no-delex clean	89.8	75.0	82.9	**72.9**	**91.6**	**78.5**	13.5	14.3
no-delex full	89.5	**76.8**	85.7	72.4	91.4	78.3	13.3	14.3
basic-delex clean	**90.5**	57.4	85.1	63.9	80.9	69.4	12.7	14.7
basic-delex full	89.1	57.6	**88.7**	64.2	80.6	69.1	12.8	14.9

Table 6.6: The table shows the comparison between different implicit lexicalization dialogue system variants using the ConvLab-2 evaluation. The *full* postfix denotes that the model was trained on the full dataset, where as *clean* denotes that only clean samples were used. The AuGPT model, which uses full delexicalization, is included for comparison. For comparison, we also include a hand-crafted dialogue system engineered specifically for MultiWOZ by Zhu et al. (2020). In the table, *suc* and *comp* stand for the success and complete rate respectively. For the description of the metrics, please refer to the Section 6.1.

method	average success	success w/ DB	success w/o DB	NLU score	response appropriateness	turns
baseline	69.6	56.8	82.4	4.34	4.18	18.5
winner	**74.8**	**70.2**	79.4	**4.54**	**4.47**	18.5
our submission	72.3	62.0	**82.6**	4.53	4.41	**17.1**

Table 6.7: The results of the DSTC 9 challenge obtained using Amazon Mechanical Turk.

From the results, we can see that the variant of the model which apart from the *reference* slot did not use any delexicalization (*no-delex*) strongly outperformed both the *basic-delex* variant and the fully delexicalized model in terms of the success rate. Furthermore, this variant had also the best precision, recall, and F1 score. When the *no-delex* variant was trained on the full dataset, it had a slightly better complete rate and substantially better success rate and book rate while having comparable precision, recall and F1 score. We can see a similar trend in the *basic-delex* variant, where the variant trained on the full dataset has a slightly better success rate and much better book rate. In the *basic-delex* case, however, the complete rate is better for the variant trained on *clean samples* only. Overall, the best variant of our dialogue system – *no-delex full* – substantially outperformed the full AuGPT system and even had comparable performance to the hand-crafted dialogue system engineered specifically for MultiWOZ by Zhu et al. (2020).

6.6 DSTC 9 challenge results

We participated in the DSTC 9 end-to-end dialogue state tracking challenge (see Section 5.5). Each competing team was allowed to submit five models for the evaluation, and therefore, we submitted the five best-performing models from Table 6.4. The selected models are denoted by an asterisk in the table. The first round of the competition used the ConvLab 2 user simulator evaluation to select the best performing model for the second round. Unexpectedly, for the second round, the variant of the model without pre-training (*w/o. pre-training* in Table 6.4) was selected instead of the full AuGPT system. In the second round, the system was evaluated on the Amazon Mechanical Turk platform by crowdsourced workers who communicated with the system in natural language in order to fulfil randomly sampled goals. At the end of the dialogue, the workers judged whether the dialogue was successful with a binary decision and they provided scores based on language understanding correctness and response appropriateness on a 5-point Likert scale. Since the workers did not have direct access to the database, the success rate with database grounding was also reported after verifying whether the requested slot values returned by the dialogue system matched the target database record specified in the dialogue goal. The average of these two success rates was computed and used for the final ranking. Our system placed third out of 10 in the competition. The comparison with the winner and a baseline can be seen in Table 6.7. The baseline was relatively strong – only 4 competitors scored higher than the baseline.

From the results, we can see that the NLU score of our approach closely matched the score of the winner. The response appropriateness scores were also similar. While the winner had the best success rate with grounding, our system had the best success rate without grounding. For a discussion on why our model had problems with the database grounding, please refer to Section 7.4. Also, our system outperformed the winner in terms of the average turns needed to finish the dialogue, suggesting more efficient conversations.

6.7 Human evaluation results

6.7.1 In-house system analysis

In addition to the results obtained from the DSTC 9 challenge, we performed an in-house error analysis of the full variant of our AuGPT dialogue system. Our expert annotators evaluated 130 dialogues in total. Out of these dialogues, 80 did not contain any errors, and 50 did. Even if the dialogue had an error, the system was sometimes able to recover, resulting in the success rate of 86.9%, i.e., 17 failed dialogues. We want to stress out that overall the system performed surprisingly well, and the errors were mostly minor. Please refer to Section 6.7.3 for examples of generated dialogues.

The annotators were tasked with classifying the errors in the dialogue into several categories based on the probable source of the error. In Table 6.9, the numbers of occurences of different error types are given. We also tried to pair each error type with its likely source. Although we classify errors by the 'component' which had caused them, the system is end-to-end and there are no traditional components such as NLG, DST. We can still identify parts of the system which correspond to traditional dialogue system components (see Section 5.7). The number of errors per dialogue system component responsible for causing the error is given in Table 6.8.

	erroneous dialogues	BS error	DB error	policy error	other error
all dialogues	50	30	4	21	6
unsuccessful dialogues	17	10	3	2	2

Table 6.8: Interactive analysis performed by human evaluators using 130 dialogue goals from the MultiWOZ dataset (Budzianowski et al., 2018). 17 of these dialogues contained an error that caused the dialogue to fail. We show summary statistics regarding the number of respective error sources (BS = belief state, DB = database). Note that some of the dialogues contain more than one error.

From Table 6.8 we can see that most errors were caused by an incorrect belief state. In particular, 30 dialogues contained an error in the belief state and in ten cases, the error caused the dialogue to fail. The policy was also a common source of errors; however, these errors rarely caused the dialogue to fail. An interesting observation is that the system was able to recover from an error in almost two thirds of the cases.

type	count	source
hallucinated values	21	BS/policy
wrong lexicalization	6	policy
missing information	5	policy
ignored input	5	BS
bad domain	4	BS
false response	4	policy
repeated output	3	policy
failed booking	3	DB/policy
other	10	–

Table 6.9: Distribution of the most common error types encountered during the human evaluation of 130 dialogues. Absolute counts of errors in the 50 erroneous dialogues are shown. The total error count is 61 as some dialogues contained more than one error. The most likely source of the error is given for each error type. For further details please refer to Section 5.7.

In Table 6.9, we can see that the most frequent error type was value hallucination. An example of this error type can be seen in Section 6.7.2. This error occurred more than three times more often than the second most common error type – wrong lexicalization. Another frequent error type was 'missing information', which occurred when the system tried to make the booking without having all the information necessary. Ignoring an user input was another relatively common error.

6.7.2 Erroneous dialogue examples

...
user: *I want to make a booking at sitar tandoori*
belief: restaurant {name = sitar tandoori}
train {arrive by = 18:00, day = sunday, departure = cambridge, destination = london liverpool street, people = 1}
system: What day would you like the booking for?
user: *the same day, 13:30*
belief: restaurant {day = sunday, name = sitar tandoori, people = 1, time = 13:30}
train {arrive by = 18:00, day = sunday, departure = cambridge, destination = london liverpool street, people = 1}
system: Booking was successful. The table will be reserved for 15 minutes. Reference number is: 00000200.

Figure 6.1: A dialogue where the system did not ask for necessary information. The user previously booked a train trip for one person (cf. the belief state in the first turn shown). When they went on to book a restaurant, the system took over the number of persons from the train ride and did not ask again.

In this section, some examples of the most frequent or interesting errors are provided to the reader. The most frequent error – *hallucinated values* – which is a

user: *I'm looking for a theatre in Cambridge*
belief: attraction {type = theatre}
system: There are 5 theatres in Cambridge. Is there an area of town you prefer?
user: *no I don't care*
belief: attraction {type = theatre}
system: I recommend Adc Theatre in the centre. Would you like more information?
user: *no, I'm also looking for some place to stay*
belief: attraction {type = theatre}
system: I have 5 theatres in the centre. Do you have a price range?
user: *I'm looking for a place to stay*
belief: attraction {type = theatre}
system: There are 5 theatres in the centre. Do you have a preference for area?
user: *I'm looking for a hotel*
belief: hotel {type = hotel} attraction {type = theatre}
system: *There are 9 hotels in the north. Do you have a price range?*

Figure 6.2: Dialogue sample with a bad domain focus and a hallucination. First, the system ignores that the user switched from searching for a theater to searching for a hotel. After accepting the new domain, the system replies with hotels "in the north" even though the user did not specify.

lack of grounding for some of the information provided by the system, can be seen at the end of Figure 6.2. The system informs the user that 'there are 9 restaurants in the north', while the user never specified they wanted the restaurant to be in the north. This type of error often came from the response decoding, as was the case of this example. The error is likely caused by errors in the delexicalization, which was far from perfect in MultiWOZ 2.0 and MultiWOZ 2.1.

Another error type – *missing information*, i.e., not asking for information that is required can be seen in Figure 6.1. In this dialogue, the system used information from a different domain without an explicit confirmation from the user. In the training dataset, this domain transfer sometimes occurs, and, therefore, the model learned to act in the same way.

Finally, another common error type is *bad domain*. This error can be seen in the middle of Figure 6.1. In this case, the system has difficulty switching to a new domain, *hotel*. Instead, it keeps recommending attractions, and the user has to repeat the request. The likely cause of the problem is the inability of the model to generalize beyond the training data and to adapt to a less frequent wording. These types of error usually get resolved by paraphrasing the same input.

6.7.3 Case study

In this section, we compare four selected variants of the proposed system by having the same conversation with it. More specifically, a random dialogue goal was sampled from the MultiWOZ test set, and an expert user[1] tried to achieve this same goal while using similar responses whenever possible. The shown variants are:

1. AuGPT (Figure 6.3) – full dialogue system

[1]The author of this thesis.

2. AuGPT-b (Figure 6.4) – AuGPT without unlikelihood loss trained on the full dataset
3. no-delex (Figure 6.5) – a variant of AuGPT where only the `reference` slot is delexicalized
4. basic-delex (Figure 6.6) – a variant of AuGPT where the delexicalization is applied to a subset of all slots described in Section 4.4.2

The shared dialogue goal consisted of the following instructions:

- You are planning your trip in Cambridge
- You are looking for a train. The train should depart from cambridge and should go to london kings cross
- The train should leave on saturday and should leave after 18:45
- Once you find the train you want to make a booking for 8 people
- Make sure you get the reference number
- You are also looking for a place to dine. The restaurant should be in the expensive price range and should be in the centre
- Once you find the restaurant you want to book a table for the same group of people at 14:15 on the same day
- If the booking fails how about 13:15
- Make sure you get the reference number

For all dialogues, we display the user input, system response and system's internal belief state. In some cases, we also display a part of the database results formatted for the reader. The format is the same as the textual representation used in the 'no-delex' variant of the dialogue system.[2] We display the lexicalized responses. However, above each lexicalized text (in italics), we show the original slot placeholder which was replaced by the lexicalizer based on database results. Therefore, the reader can easily see both the generated and lexicalized responses. Finally, sometimes the belief state was too long, and some parts were left out. This is denoted by '. . . ' in dialogue examples.

Overall, all variants performed quite well on this dialogue goal. The conversation with the full AuGPT dialogue system can be seen in Figure 6.3. The second system reply is quite challenging since the system has to be able to correctly lexicalize the `[id]`, `[leave at]`, and `[arrive by]` placeholders, which must come from the same database entity. Also, notice how the system correctly transfers the information about the number of people from the *train* domain to the *restaurant* domain.

The second dialogue with the AuGPT-b variant is displayed in Figure 6.4. We can see the lexicalization fail in the second system reply. The `[leave at]` placeholder is repeated two times and is incorrectly lexicalized from the first and the second database entities. The first placeholder should have repeated user preference on train departure, whereas the second placeholder should have stated the departure time of the first returned database entity. Unfortunately, the lexicalizer does not have any information to help it decide which values it should use.

The dialogue generated by the *no-delex* variant, which lexicalizes only the `[reference]` token can be seen in Figure 6.5. We can see that in several places

[2]For all variants, even if a different format is used intenally

(denoted in green), the system correctly copies the correct text from the database results. Unfortunately, at the end of the dialogue (5^{th} turn), we can also see the system hallucinating a slot value that it cannot know from the database results only. More specifically, the system recommends a restaurant called 'bedouin' which really exists but was not one of the two names returned in the database results string.

The last example (Figure 6.6) was generated by the *basic-delex* variant of the system, which lexicalizes only a small subset of the database columns. Same as in the previous example the system hallucinates a name of an existing restaurant and even its type of food.

user (1): I am looking for a train from Cambridge to London Kinks Cross.

belief (1): train {departure = cambridge, destination = london kings cross}

system (1): There are 70 trains departing from Cambridge to London Kings Cross. What day would you like to travel?

user (2): I would like to leave on Saturday after 18:45.

belief (2): train {day = saturday, departure = cambridge, destination = london kings cross, leave at = 18:45}

db (2): train (3) {arrive by = 19:51, 21:51, 23:51; id = TR0427, TR0925, TR4898; leave at = 19:00, 21:00, 23:00; price = 18.88 pounds; reference = 00000057}

system (2): *TR0427* [id] leaves at *19:00* [leave at] and arrives by *19:51* [arrive by]. Would you like to book this train?

user (3): Yes, I would like to book it for eight people.

belief (3): train {day = saturday, departure = cambridge, destination = london kings cross, leave at = 18:45, people = 8}

system (3): I have booked it for you. Your reference number is *00000057* [reference]. Is there anything else I can help you with?

user (4): I am also looking for an expensive restaurant in the centre.

belief (4): restaurant {area = centre, price range = expensive} train {day = saturday, departure = cambridge, destination = london kings cross, leave at = 18:45, people = 8}

db (4): restaurant (33) {address = 106 Regent Street City Centre, ...; area = centre (33); food = indian (6); name = Curry Garden, Stazione Restaurant and Coffee Bar, ...; phone = 01223302330; postcode = CB21DP; price range = expensive (33); reference = 00000002; type = restaurant (33)}, train (3)

system (4): There are 33 *expensive* [price range] restaurants in the *centre* [area]. Is there a particular type of food you would like?

user (5): I have no preference.

belief (5): restaurant {area = centre, price range = expensive} train {day = saturday, departure = cambridge, destination = london kings cross, leave at = 18:45, people = 8}

system (5): I would recommend *Curry Garden* [name]. Would you like a reservation?

user (6): I would like to book a table for the same group of people at 14:15 on the same day.

belief (6): restaurant {area = centre, day = saturday, name = curry garden, people = 8, price range = expensive, time = 14:15} train {day = saturday, departure = cambridge, destination = london kings cross, leave at = 18:45, people = 8}

system (6): Booking was successful. The table will be reserved for 15 minutes. Reference number is: *00000002* [reference].

Figure 6.3: An example of a dialogue generated by the full AuGPT system. Details are given in Section 6.7.3

user (1): I am looking for a train from Cambridge to London Kinks Cross.

belief (1): train {departure = cambridge, destination = london kings cross}

system (1): There are 70 trains traveling between Cambridge and London Kings Cross. What day would you like to travel?

user (2): I would like to leave on Saturday after 18:45.

belief (2): train {day = saturday, departure = cambridge, destination = london kings cross, leave at = 18:45}

db (2): train (3) {arrive by = 19:51, 21:51, 23:51; id = TR0427, TR0925, TR4898; leave at = 19:00, 21:00, 23:00; price = 18.88 pounds; reference = 00000057}

system (2): There are 3 trains that leave after [leave at] ***19:00*** . The earliest leaves at [leave at] ***21:00*** . Would you like me to book it for you?

user (3): Yes, I would like to book it for eight people.

belief (3): train {day = saturday, departure = cambridge, destination = london kings cross, leave at = 18:45, people = 8}

system (3): Booking was successful, the total fee is [price] *151.04 pounds* payable at the station. Reference number is: [reference] *00000057*.

user (4): Ok. I am also looking for an expensive restaurant in the centre.

belief (4): restaurant {area = centre, price range = expensive} train {...}

db (4): restaurant (33) {address = 106 Regent Street City Centre, ...; area = centre (33); food = indian (6); name = Curry Garden, Stazione Restaurant and Coffee Bar, ...; phone = 01223302330; postcode = CB21DP; price range = expensive (33); reference = 00000002; type = restaurant (33)}, train (3)

system (4): There are 33 [price range] *expensive* restaurants in the [area] *centre*. Is there a particular type of food you would like?

user (5): I have no preference.

belief (5): restaurant {area = centre, price range = expensive} train {...}

system (5): I would recommend [name] *Curry Garden*. Would you like a reservation?

user (6): I would like to book a table for the same group of people at 14:15 on the same day.

belief (6): restaurant {area = centre, day = saturday, name = curry garden, people = 8, price range = expensive, time = 14:15} train {...}

system (6): Booking was successful. The table will be reserved for 15 minutes. Reference number is: [reference] *00000002*.

Figure 6.4: An example of a dialogue generated by the AuGPT-b dialogue system trained without the unlikelihood loss on the full training dataset. Note the error in the second system response where `[leave at]` tokens are incorrectly lexicalized. Details are given in Section 6.7.3.

user (1): I am looking for a train from Cambridge to London Kinks Cross.
belief (1): train {departure = cambridge, destination = london kings cross}
system (1): What day and time would you like to travel?
user (2): I would like to leave on Saturday after 18:45.
belief (2): train {day = saturday, departure = cambridge, destination = london kings cross, leave at = 18:45}
db (2): train (3) {arrive by = 19:51, 21:51, 23:51; id = TR0427, TR0925, TR4898; leave at = 19:00, 21:00, 23:00; price = 18.88 pounds; reference = 00000057}
system (2): **TR0427** leaves at **19:00** and arrives by **19:51**. Would you like me to book it for you?
user (3): Yes, I would like to book it for eight people.
belief (3): train {day = saturday, departure = cambridge, destination = london kings cross, leave at = 18:45, people = 8}
db (3): train (3) {arrive by = 19:51, 21:51, 23:51; id = TR0427, TR0925, TR4898; leave at = 19:00, 21:00, 23:00; price = **151.04** pounds; reference = 00000057}
system (3): Booking was successful, the total fee is **151.04** GBP payable at the station. Reference number is: *00000057*[reference].
user (4): Ok. I am also looking for an expensive restaurant in the centre.
belief (4): restaurant {area = centre, price range = expensive} train {...}
db (4): restaurant (**33**) {address = 106 Regent Street City Centre, ...; area = centre (33); food = indian (6); name = Curry Garden, Stazione Restaurant and Coffee Bar, ...; phone = 01223302330; postcode = CB21DP; price range = expensive (33); reference = 00000002; type = restaurant (33)}, train (3)
system (4): There are **33** expensive restaurants in the centre. Is there a particular type of food you would like?
user (5): I have no preference.
belief (5): restaurant {area = centre, price range = expensive} train {...}
db (5): restaurant (33) {address = 106 Regent Street City Centre, ...; area = centre (33); food = indian (6); name = Curry Garden, Stazione Restaurant and Coffee Bar, ...; phone = 01223302330; postcode = CB21DP; price range = expensive (33); reference = 00000002; type = restaurant (33)}, train (3)
system (5): I would recommend **bedouin**. Would you like a reservation?
user (6): I would like to book a table for the same group of people at 14:15 on the same day.
belief (6): restaurant {area = centre, day = saturday, name = bedouin, people = 8, price range = expensive, time = 14:15} train {...}
system (6): Booking was successful. The table will be reserved for 15 minutes. Reference number is: *00000059*[reference].

Figure 6.5: An example of a dialogue generated by the *no-delex* variant of the AuGPT dialogue system where only the `reference` slot is lexicalized. The text in red color denotes hallucinated values and green denotes correctly copied values. Details are given in Section 6.7.3.

user (1): I am looking for a train from Cambridge to London Kinks Cross.
belief (1): train {departure = cambridge, destination = london kings cross}
system (1): What day and time will you be departing?
user (2): I would like to leave on Saturday after 18:45.
belief (2): train {day = saturday, departure = cambridge, destination = london kings cross, leave at = 18:45}
db (2): train (3) {arrive by = 19:51, 21:51, 23:51; id = TR0427, TR0925, TR4898; leave at = 19:00, 21:00, 23:00; price = 18.88 pounds; reference = 00000057}
system (2): The earliest train after that time is *TR0427*[id] which leaves Cambridge at **19:00** and arrives in London Kings Cross at **19:51**. Would you like to book a ticket?
user (3): Yes, I would like to book it for eight people.
belief (3): train {day = saturday, departure = cambridge, destination = london kings cross, leave at = 18:45, people = 8}
db (3): train (3) {arrive by = 19:51, 21:51, 23:51; id = TR0427, TR0925, TR4898; leave at = 19:00, 21:00, 23:00; price = 151.04 pounds; reference = 00000057}
system (3): Booking was successful, the total fee is **151.04** GBP payable at the station. Reference number is: *00000057*[reference].
user (4): Ok. I am also looking for an expensive restaurant in the centre.
belief (4): restaurant {area = centre, price range = expensive} train {...}
db (4): restaurant (33) {address = 106 Regent Street City Centre, ...; area = centre (33); food = indian (6); name = Curry Garden, Stazione Restaurant and Coffee Bar, ...; phone = 01223302330; postcode = CB21DP; price range = expensive (33); reference = 00000002; type = restaurant (33)}, train (3)
system (4): There are **33** expensive restaurants in the centre. Is there a particular type of food you would like?
user (5): I have no preference.
belief (5): restaurant {area = centre, price range = expensive} train {...}
db (5): restaurant (33) {address = 106 Regent Street City Centre, ...; area = centre (33); food = indian (6); name = Curry Garden, Stazione Restaurant and Coffee Bar, ...; phone = 01223302330; postcode = CB21DP; price range = expensive (33); reference = 00000002; type = restaurant (33)}, train (3)
system (5): I would recommend **bedouin**, a **british cuisine**. Would you like to make a reservation?
user (6): I would like to book a table for the same group of people at 14:15 on the same day.
belief (6): restaurant {area = centre, day = saturday, name = bedouin, people = 8, price range = expensive, time = 14:15} train {...}
system (6): Your table is booked, ref # *00000059*[reference]. Is there anything else I can help you with today?

Figure 6.6: An example of a dialogue generated by the *basic-lex* variant of the AuGPT dialogue system where some slots are lexicalized. The text in red color denotes hallucinated values and green denotes correctly copied values. Details are given in Section 6.7.3.

7. Discussion

In this chapter, we discuss the results of experiments from Chapter 6 and more general aspects of the AuGPT dialogue system. Section 7.1 starts with an analysis of the results of the MultiWOZ evaluation presented in Section 6.2. We discuss one peculiar aspect of the evaluation which is that it requires the model to return the name of the booked entity even if it is not required by the dialogue goal. We show how this aspect of the evaluation influences our results and discuss possible causes of the problem. We also explain the reason why SOLOIST (Peng et al., 2020) performed better than AuGPT when in Section 6.3 we expected AuGPT to have a superior performance. In Section 7.2, we consider the results of the ablation study presented in Section 6.3. The back-translation (see Section 4.9) and consistency detection auxiliary task (see Section 4.7.1) are discussed in more detail. Section 7.3 analyzes the results presented in Section 6.5, where delexicalization was restricted to a subset of all slots (see Section 4.4.2). The caveats of using delexicalization are discussed as well as problems with models that decode the lexicalized response directly. We suggest possible future improvements which may remedy some of the problems. Finally, in Section 7.4 we examine the results of the DSTC 9 competition from Section 6.6 and human evaluation from Section 6.7 and draw overall conclusions from the results.

7.1 Automatic evaluation

In Section 6.2 we saw a comparison between the AuGPT and AuGPT-b variant of the system on the MultiWOZ evaluation. The AuGPT-b variant outperformed the full system on the MultiWOZ 2.1 dataset, while the full system was better on MultiWOZ 2.0. For training of the dialogue system, it is important that the annotations are precise. In Section 6.3, we saw that while adding noise to input utterances increase the robustness of the system, adding noise to the annotations has an opposite effect (see Section 7.2). In fact, the noise in the data annotations could have caused the discrepancy in the dialogue system performance when trained on the two different datasets. MultiWOZ 2.1 is considered a cleaner version of the dataset with a better delexicalization. Therefore, when the system was trained only on clean samples, it had a better relative performance on the noisier dataset.

Another interesting aspect of the MultiWOZ evaluation that we noticed was that in order for the entity to be matched to have high inform rate, the system had to explicitly output the `[name]` placeholder, i.e., mention the name of a venue. This requirement was enforced even when the `[name]` was not a part of the dialogue goal. Unfortunately, we noticed that by using the unlikelihood loss, we decrease the probability of decoding the `[name]` placeholder and, therefore, decreasing the performance of the dialogue system in terms of the MultiWOZ's inform and success rates. If the evaluation did not require the `[name]` placeholder to be generated, AuGPT would be better than AuGPT-b in all evaluations and would achieve the inform and success rates of 91.4% and 72.9% respectively. Compare it to current values of 83.5% and 69.1%.

On the MultiWOZ 2.0 dataset, AuGPT dialogue system was outperformed

in terms of success rate and inform rate by many other systems. An interesting inconsistency can be seen in the SOLOIST's results. In Section 6.3, we compare the SOLOIST variant of the consistency with our dialogue system and can deduce, that our system should be better. The likely cause of the problem is that our reimplementation of the SOLOIST paper achieves worse performance than the numbers reported by Peng et al. (2020). In our experiments, we have noticed that these dialogue systems are sensitive to the implementation details in the data preprocessing pipeline, e.g. in some publicly available implementations the belief state annotations are manually fixed in the source code. It is likely that we did not optimize the data preprocessing for the MultiWOZ 2.0 dataset enough and, therefore, made our performance on the dataset worse.

7.2 Importance of individual contributions

In Section 6.3, we saw that in the ConvLab 2 evaluation, the full AuGPT model performed the best. However, in the MultiWOZ evaluation, the AuGPT model that did not use the unlikelihood loss for the response and was trained on the full training dataset performed better. One possible cause for this discrepancy, as was already mentioned in Section 7.1, was that the MultiWOZ evaluation required the model to generate the `[name]` placeholder. The dialogues, in which the `[name]` was required to be generated, but was not part of the goal annotation would have zero inform rate. Unfortunately, by using only the clean samples for training, we completely ignore this data during training. Therefore, training only on the *clean samples* decreases the performance in terms of the complete and success rates. The unlikelihood loss also decreases the probability of decoding the `[name]` token by giving less probability mass on repeated tokens. The `name` string is also part of the belief state and, therefore, the probability of decoding it is lower. In order to solve the problem, the unlikelihood objective should be modified to ignore the 'name' string and the dataset filtering should relax the requirement on generating the `[name]` placeholder.

Perhaps the biggest performance gain was obtained from using the back-translations for dataset augmentation. By manually examining the paraphrased utterances, we have noticed that a lot of them were clearly wrong – from first 100 utterances we have estimated the percentage to be around one sixth. Nevertheless, when these noisy samples were used for the training the performance of the system increased. This shows that using more data is beneficial for the training of these large models even if the quality decreases slightly. On the other hand, this cannot be said for the labels, which need to be precise, as was demonstrated in experiments with cleaner version of the dataset in Section 6.3.

From the results of the ablation study (Section 6.3), it is clear that the newer consistency detection auxiliary task outperforms the SOLOIST's (Peng et al., 2020) by a wide margin. The reason for this is probably that the original consistency task was too easy for the model to learn. It is easy to detect the training sample inconsistency, because if the belief state is randomly resampled, its set of domains is likely different from the dialogue context. Detecting a random response is also a fairly simple task and the model is able to learn it quickly. By monitoring the training progress, we have noticed that the performance of the original consistency

task soon reached a very high value and the gradient was negligible compared to the cross-entropy loss part. Our proposed consistency detection is much more difficult for the model to learn. The model must pay attention to all slot values and validate them against the rest of dialogue context. This hypothesis was supported by the training performance, where the newer consistency detection task took much longer to converge.

7.3 Generating lexicalized responses

From careful examination of the output of the system in the error analysis described in Section 5.7, the delexicalization-lexicalization process has been identified as a main source of error in the system (see Section 6.7). An alternative has been proposed to abandon the delexicalization and pass the database results as the input to the model instead (see Section 4.4.2). In Section 6.5, we have seen that variants using delexicalization only for the *reference* slot outperformed the original system in terms of the success rate. It is, however, difficult to decide which variant is really better.

The problem with the original lexicalization is that is is ambiguous. It is difficult to pair placeholders in the delexicalized response with data from the database. This was demonstrated in Figure 6.4. We cannot design more specific placeholders, e.g., by adding the entity index to the placeholder, because we do not have the annotation that would enable us to do that. It is difficult to even pair the placeholders with the database results during training due to the mismatch between the dataset and the database (see Section 5.1.1). A second problem is that the delexicalization-lexicalization process may damage the grammatical structure of the sentence. For example, the `fee` placeholder could have values such as: '2 GBP', '4 GBP', or 'free'. This could lead to sentences such as: 'The parking costs free per day'.

The implicit lexicalization overcomes the problem with ungramatical surface realisation. Unfortunately, as was demonstrated in Examples 6.5 and 6.6, hallucination is a common culprit of these approaches. During training, we used the natural database order when generating first n texts in the string representation of the database results. When users underspecified the search query, for example if they stated they did not have a preference on the food type in a restaurant search, the system had too many options to choose from, but the ones occuring in the training data sometimes were not among first n entities returned from the database. This forces the model to memorize the data from the training set instead of using the database. Also, when the model was trained on the delexicalized responses, it was working with more general data, i.e., the data was less sparse. When implicit lexicalization is used, the risk of overfitting increases substantially.

To solve the first problem, we could use a better ordering of the database results before we form the database results string. The results could be ordered in such a way that the entities which are left at the end of the dialogue – after the user specifies all constraints – would be among the first ones. This would help because when the system recommends something, the user usually accepts the proposal and the entity gets converted into the belief state in the next dialogue turn. Therefore, if we used the belief state from future turns to sort the database results, we would increase the probability of the recommended entity (in system

response) being in the database result string, and the model would be less prone to hallucination. To tackle the problem of using less general data, we could augment the training samples by delexicalizing the responses and lexicalizing back with random values. Unfortunately, this could bring back some problems of the traditional lexicalization.

Finally, while generating lexicalized responses could offer higher response quality, in its current form the approach is less predictable. The proposed improvements could, however, prove the viability of the approach. The reason why it was not selected as the best dialogue system variant during our research was that we have designed the system after the DSTC 9 challenge. Also, when lexicalized responses are generated, we cannot use the MultiWOZ evaluation (see Section 5.4.1).

7.4 Human analysis

The proposed model performed well compared to other competitors in the DSTC 9 challenge (see Section 6.6). The detailed results, however, provided us with some insight into the weak spots of our method. While the method had the highest success rate judged by users, it lacked in database grounding. This suggests that the model could be hallucinating more than alternative approaches.

From our own system analysis in Section 6.7, we obtained a success rate of 86.9%, which is slightly higher than the success rate of 82.6% obtained from the DSTC 9 challenge. This discrepancy could have been caused by our expert annotators being more motivated to let the system recover from an error than crowdsourced workers. The analysis further suggest that the belief state generation was the most frequent source of error by a wide margin. In this case, the likely cause of the problem is the noise in the dataset annotation.[1] Also, the system was able to recover from most errors suggesting some robustness.

By far the most frequent error type was hallucination. This proves our suspicion raised based on the DSTC 9 results, and database grounding. There are multiple possible causes for hallucination. One of them is the use of the unlikelihood loss for the response predictor and nucleus sampling for the response generation. However, this cause is not very plausible since different variants without unlikelihood loss and with different sampling strategies have been experimented with thoroughly. An alternative cause of the problem could be in the belief state annotations of the dataset. If the labels are too noisy, the system does not learn to rely on them, which may cause the hallucination problem. The errors in the dataset could also cause the third most frequent error – missing information. In some dialogues in the training set, the system makes the booking without confirming all the information. The data-driven approach then learns these biases.

Finally, delexicalization could also a common source of error. It is, however, difficult to verify if it is the case. Methods that perform well in the MultiWOZ evaluation may produce poor responses after the response is lexicalized, because the evaluation only considers the presence of particular placeholders. The user simulator could provide a more realistic evaluation, however, the simulator often

[1] Recently, newer versions of the MultiWOZ dataset were published (Zang et al., 2020; Han et al., 2020; Ye et al., 2021) which fix some of the errors in the dataset annotation.

generates low-quality dialogue. Furthermore, the simulated user evaluation only gives a crude indication of the quality of the generated responses.

8. Conclusion

In this thesis, we fulfiled all objectives stated in Chapter 1. We proposed a task-oriented dialogue system called AuGPT based on the GPT-2 pre-trained transformer-based language model (Radford et al., 2019). The system uses two-stage decoding. First, we use the language model to autoregressively decode the string representation of the belief state, which comprises the dialogue history and is used for querying the database. Then, we pass the dialogue context, the belief state, and the number of database results as inputs to the same language model and decode a delexicalized response – a response where all slot values such as names of hotels are replaced with placeholders. Then, we use the database results from the previous step to replace the placeholders in the delexicalized response.

The presented system extends the original SOLOIST (Peng et al., 2020) dialogue system by introducing several contributions:

- To increase the diversity of the training dataset, we use the back-translation process, where we translate the entire dataset into several languages and back to English to obtain paraphrases. During training, we randomly resample each utterance from the set of available paraphrases.
- While we use the standard cross-entropy loss to train the belief state and response predictor, we also use the unlikelihood loss for the response. Similarly, we decode the belief state greedily, but we use nucleus sampling for the response.
- We introduce different auxiliary training tasks to help with the optimization process. One auxiliary task, which was originally designed by Peng et al. (2020), corrupts half of the training dataset and trains a binary classifier to detect if the dialogue is corrupted or not. We suggest replacing the SOLOIST's consistency detection auxiliary task with a modified version that outperforms the original one by a wide margin.
- We use automated metrics to filter the training dataset to reduce the noise in the dataset's annotations and increase the trained model's quality.
- We propose a different approach to incorporating the database results in the generated responses. Instead of using the delexicalized responses, we pass the database results as the input to the language model and allow it to generate the lexicalized response.

We carefully evaluated the dialogue system using both automatic evaluation and manual analysis. An ablation study assessed the importance of each contribution. Also, each decoding step of the system was evaluated individually, providing insight into the source of errors in the dialogue system. According to the study results, training data augmentation using back-translation via multiple languages and a modified auxiliary training objective for dialogue consistency detection contributed the most to AuGPT's performance. We used both the MultiWOZ corpus-based evaluation (Budzianowski et al., 2018) and the ConvLab 2 simulated user evaluation (Eric et al., 2020) to compare the method to state-of-the-art approaches in terms of the quality of generating correct delexicalized responses.

A variant of the AuGPT dialogue system placed third out of ten in the DSTC 9 challenge (Gunasekara et al., 2020), showing satisfactory results and providing detailed human analysis. To gain more insight into the dialogue system's behaviour, a team of expert annotators manually evaluated the system, providing qualitative and quantitative results.

The AuGPT dialogue system was designed in collaboration with my colleagues, who helped me with dataset loading and preprocessing and with training the models on GPU clusters. After taking part in the DSTC 9 challenge, we have presented the AuGPT dialogue system on the DSTC 9 workshop at AAAI 2021 conference and wrote a paper that is currently under submission (Kulhánek et al., 2021). We made the source code and pre-trained models publicly available[1] and included them also in the thesis (see Appendix A.2 for more details).

Future research should investigate implicit lexicalization further. As was suggested in Chapter 7, ordering the database results to match the response and using augmentation by replacing the concrete values in the dialogues with different, randomly sampled ones should be explored. To improve the transfer to new domains, one can also learn domain embeddings and optimize them together with the model. Also, exploring latent representations of the belief state and optimizing them jointly with the system may be an interesting area of research. The latent belief state representation could help the system overcome some errors in the dataset annotations.

[1]The source code is available at `https://github.com/ufal/augpt`, while the trained models are available at `https://huggingface.co/jkulhanek`.

Bibliography

Yoav Artzi and Luke Zettlemoyer. 2011. Bootstrapping semantic parsers from conversations. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 421–432, Edinburgh, Scotland, UK. Association for Computational Linguistics.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. 2020. wav2vec 2.0: A framework for self-supervised learning of speech representations. *arXiv preprint arXiv:2006.11477*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations (ICLR2015)*, San Diego, CA, USA.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Daniel G Bobrow, Ronald M Kaplan, Martin Kay, Donald A Norman, Henry Thompson, and Terry Winograd. 1977. GUS, a frame-driven dialog system. *Artificial intelligence*, 8(2):155–173.

Paweł Budzianowski and Ivan Vulić. 2019. Hello, it's GPT-2 – how can I help you? towards the use of pretrained language models for task-oriented dialogue systems. In *Proceedings of the 3rd Workshop on Neural Generation and Translation (WNGT)*, pages 15–22, Hong Kong.

Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Inigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gašić. 2018. MultiWOZ – a large-scale multi-domain Wizard-of-Oz dataset for task-oriented dialogue modelling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, page 5016–5026, Brussels, Belgium.

Stephan Busemann and Helmut Horacek. 1998. A flexible shallow approach to text generation. In *Natural Language Generation*.

Bill Byrne, Karthik Krishnamoorthi, Chinnadhurai Sankar, Arvind Neelakantan, Daniel Duckworth, Semih Yavuz, Ben Goodrich, Amit Dubey, Kyu-Young Kim, and Andy Cedilnik. 2019. Taskmaster-1: Toward a realistic and diverse dialog dataset. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, page 4516–4525, Hong Kong.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation.

In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 4171–4186, Minneapolis, MN, USA.

Bhuwan Dhingra, Lihong Li, Xiujun Li, Jianfeng Gao, Yun-Nung Chen, Faisal Ahmad, and Li Deng. 2017. Towards end-to-end reinforcement learning of dialogue agents for information access. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 484–495.

Ondřej Dušek and Filip Jurčíček. 2016. Sequence-to-sequence generation for spoken dialogue via deep syntax trees and strings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 45–51, Berlin, Germany. Association for Computational Linguistics.

Ondřej Dušek and Filip Jurcicek. 2019. Neural generation for Czech: Data and baselines. In *Proceedings of the 12th International Conference on Natural Language Generation*, pages 563–574.

Markus Eberts and Adrian Ulges. 2019. Span-based joint entity and relation extraction with transformer pre-training. *arXiv preprint arXiv:1909.07755*.

Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. 2018. Understanding back-translation at scale. In *Proceedings of the 2018 EMNLP*, pages 489–500, Brussels, Belgium.

Arash Einolghozati, Sonal Gupta, Mrinal Mohit, and Rushin Shah. 2019. Improving robustness of task oriented dialog systems. *arXiv preprint arXiv:1911.05153*.

Henry Elder, Robert Burke, Alexander O'Connor, and Jennifer Foster. 2020. Shape of synth to come: Why we should use synthetic data for english surface realization. In *Proceedings of the 58th ACL*, pages 7465–7471, Online.

Mihail Eric, Rahul Goel, Shachi Paul, Adarsh Kumar, Abhishek Sethi, Peter Ku, Anuj Kumar Goyal, Sanchit Agarwal, Shuyang Gao, and Dilek Hakkani-Tur. 2020. MultiWOZ 2.1: A consolidated multi-domain dialogue dataset with state corrections and state tracking baselines. In *Proceedings of the 12th Language Resources and Evaluation Conference (LREC)*, pages 422–428, Marseille, France.

Mihail Eric, Lakshmi Krishnan, Francois Charette, and Christopher D. Manning. 2017. Key-value retrieval networks for task-oriented dialogue. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, page 37–49, Saarbrücken, Germany.

Christian Federmann, Oussama Elachqar, and Chris Quirk. 2019. Multilingual whispers: Generating paraphrases with translation. In *Proceedings of the 5th Workshop on Noisy User-generated Text (W-NUT 2019)*, pages 17–26, Hong Kong.

Jianfeng Gao, Michel Galley, and Lihong Li. 2019a. *Neural Approaches to Conversational AI: Question Answering, Task-oriented Dialogues and Social Chatbots*. Now Foundations and Trends.

Xiang Gao, Yizhe Zhang, Sungjin Lee, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. 2019b. Structuring latent spaces for stylized response generation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1814–1823.

Sebastian Gehrmann, Yuntian Deng, and Alexander M Rush. 2018. Bottom-up abstractive summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4098–4109.

Sergey Golovanov, Rauf Kurbanov, Sergey Nikolenko, Kyryl Truskovskyi, Alexander Tselousov, and Thomas Wolf. 2019. Large-scale transfer learning for natural language generation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6053–6058, Florence, Italy. Association for Computational Linguistics.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Alex Graves. 2013. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

Claudio Greco, Barbara Plank, Raquel Fernández, and Raffaella Bernardi. 2019. Psycholinguistics meets continual learning: Measuring catastrophic forgetting in visual question answering. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3601–3605, Florence, Italy. Association for Computational Linguistics.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor OK Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1631–1640.

Chulaka Gunasekara, Seokhwan Kim, Luis Fernando D'Haro, Abhinav Rastogi, Yun-Nung Chen, Mihail Eric, Behnam Hedayatnia, Karthik Gopalakrishnan, Yang Liu, Chao-Wei Huang, Dilek Hakkani-Tür, Jinchao Li, Qi Zhu, Lingxiao Luo, Lars Liden, Kaili Huang, Shahin Shayandeh, Runze Liang, Baolin Peng, Zheng Zhang, Swadheen Shukla, Minlie Huang, Jianfeng Gao, Shikib Mehri, Yulan Feng, Carla Gordon, Seyed Hossein Alavi, David Traum, Maxine Eskenazi, Ahmad Beirami, Eunjoon, Cho, Paul A. Crook, Ankita De, Alborz Geramifard, Satwik Kottur, Seungwhan Moon, Shivani Poddar, and Rajen Subba. 2020. Overview of the ninth dialog system technology challenge: DSTC9. *arXiv preprint arXiv:2011.06486*.

Dilek Hakkani-Tür, Gokhan Tur, Asli Celikyilmaz, Yun-Nung Chen, Jianfeng Gao, Li Deng, and Ye-Yi Wang. 2016. Multi-domain joint semantic frame parsing using bi-directional rnn-lstm. In *Interspeech 2016*, pages 715–719.

Donghoon Ham, Jeong-Gwan Lee, Youngsoo Jang, and Kee-Eung Kim. 2020. End-to-end neural pipeline for goal-oriented dialogue systems using GPT-2. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 583–592, Online.

Ting Han, Ximing Liu, Ryuichi Takanobu, Yixin Lian, Chongxuan Huang, Wei Peng, and Minlie Huang. 2020. MultiWOZ 2.3: A multi-domain task-oriented dataset enhanced with annotation corrections and co-reference annotation. *arXiv preprint arXiv:2010.05594*.

Matthew Henderson, Blaise Thomson, and Jason D Williams. 2014. The second dialog state tracking challenge. In *Proceedings of the 15th annual meeting of the special interest group on discourse and dialogue (SIGDIAL)*, pages 263–272.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. In *Proceedings of the International Conference on Learning Representations (ICLR)*, Online.

Eric Horvitz and Tim Paek. 1999. A computational architecture for conversation. In *UM99 User Modeling*, pages 201–210. Springer.

Ehsan Hosseini-Asl, Bryan McCann, Chien-Sheng Wu, Semih Yavuz, and Richard Socher. 2020. A simple language model for task-oriented dialogue. *arXiv preprint arXiv:2005.00796*.

Minlie Huang, Xiaoyan Zhu, and Jianfeng Gao. 2020. Challenges in building intelligent open-domain dialog systems. *ACM Transactions on Information Systems (TOIS)*, 38(3):1–32.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional LSTM-CRF models for sequence tagging. *arXiv preprint arXiv:1508.01991*.

Shaojie Jiang and Maarten de Rijke. 2018. Why are sequence-to-sequence models so dull? understanding the low-diversity problem of chatbots. In *Proceedings of the 2018 EMNLP Workshop SCAI: The 2nd International Workshop on Search-Oriented Conversational AI*, pages 81–86.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. TinyBERT: Distilling BERT for natural language understanding. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pages 4163–4174.

Lifeng Jin, David King, Amad Hussein, Michael White, and Douglas Danforth. 2018. Using paraphrasing and memory-augmented models to combat data sparsity in question interpretation with a virtual patient dialogue system. In *Proceedings of the Thirteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 13–23, New Orleans, Louisiana. Association for Computational Linguistics.

Rohit Kate and Raymond Mooney. 2006. Using string-kernels for learning semantic parsers. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 913–920.

John F Kelley. 1984. An iterative design methodology for user-friendly natural language office information applications. *ACM Transactions on Information Systems (TOIS)*, 2(1):26–41.

Young-Bum Kim, Sungjin Lee, and Karl Stratos. 2017. ONENET: Joint domain, intent, slot prediction for spoken language understanding. In *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 547–553. IEEE.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Ioannis Konstas, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer. 2017. Neural AMR: Sequence-to-sequence models for parsing and generation. In *Proceedings of the 55th ACL*, pages 146–157, Vancouver, Canada.

Jonáš Kulhánek, Vojtěch Hudeček, Tomáš Nekvinda, and Ondřej Dušek. 2021. Augpt: Dialogue with pre-trained language models and data augmentation. *arXiv preprint arXiv:2102.05126*.

John D Lafferty, Andrew McCallum, and Fernando C N Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, page 282–289, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Minh Le Nguyen, Akira Shimazu, and Xuan-Hieu Phan. 2006. Semantic parsing with structured SVM ensemble classification models. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 619–626.

Sungjin Lee, Qi Zhu, Ryuichi Takanobu, Zheng Zhang, Yaoqin Zhang, Xiang Li, Jinchao Li, Baolin Peng, Xiujun Li, Minlie Huang, et al. 2019. ConvLab: Multi-domain end-to-end dialog system platform. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 64–69.

Wenqiang Lei, Xisen Jin, Min-Yen Kan, Zhaochun Ren, Xiangnan He, and Dawei Yin. 2018. Sequicity: Simplifying task-oriented dialogue systems with single sequence-to-sequence architectures. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1437–1447, Melbourne, Australia.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Margaret Li, Stephen Roller, Ilia Kulikov, Sean Welleck, Y.-Lan Boureau, Kyunghyun Cho, and Jason Weston. 2020. Don't say that! making inconsistent dialogue unlikely with unlikelihood training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, page 4715–4728, Online.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Bing Liu and Ian Lane. 2016. Attention-based recurrent neural network models for joint intent detection and slot filling. *Interspeech 2016*, pages 685–689.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Nurul Lubis, Christian Geishauser, Michael Heck, Hsien-chin Lin, Marco Moresi, Carel van Niekerk, and Milica Gasic. 2020. LAVA: Latent action spaces via variational auto-encoding for dialogue policy optimization. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 465–479, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421.

Dominik Macháček, Jonáš Kratochvíl, Sangeet Sagar, Matúš Žilinec, Ondřej Bojar, Thai-Son Nguyen, Felix Schneider, Philip Williams, and Yuekun Yao. 2020. ELITR non-native speech translation at IWSLT 2020. In *Proceedings of the 17th International Conference on Spoken Language Translation (IWSLT)*, page 200–208, Online.

Andrea Madotto, Zhaojiang Lin, Chien-Sheng Wu, Jamin Shin, and Pascale Fung. 2020. Attention over parameters for dialogue systems. *arXiv preprint arXiv:2001.01871*.

François Mairesse and Steve Young. 2014. Stochastic language generation in dialogue using factored language models. *Computational Linguistics*, 40(4):763–799.

Shikib Mehri, Tejas Srinivasan, and Maxine Eskenazi. 2019. Structured fusion networks for dialog. In *Proceedings of the 20th Annual SIGdial Meeting on Discourse and Dialogue*, pages 165–177, Stockholm, Sweden.

Helen M Meng, Carmen Wai, and Roberto Pieraccini. 2003. The use of belief networks for mixed-initiative dialog modeling. *IEEE Transactions on Speech and Audio Processing*, 11(6):757–773.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. 2018. Mixed precision training. In *Proceedings of the ICLR*, Vancouver, Canada.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Rafael Müller, Simon Kornblith, and Geoffrey Hinton. 2019. When does label smoothing help? *arXiv preprint arXiv:1906.02629*.

Aaron Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George Driessche, Edward Lockhart, Luis Cobo, Florian Stimberg, et al. 2018. Parallel wavenet: Fast high-fidelity speech synthesis. In *International conference on machine learning*, pages 3918–3926. PMLR.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 311–318, Philadelphia, PA, USA.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*, pages 8024–8035. Vancouver, Canada.

Baolin Peng, Chunyuan Li, Jinchao Li, Shahin Shayandeh, Lars Liden, and Jianfeng Gao. 2020. SOLOIST: Few-shot task-oriented dialog with a single pre-trained auto-regressive model. *arXiv preprint arXiv:2005.05298*.

Stephen G Pulman. 1996. Conversational games, belief revision and bayesian networks. In *Proceedings of the 7th Computational Linguistics in the Netherlands meeting*.

Libo Qin, Xiao Xu, Wanxiang Che, Yue Zhang, and Ting Liu. 2020. Dynamic fusion network for multi-domain end-to-end task-oriented dialog. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6344–6354, Online. Association for Computational Linguistics.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. Technical report, OpenAI.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. Technical report, OpenAI.

Osman Ramadan, Paweł Budzianowski, and Milica Gasic. 2018. Large-scale multi-domain belief tracking with knowledge sharing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 432–437.

Abhinav Rastogi, Xiaoxue Zang, Srinivas Sunkara, Raghav Gupta, and Pranav Khaitan. 2020. Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 8689–8696, New York, NY, USA.

Ehud Reiter and Robert Dale. 1997. Building applied natural language generation systems. *Natural Language Engineering*, 3(1):57–87.

Jost Schatzmann, Karl Weilhammer, Matt Stuttle, and Steve Young. 2006. A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies. *Knowledge Engineering Review*, 21(2):97–126.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 86–96, Berlin, Germany.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

David Suendermann, Keelan Evanini, Jackson Liscombe, Phillip Hunter, Krishna Dayanidhi, and Roberto Pieraccini. 2009. From rule-based to statistical grammars: Continuous improvement of large-scale spoken dialog systems. In *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4713–4716. IEEE.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27 (NeurIPS)*, pages 3104–3112. Montréal, Canada.

Alan Turing. 1950. Mind. *Mind*, 59(236):433–460.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5998–6008, Long Beach, CA, USA.

Yuxuan Wang, RJ Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, et al. 2017. Tacotron: Towards end-to-end speech synthesis. *arXiv preprint arXiv:1703.10135*.

Sean Welleck, Ilia Kulikov, Stephen Roller, Emily Dinan, Kyunghyun Cho, and Jason Weston. 2020. Neural text generation with unlikelihood training. In *Proceedings of the International Conference on Learning Representations (ICLR)*, Online.

TH Wen, M Gašić, N Mrkšić, PH Su, D Vandyke, and S Young. 2015a. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. In *Conference Proceedings-EMNLP 2015: Conference on Empirical Methods in Natural Language Processing*, pages 1711–1721.

Tsung-Hsien Wen, Milica Gasic, Dongho Kim, Nikola Mrkšić, Pei-Hao Su, David Vandyke, and Steve Young. 2015b. Stochastic language generation in dialogue using recurrent neural networks with convolutional sentence reranking. In *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGdial)*, pages 275–284, Prague, Czechia.

Tsung-Hsien Wen, Milica Gašić, Nikola Mrkšić, Lina M. Rojas-Barahona, Pei-Hao Su, Stefan Ultes, David Vandyke, and Steve Young. 2016a. Conditional generation and snapshot learning in neural dialogue systems. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2153–2162, Austin, TX, USA.

Tsung-Hsien Wen, Milica Gašic, Nikola Mrkšic, Lina M Rojas-Barahona, Pei-Hao Su, David Vandyke, and Steve Young. 2016b. Multi-domain neural network language generation for spoken dialogue systems. In *Proceedings of NAACL-HLT*, pages 120–129.

Tsung-Hsien Wen, Milica Gašić, Nikola Mrkšić, Pei-Hao Su, David Vandyke, and Steve Young. 2015c. Semantically conditioned LSTM-based natural language generation for spoken dialogue systems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1711–1721, Lisbon, Portugal. Association for Computational Linguistics.

Tsung-Hsien Wen, David Vandyke, Nikola Mrkšić, Milica Gašić, Lina M. Rojas-Barahona, Pei-Hao u, Stefan Ultes, and Steve Young. 2017. A network-based end-to-end trainable task-oriented dialogue system. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 438–449, Valencia, Spain.

Jason D Williams, Kavosh Asadi Atui, and Geoffrey Zweig. 2017. Hybrid Code Networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 665–677.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019a. HuggingFace's Transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Thomas Wolf, Victor Sanh, Julien Chaumond, and Clement Delangue. 2019b. TransferTransfo: A transfer learning approach for neural network based conversational agents. *arXiv preprint arXiv:1901.08149*.

Chien-Sheng Wu, Steven Hoi, Richard Socher, and Caiming Xiong. 2020. ToD-BERT: Pre-trained natural language understanding for task-oriented dialogues.

In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, page 917–929, Online.

Chien-Sheng Wu, Andrea Madotto, Ehsan Hosseini-Asl, Caiming Xiong, Richard Socher, and Pascale Fung. 2019. Transferable multi-domain state generator for task-oriented dialogue systems. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 808–819.

Fanghua Ye, Jarana Manotumruksa, and Emine Yilmaz. 2021. MultiWOZ 2.4: A multi-domain task-oriented dialogue dataset with essential annotation corrections to improve state tracking evaluation. *arXiv preprint arXiv:2104.00773*.

Steve Young, Milica Gašić, Simon Keizer, François Mairesse, Jost Schatzmann, Blaise Thomson, and Kai Yu. 2010. The hidden information state model: A practical framework for POMDP-based spoken dialogue management. *Computer Speech & Language*, 24(2):150–174.

Steve Young, Milica Gašić, Blaise Thomson, and Jason D Williams. 2013. POMDP-based statistical spoken dialog systems: A review. *Proceedings of the IEEE*, 101(5):1160–1179.

Xiaoxue Zang, Abhinav Rastogi, and Jindong Chen. 2020. MultiWOZ 2.2: A dialogue dataset with additional annotation corrections and state tracking baselines. In *Proceedings of the 2nd Workshop on Natural Language Processing for Conversational AI*, pages 109–117.

Luke Zettlemoyer and Michael Collins. 2007. Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 678–687.

Yichi Zhang, Zhijian Ou, Huixin Wang, and Junlan Feng. 2020a. A probabilistic end-to-end task-oriented dialog model with latent belief states towards semi-supervised learning. In *Proceedings of the 2020 EMNLP*, page 9207–9219, Online.

Yichi Zhang, Zhijian Ou, and Zhou Yu. 2020b. Task-oriented dialog systems that consider multiple appropriate responses under the same context. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 9604–9611, New York, NY, USA.

Yizhe Zhang, Siqi Sun, Michel Galley, Yen-Chun Chen, Chris Brockett, Xiang Gao, Jianfeng Gao, Jingjing Liu, and Bill Dolan. 2020c. DIALOGPT : Large-scale generative pre-training for conversational response generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL): System Demonstrations*, pages 270–278, Online.

Qi Zhu, Zheng Zhang, Yan Fang, Xiang Li, Ryuichi Takanobu, Jinchao Li, Baolin Peng, Jianfeng Gao, Xiaoyan Zhu, and Minlie Huang. 2020. ConvLab-2: An open-source toolkit for building, evaluating, and diagnosing dialogue systems. In *Proceedings of the 58th ACL: System Demonstrations*, pages 142–149, Online.

List of Figures

List of Tables

List of Abbreviations

CCG combinatory categorial grammar 15

CRF conditional random field 15

DB database 22, 40

DNN deep neural network 4, 7, 8, 10, 16

DST dialogue state tracking 14, 17, 37, 39, 46, 47, 49

ELBO variational evidence lower bound 18

LM language model 6, 7, 13, 20, 22–26, 28

NLG natural language generation 14, 15, 17, 37–40, 46, 47, 49

NLP natural language processing 6, 9, 31

NLU natural language understanding 14, 15, 17, 19, 21, 36, 37, 40, 48

RNN recurrent neural network 8–10, 16, 21

SVM support vector machine 15

A. Attachments

A.1 Implicit lexicalization details

column name	type	n
name	text	3
address	text	2
department	text	1
reference	text	1
phone	text	1
price	text	3
car	text	1
time	text	1
postcode	text	1
leave at	text	4
arrive by	text	4
price range	categorical	3
area	categorical	3
start	categorical	3
food	categorical	3
type	categorical	3

Table A.1: This table displays the details on the formatting of each column used in the database result formatting for implicit lexicalization. For the text column type, we return the first n entities, or if the total number of entities is larger than n, we return the first $n-1$ entities followed by '...'. For the categorical column type, we return top-n most frequently occurring categories, and if the total number of categories is larger, we add '...'.

A.2 AuGPT framework source code

The attached content contains the source code needed to train and evaluate the dialogue system. In this section, we give short introduction into how to use the source code. For the rest of the section, we assume that the attached content is saved in a folder denoted as `<augpt>`.

Preparing the development environment

In order to use AuGPT, you have to have Python 3.7 installed. First, navigate to `<augpt>` folder and install the required packages by running the following command:

```
pip install -r requirements.txt
```

Downloading datasets

To download datasets, run `scripts/download_<dataset>.py`, where `<dataset>` is the name of the dataset you want to download. Supported datasets:

1. *taskmaster*: The Taskmaster corpus (Byrne et al., 2019) comprising over 55,000 spoken and written task-oriented dialogs in over a dozen domains.
2. *schemaguided*: The Schema-Guided Dialogue (Rastogi et al., 2020) dataset consisting of over 20k annotated multi-domain, task-oriented conversations between a human and a virtual assistant.
3. *multiwoz*: The MultiWOZ 2.0 dataset (Budzianowski et al., 2018) - a large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling.
4. *convlab_multiwoz*: The MultiWOZ 2.1 dataset (Eric et al., 2020) - a cleaner version of MultiWOZ 2.0 with span information.

In this work, we denote the union of *taskmaster* and *schemaguided* as *bigdata*.

Interact and generate

To run the model in interactive mode, you can use `interact.py` utility. Alternatively, to use the model in your code, you can modify the following code:

```
import pipelines  # Required here, modifies the transformers
#                   package to support AuGPT pipeline.
import transformers

# Loads the pipeline with MultiWOZ 2.1 model
pipeline = transformers.pipeline('augpt-conversational',
                                 '<checkpoint>')

# Either AuGPTConversation or Conversation can be used
conversation = pipelines.AuGPTConversation('Hi, I need a hotel')

conversation = conversation(pipeline)
print(conversation.generated_responses[-1])
```

The `<checkpoint>` denotes the path to trained model checkpoint. You can use `<augpt>/checkpoints/augpt-mw-21`.

To generate the predictions, use `generate.py` script.

```
./generate.py --model <augpt>/checkpoints/augpt-mw-21 \
              --dataset multiwoz-2.1-test \
              --file predictions.txt
```

Training and evaluation

The following scripts creates a virtual environment and installs required packages for training and ConvLab-2 evaluation.

```
python -m venv ~/envs/dstc
source ~/envs/dstc/bin/activate
pip install -r requirements.txt
cd ~/source
git clone git@github.com:ufal/ConvLab-2.git
cd ConvLab-2
git reset --hard 8b4464c57de0fbc497ce3532532c30ae461906e9
pip install -e . --no-deps
python -m spacy download en_core_web_sm
```

Training bigdata model

The *bigdata* pre-trained model can be reproduced using the following command

```
./train.py --epochs 8 --restrict-domains \
     --train-dataset schemaguided-train+taskmaster-train \
     --dev-dataset schemaguided-dev+taskmaster-dev \
     --validation-steps 10000 --logging-steps 1000 \
     --warmup-steps 5000 --evaluation-dialogs 0 --fp16
```

The pre-trained model can also be downloaded from the Hugging Face model repository as `jkulhanek/augpt-bigdata`.

Fine-tuning on MultiWOZ

The pretrained model can be finetuned on MultiWOZ 2.1 dataset as follows:

```
./train_multiwoz.py --train-dataset multiwoz-2.1-train \
        --dev-dataset multiwoz-2.1-val \
        --model jkulhanek/augpt-bigdata \
        --backtranslations <augpt>/backtranslations/multiwoz.yaml \
        --response-loss unlikelihood \
        --epochs 10 --fp16 --clean-samples
```

For MultiWOZ 2.0, substitute the correct dataset version.

Distributed training

To start the training on single CPU node (for testing), run the training with the following arguments:

```
./train.py --no-cuda --gradient-accumulation-steps 4
```

NOTE: For optimal performance at least *four* GPUs are required for training. To run the training with single GPU:

```
./train.py --gradient-accumulation-steps 4
```

To run on single node with multiple GPUs, run the following command:

```
python -m torch.distributed.launch \
    --nproc_per_node=<NUM_GPUS_YOU_HAVE> train.py
```

In this case the expected number of GPUs is four, you may need to adjust `learning_rate` and/or `gradient-accumulation-steps` accordingly.
To run the training on multiple nodes with multiple GPUs, you can use pytorch launch utility https://pytorch.org/docs/stable/distributed.html#launch-utility. Alternatively, consult your job scheduling system. You may need to set the environment variables:
`LOCAL_RANK`, `RANK`, `WORLD_SIZE`, `MASTER_PORT`, `MASTER_ADDR`. In this case, `RANK` is global number of current process across the world and `LOCAL_RANK` is the number of each process running on single node. Every node is required to have as many GPUs as there are processes running on single machine.

Evaluation

ConvLab-2 evaluation

To evaluate your trained model using ConvLab-2 evaluation (Zhu et al., 2020), run the following script:

```
./evaluate_convlab.py --model <checkpoint>
```

MultiWOZ 2.x evaluation

To evaluate your trained model using MultiWOZ evaluation, run the following:

```
./evaluate_multiwoz.py --model <checkpoint> \
    --dataset multiwoz-2.1-test
```

If you have your predictions generated by running `generate.py` script, you can evaluate them by running:

```
./evaluate_multiwoz.py --file predictions.txt \
    --dataset multiwoz-2.1-test
```

For MultiWOZ 2.0, substitute the correct dataset version.