SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Information Systems

Augmenting Knowledge-Based Conversational Search Systems With Large Language Models

Manuel Klettner

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Information Systems

Augmenting Knowledge-Based Conversational Search Systems With Large Language Models

Erweiterung von wissensbasierten Konversationssuchsystemen mit großen Sprachmodellen

Author:	Manuel Klettner
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	M.Sc. Phillip Schneider
Submission Date:	October 15, 2023

I confirm that this master's thesis in information systems is my own work and I have documented all sources and material used.

Munich, October 15, 2023 Manuel Klettner

Acknowledgments

I want to seize this opportunity to express my gratitude to everyone who supported me through my academic journey, particularly during the process of writing this master's thesis. Specifically, I thank Prof. Dr. Florian Matthes, who enabled me to work on such an exciting topic with significant current relevance. Furthermore, I extend my appreciation to M.Sc. Phillip Schneider for consistently offering valuable guidance and advice throughout the creation of this thesis. Lastly, I would like to thank my family and friends, who unwaveringly supported me during my academic pursuit.

Abstract

Conversational interfaces are increasingly used by websites and apps, including office software as well as creativity tools. This trend reflects a shift towards natural language in human-computer interaction, which is also facilitated by utilizing Large Language Models to generate responses aligned with user intents. Although this approach is effective for tasks like text summarization and text generation, there are still some shortcomings. Issues include hallucination, outdated information, data lineage, and reproducibility. Integrating a knowledge base as a grounding mechanism can mitigate these problems in the context of conversational interfaces for information search. Such a system needs to fulfill two main tasks. First, it has to translate the user intent into a database query, also known as a specific form of semantic parsing. Subsequently, a response has to be generated, grounded on the retrieved data. This thesis empirically evaluates the suitability of Large Language Models for augmenting components of such a system. We analyze the effectiveness of four Large Language Models and different prompting techniques for semantic parsing on a conversational question-answering dataset called SPICE. We identify common errors in the generated SPARQL queries using automatic metrics and a human evaluation. Furthermore, we discuss strategies to mitigate such issues and enhance performance. Our experiments show that Large Language Models can generate database queries based on conversations. Furthermore, we observe that fine-tuning or few-shot prompting can substantially improve their semantic parsing capabilities. We assess four Large Language Models of varying sizes in combination with different prompting techniques for data-to-text generation. We utilize the triple verbalization benchmark called WebNLG to analyze their performance, reliability, and common issues of predicted texts. We demonstrate with our experiments that Large Language Models can create natural language text based on a set of related triples. The models' abilities can be improved significantly through fine-tuning techniques, few-shot prompting, and post-processing. We create two new model variations of LLaMA by fine-tuning it for semantic parsing and data-to-text generation. As part of this thesis, we share the corresponding model adapters created using Low-Rank Adaptation. These models outperform the 20-times larger GPT-3.5-Turbo model on the respective tasks.

Contents

1 Introduction

1.1 Motivation

Interest in conversational interfaces has risen significantly over the last few years. Apart from research, they have found their way into many people's lives and become increasingly integrated into consumer and enterprise applications. One product group that has become popular in recent years is voice assistants such as Amazon Alexa, Google Assistant, or Siri. They are integrated into smartphones, smart speakers, or headphones and can send commands to smart home devices like lights, blinds, and thermostats, play music, or answer general questions. Another category of applications is text-based conversational interfaces. These can be bots or plugins for communication platforms such as Slack, Discord, or Telegram and enable interactions with "systems" in addition to humans. These chatbots can deliver event notifications through text messages and empower users to request information or command external systems using written instructions. AI image generators are another group of text-based tools that became popular. They generate an image based on the user's textual description in the prompt. Two examples that deliver impressive results are Stable Diffusion and Dall-E3. Due to a paradigm shift in the last year, there is considerable attention on conversational agents based on Large Language Models (LLMs). Prior approaches lead to agents supporting only a limited variety of interactions and tasks, usually within a predefined domain. LLMs, on the other hand, provide increasing flexibility, with chatbots supporting users while learning languages, writing code, summarizing texts, writing essays, and searching for information.

In this thesis, we focus on text-based conversational search. Hence, we are interested in information retrieval based on dialogue interactions with multiple turns. Such systems help to reduce the search's result space iteratively to relevant information [1, 2]. Although various approaches are already available, current solutions still have downsides.

A common approach used before the rise of LLMs, referred to as the conventional approach, works as follows. For every type of request the conversational agent should handle, a corresponding user intent is defined by providing a list of examples. Furthermore, the agent's responses are specified by creating system utterances. Multiple alternative versions can be provided for each response type, and a single one is selected randomly at run time. These template responses can also contain placeholders filled with different information depending on the user's request. Stories connect intents and utterances. They describe the conversational flow between user and agent within one interaction. Examples of frameworks for developing such systems are Rasa, Amazon Lex, and Google Dialogue Flow. One problem with this approach is the agent's limited functionality and domain expertise since all supported interactions must be defined manually. Hence, only a restricted set of user intents can be

handled appropriately, while others lead to fallback responses that frequently fail to meet user expectations. Furthermore, template queries are needed to integrate external knowledge sources, such as information from a database. These queries need to be created manually and are enriched with entities extracted from the user intent during run time. This manual creation process naturally limits supported functionality. Additionally, some template adaptations might be required if the database schema changes. The rule-based nature of this approach can lead to frustrated users since they need to memorize supported functionality, and slight deviations in wording might lead to fallback responses.

A method that addresses the downsides of the restricted conventional approach and recently gained lots of attention utilizes Pre-Trained Language Models (PLMs), trained on vast amounts of data self-supervised by predicting tokens in natural language texts. Through this procedure, the model learns to understand and generate text while capturing knowledge contained in the training data. Scaling up the model size and training leads thereby to emergent capabilities of LLMs [3, 4]. The abilities of LLM can be refined further via fine-tuning on conversational data, instruction fine-tuning, or Reinforcement Learning from Human Feedback (RLHF) [5, 6]. Using these methods, the model learns to align with human preferences and extrapolates this behavior to unseen questions and tasks. Although this approach has considerable potential, some drawbacks and risks still need to be considered, especially in the area of conversational search. One issue when employing LLMs for information retrieval is the occurrence of hallucinations. This phenomenon refers to situations in which the model produces responses that sound convincing despite being factually incorrect. Another problem is the unreliability of responses, which might change when asking the same question multiple times, occasionally resulting in contradicting answers. Furthermore, intrinsic knowledge of models is restricted to their training data. Hence, any events that happened after training are unknown. Additionally, the predicted output lacks references to the sources from which the answer is derived, making the verification process challenging [7, 8]. Thus, relying on the currently available systems for information retrieval comes with substantial downsides and risks. Grounding mechanisms are required to mitigate these risks, ensuring truthful and reliable responses.

We address these issues by augmenting the traditional chatbot architecture with LLMs while utilizing a graph knowledge base to ground its responses. Therefore, we mainly focus on two tasks. Firstly, translating natural language user questions into database queries, also known as a specific form of semantic parsing. The second task, data-to-text generation, deals with creating a natural language response based on structured data. For these use cases, we evaluate LLMs optimized for conversational interactions, possessing various sizes and training methodologies. Furthermore, we perform a series of experiments with different prompting and post-processing techniques.

1.2 Problem Statement

With this thesis, we want to address common issues of conventional (Problem 1) as well as LLM-based (Problem 2-5) conversational search systems.

Problem 1 - Domain Specificity The functionality and domain are limited when using the conventional approach to create a conversational search system. Moreover, the system's complexity significantly increases when supporting multiple use cases and conversation flows. The cause of this issue is that user intents, system utterances, and stories need to be defined manually. Furthermore, template queries need to be created when integrating a database to acquire knowledge. Figure 1.1 shows an example of defining an intent, utterance, and story when using the Rasa framework to create a chatbot that responds to a greeting from the user.

Thus, while conversational agents employing LLMs can support users on a wide range of requests, the conventional approach is inherently limited, especially when dealing with unanticipated user requests.

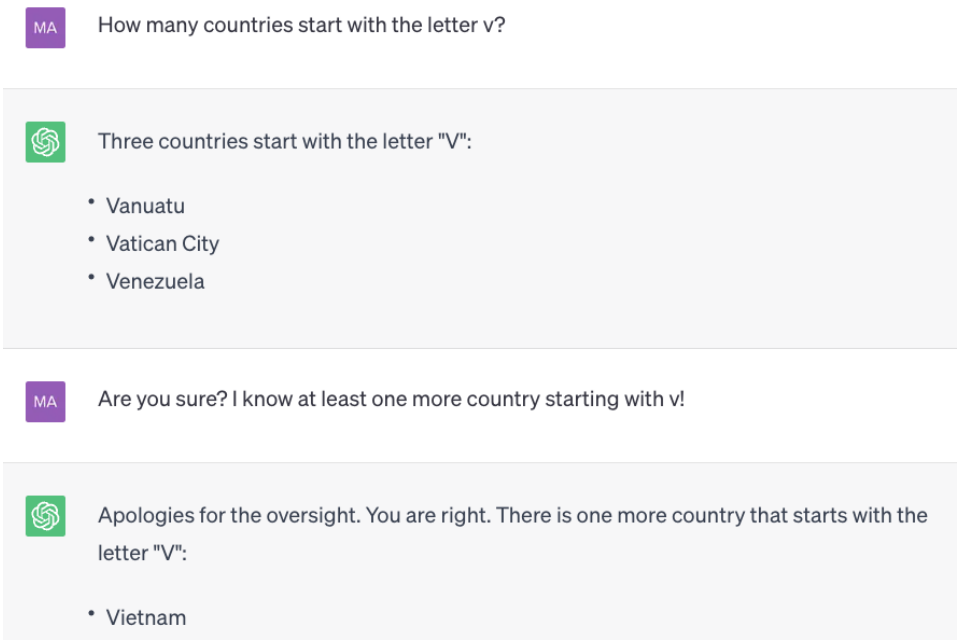
```yaml
nlu:
- intent: greet
  examples: |
    - hey
    - hello
    - hi
    ...

responses:
  utter_greet:
  - text: Hey, I'm a Bob your digital assistant. What can I do for you?
  - text: Hi! I'm your digital assistant Bob. How can help you today?

stories:
- story: happy path
  steps:
  - intent: greet
  - action: utter_greet

```

Figure 1.1: This figure shows the definition of a user intent (*greet*), a user utterance (*utter_greet*), and a story (*happy path*) for a chatbot created with the Rasa framework that responds to a greeting of the user. The list of examples for the intent is shortened to increase readability

Problem 2 - Hallucinations As LLMs learn how an expected answer looks like during training, it sometimes creates responses that sound convincing to users but are factually incorrect. In some instances, this is due to lacking knowledge of the model; in other cases, the model contains the correct information but does not include it in the response. Thus, when users trust these systems, it can lead to negative societal implications, such as misinformation and dissemination of conspiracy theories. An example of such a hallucination occurred in Google's announcement of their chatbot Bard. When it was asked about discoveries from the James Webb Space Telescope, it responded, among other things, with: "JWST took the very first pictures of a planet outside of our own solar system" [9]. According to NASA's website, this is factually incorrect as it states that the first image was taken in 2004 and hence before

the James Webb Space Telescope became operational [10].

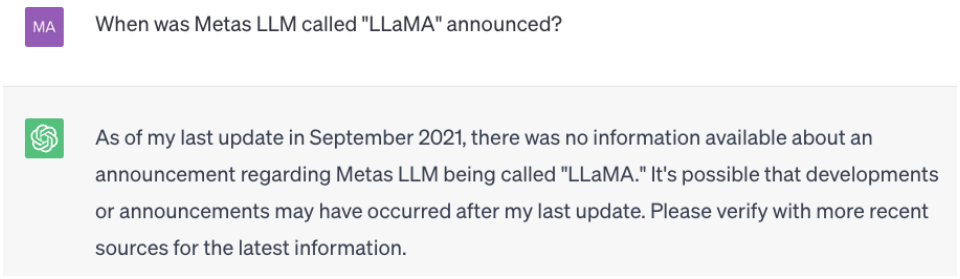
Problem 3 - Reproducibility Asking the chatbot multiple times the same question leads to different answers that are not reliably reproducible. In some instances, these responses are just varying formulations of the same content, but in other cases, they are inconsistent, including information that might contradict each other. When the model returns a faulty response, it may occasionally recognize its mistake after being asked about the correctness of its response. This issue demonstrates that incorrect responses are not always caused by a lack of knowledge. Figure 1.2 shows such an example using ChatGPT.



Source: This image was created using ChatGPT (GPT-3.5) in June 2023 [11].

Figure 1.2: In this figure, we ask ChatGPT about countries starting with the letter v, and it returns a wrong response but recognizes its error when asking for clarification.

Problem 4 - Outdated Information LLMs acquire knowledge during training of the model weights. Since the weights are fixed during inference time, it does not learn new information while users interact with it. Hence, the knowledge of a LLM is limited by its training data. Once the model is trained, intrinsic information can only be updated through additional training or fine-tuning, which requires significant computational resources, time, and energy. When we ask ChatGPT about a recent event, it responds that it was only updated with new information until September 2021. This example is displayed in Figure 1.3.



Source: This image was created using ChatGPT (GPT-3.5) in June 2023 [11].

Figure 1.3: In this figure, we ask ChatGPT about the announcement of Metas "LLaMA" and receive the response that it does not know due to a knowledge cutoff date

Problem 5 - Data Lineage To gain an understanding of language, LLMs are trained on vast amounts of data. In the inference process, the generated responses can draw upon information from many documents that constitute the underlying training data. As users have no insight into the sources of the generated response, they do not know if the answer is trustworthy and correct. Hence, one needs to verify the responses with external means, limiting its usefulness as a search engine. Furthermore, quality, trustworthiness, and possible biases of training data can not be assessed by users due to the large quantities of documents and because most companies do not publish information about their training process.

Scope Above, we have identified one problem with employing the traditional approach for conversational search systems and highlighted four challenges when utilizing a strictly LLM-based method. To address these limitations, in this thesis, we consider a system that combines both approaches by augmenting individual components of a conventional chatbot with LLMs. Figure 1.4 visualizes such a system. The input is a textual prompt containing a natural language question. The Natural Language Understanding (NLU) component, shown on the left, handles this request. It is responsible for translating the question into a database query, a task also known as semantic parsing. Subsequently, the query is sent to a knowledge base to acquire relevant information. Afterward, the retrieved data is forwarded to the dialogue management component. It keeps track of previous interactions with the user, including a history of states and entities based on the current question, data received from the NLU component, as well as former requests and responses. This component decides in what manner the conversational agent should respond. To be more specific, it does not generate a concrete answer text but rather decides the type of response the system should provide. This answer could be, for example, a clarifying question or the requested information. Due to limited time, a dedicated dialogue management component is out of the scope of this thesis. For state and entity tracking, we rely on the capabilities of the LLM used for NLU. A dedicated dialogue management approach can be addressed in future work. Finally, the user question and retrieved data from the knowledge base are used to create an answer. The response is returned in natural language to ensure user-friendliness. The part of the system

responsible for this operation is Natural Language Generation (NLG), and the corresponding task is called data-to-text generation.

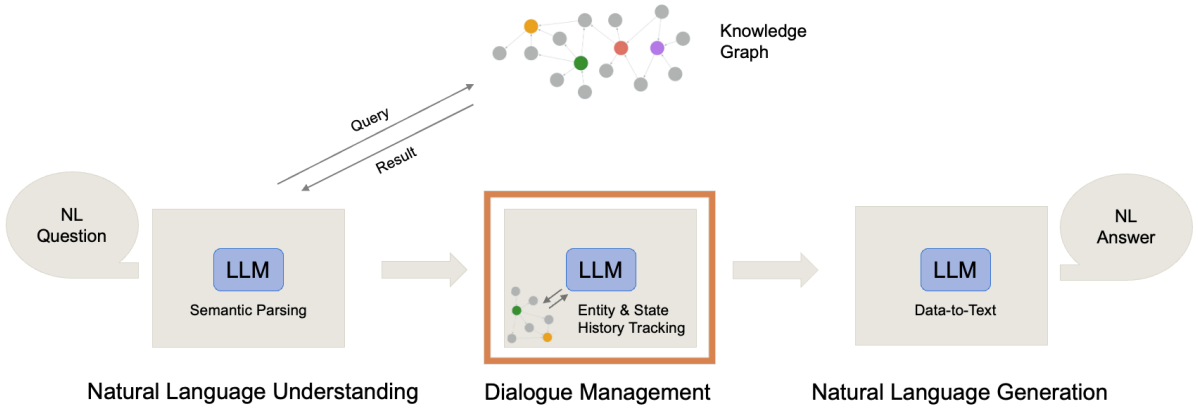


Figure 1.4: This figure shows the scope of the thesis. The out-of-scope component is highlighted in orange

1.3 Research Challenges & Research Questions

Challenge 1: We need to investigate approaches for harnessing LLMs in semantic parsing and text generation

RQ1: Which previous studies have investigated using LLMs for the tasks of semantic parsing and text generation? Semantic parsing and text generation are no new research fields. Consequently, with research question 1, we want to examine existing methodologies that address the stated tasks. This process is crucial since it allows us to discern effective approaches from those that require improvement. Furthermore, it enables us to evaluate our system against the performance of existing solutions.

Challenge 2: We need to find suitable LLMs and prompts for the two considered tasks

RQ2: What selection of Large Language Models and Prompting techniques are suitable for a comparative analysis of the considered tasks? Our work utilizes existing pre-trained language models since training an LLM from scratch takes tremendous time and resources. As a multitude of LLMs are available, it is impossible to consider them all. Thus, we need to select promising ones with different characteristics like model size, training procedure, and training data. Furthermore, the formulation and structure of a prompt significantly impact a model's performance. Therefore, we need to select some well-established prompting techniques for comparison.

RQ3: How capable are the selected Large Language Models and prompting strategies for semantic parsing and triples-to-text generation based on automatic and human evaluation? With the third

research question, we investigate the performance of chosen LLMs and prompting techniques. We compare them against each other and do an evaluation specifically for semantic parsing and triples-to-text generation. For this purpose, we use automatic metrics to get a broad understanding of their capabilities on a large number of samples. Additionally, we want to get detailed insights into their shortcomings and limitations. Thus, we conclude the analysis with a human evaluation on a subset of generations. We provide the code of our experiments and the adapters for the two fine-tuned LLaMA LLMs in our GitLab repository [1].

1.4 Outline

To address the identified research questions, this thesis is structured as follows. Chapter 2 provides the foundational knowledge required to understand this thesis's content. In Chapter 3, we give an overview of related research. The fourth chapter addresses the methodology used to obtain our results. First, it describes the procedure of our Literature Review. Then, the criteria for selecting LLMs and prompting techniques are explained. After that, it reports the utilized approaches and datasets for semantic parsing and data-to-text generation. The results of the three research questions are contained in Chapter 5. A detailed analysis and insights derived from our results are part of the sixth chapter. There, we start by reporting our key findings and conclude by reporting the limitations of our work. The last chapter provides a summary of the thesis and suggestions for further research.

[1] https://gitlab.lrz.de/ge49gah/MastersThesis_Code

2 Fundamentals

This chapter provides the foundational knowledge required to understand the rest of this thesis. It starts by defining dialogue systems and, more specifically, Conversational Search (CS). Moreover, semantic parsing and data-to-text generation, two crucial parts of the considered CS system using an external knowledge base, are explained. We continue with the Natural Language Processing (NLP) section by providing an overview of transformers, the most prominent methodology for creating LLMs, and describe the properties of the different popular language models. At the end of this chapter, we explain the evaluation metrics utilized to analyze the selected LLMs and prompting techniques, including the performance metrics used to evaluate semantic parsing and the sentence similarity measurements employed to test data-to-text generation.

2.1 Dialogue Systems

According to Deriu et al. [12], there are three main classes of Dialogue Systems (DS). These are conversational, task-based, or question-answering systems. They are controlled by the user in natural language and can be employed for different use cases like virtual assistants or Information-seeking systems. DS provide the user with a text-based, speech-based, or multi-modal interface. Thereby, dialogues are usually structured in turns.

- *Conversational agents* often try to mimic human behavior and are built for open-domain dialogues following an unstructured nature. An early example of such a system is called ELIZA [13].
- *Task-based systems* help the user to solve a specific task. They are usually developed for a particular domain and follow a predefined structure. An example is a virtual assistant in a car that helps the driver to set the desired destination in the navigation system.
- *Question Answering systems* try to answer natural language questions of the user. They commonly utilize external knowledge bases to retrieve the required information and can be capable of only single-turn or multi-turn interactions supporting follow-up questions.

2.1.1 Conversational Search

CS is defined by Vakulenkco et al. [14] as "the task of retrieving relevant information by using a conversational interface". Conversational interfaces enable humans to interact with machines in natural language, either by voice or text, and aim to make the interaction with systems more intuitive. Moreover, they are well-suited for explorative search, involving

the process of refining the search space through multiple conversational exchanges. This capability is essential in extensive, multi-domain information sources like knowledge graphs [15, 16]. Multiple inconsistent definitions are available for CS, and researchers have no clear consensus yet on which one to prefer. In general, these systems have the goal of helping users to describe their information needs and provide suitable results that fulfill those demands. In addition, it should be possible for users to refine results based on their preferences.

Radlinki et al. [17] defines five desired properties for conversational search systems as follows:

1. *User Revealment*, meaning that the system helps users to define their information needs
2. *System Revealment*, which refers to the capability of the system to show its abilities and properties of its knowledge base
3. *Set Retrieval*, describes that the system assists the user by providing sets of complementary items that fit the given information need
4. *Memory*, means that the system keeps track of the conversation history so that the user or system can refer to previously mentioned statements
5. *Mixed Initiative* denotes the ability of the system to choose an appropriate level of initiative, while users can always take it.

Although the system we address in this thesis does not fulfill all these properties, we still consider it a conversational search system. The primary reason is that in addition to conducting natural language information searches, the user can refine search results by providing follow-up questions. This helps to uncover the system's capabilities and assists in defining the user's information needs. Furthermore, the system maintains a record of the conversation history, allowing for indirect references to entities that have been mentioned previously.

2.1.2 Semantic Parsing

Translating a natural language utterance into a logical, machine-readable representation, like a database query, is called semantic parsing. Much of the existing research focuses on text-to-SQL transformation, but some work also addresses other query languages such as SPARQL or Cypher. Traditional methods for creating database queries based on natural language rely on templates that are selected and filled using rule-based approaches. Most current research utilizes PLMs instead to achieve state-of-the-art results. The natural language question is often first translated into a constrained intermediate language before the final database query is created. Semantic parsing is a crucial task, as it has the potential to be the enabler for utilizing information from databases for non-technical users.

An example of semantic parsing, more specifically text-to-SPARQL translation, is:

```
Question: "How tall is the Eiffel tower?"

Query: "SELECT ?x WHERE { wd:Q243 wdt:P2048 ?x . }"
```

The variable of the result value is thereby "?x" while Q243 refers to the Wikidata entity "Eiffel Tower" and P2048 to the property "height".

2.1.3 Data-To-Text Generation

Verbalizing structured data into natural language text is also know as data-to-text translation. Structured data can be JSON, XML, RDF triples, or tables. Hence, research consists of multiple tasks, such as generating explanations for table values or creating fluent text for a set of RDF triples. An RDF triple contains the following form:

```
:alice foaf:age 25 .
```

It contains a subject (alice), predicate (age), and object (25) as well as namespace prefixes, here defined as ":" and "foaf:". Verbalizing a set of such triples is the task we address in this thesis. For the above RDF triple, a desired generation would be:

```
Alice is 25 years old.
```

Triples-to-text translation is relevant to make graph data, usually specified as RDF triples, more understandable to regular users. As for many NLP tasks, this has traditionally been addressed using strategies comprised of rules and templates. Current state-of-the-art approaches use PLMs, combined with fine-tuning, prompt-tuning, or prefix-tuning.

2.2 Pre-Trained Language Models

PLMs are models already trained on vast amounts of data. Thus, they can be employed directly to solve general tasks or fine-tuned to enhance their performance on domain or task-specific instructions. In the following, we first give an overview of the transformer architecture widely used for PLMs. Afterward, we provide an overview of popular LLMs and their characteristics.

2.2.1 Transformers

All explanations of this section and its subsections are based on [18]. The transformer architecture was introduced by Vaswani et al. in the year 2017 with their paper titled: "Attention is all you need" [19]. This publication led to a paradigm shift, with most state-of-the-art approaches of NLP tasks currently utilizing transformer models. Using self-supervised learning, they are trained on large amounts of textual data, usually from the internet. This learning approach means that the data does not need to be labeled by humans manually. Instead, tasks are selected that use the texts themselves as ground truth. For example, one such training method for self-supervised learning is to mask a word in a text, predict it, and compare it with the original term. The models gain a statistical understanding of language through training on various texts. These are called foundational models or PLMs, as they have general language knowledge but are not yet trained for specific tasks. They can be fine-tuned on task-related training data to optimize a PLM for a particular use case. For a

more detailed explanation of the transformer architecture, we refer to the initial paper by Vaswani et al. [19]. In the following, we explain three different types of transformer models.

Auto-Encoding Transformer

If a model only uses the encoder part of the transformer in its architecture, it is called an auto-encoding transformer or encoder model. Transformers use a mechanism called attention in their architecture, which indicates words that are most important to consider when predicting a token. For encoder models, attention layers can access the entire input sentence. Thus, when a word in the sentence is masked, it can take terms to the left and right of the mask into account, which is called bi-directional attention. Auto-encoding transformers can be pre-trained using masked word prediction, which refers to predicting a corrupted (masked) word in a sentence. This architecture best suits sentence classification, named entity recognition, and extractive question answering. One of the most popular models of this family is Bert.

Auto-Regressive Transformer

A decoder model, called auto-regressive transformer, only uses the decoder part of transformer architecture. In contrast to encoder models, its attention layers can only access words before a given word. Therefore, pre-training is usually done by predicting the next token of an unfinished sentence. Since the original text is known, the predictions are compared to the ground truth to calculate the loss and update the model weights. These kinds of models are most suitable for text generation tasks, with GPT-2 being an example of a model belonging to this family.

Sequence-to-Sequence Transformer

Encoder-decoder models, also known as sequence-to-sequence transformers, use both parts of the transformer architecture, as proposed in the original paper. Thereby, the attention layers of the encoder have access to the entire input sentence. In contrast, the attention layers of the decoder can only attend to words before the given word. For pre-training such models, multiple subsequent words can be replaced with a single mask. The goal of this task is to predict the original words. Encoder-decoder models are usually used for summarizing, translation, or generative question answering. Popular models are BART and T5.

2.2.2 Popular Large Language Models

This subsection describes the fundamentals of popular LLMs. We give a brief overview of the utilized training strategies and datasets. Furthermore, we name some essential properties of the models. For detailed information, we refer to the corresponding papers.

LLaMA LLaMA refers to a group of models developed by Meta and open-sourced in February 2023 with a non-commercial license. They are available in four different sizes. To be more specific, a 7B, 13B, 33B, and 65B parameter model is available. The smallest LLM (7B) of this family utilizes an adapted transformer model architecture and has been pre-trained on publicly available and open-source data. For this training, they use multiple datasets, including:

- English CommonCrawl [20] and C4 [21]
- GitHub projects distributed under open-source licenses like MIT
- Wikipedia entries in 20 different languages
- Two Book datasets
- ArXiv for scientific data
- StackExchange for questions and answers

The 7B model was trained with around one trillion tokens, with most tokens only being used for one epoch during training. [22]

Alpaca A version of LLaMA that improves its instruction-following capabilities through supervised fine-tuning is Alpaca. This LLM is created by researchers at Stanford and is available with 7B parameters. The model is fine-tuned on 52k instruction-following examples. To create this dataset, the researchers use the self-instruct approach [23], utilizing OpenAI's text-davinci-003. As Alpaca is based on LLaMA, its license only permits non-commercial use. [24]

Vicuna The Vicuna models are a fine-tuned version of LLaMA optimized for conversations with a 7B and 13B model being available. For fine-tuning, they created a dataset based on ChatGPT conversations that users shared on the website ShareGPT.com. The dataset contains around 70k samples, and the authors do not publish it. The model allows multi-turn conversations with a maximum context length of 2048 and uses the same license as LLaMA. [25]

LLaMA - Finetuned Like Vicuna, this model is also based on LLaMA but fine-tuned for our specific tasks. Instead of full fine-tuning, we use the Low-Rank Adaptation of Large Language Models (LoRA) [26] approach. This technique significantly reduces trainable parameters and GPU memory requirements while performing similar to regular fine-tuning. We use the scripts from the git-cloner GitHub repository [27] based on the Vicuna training code [28] but optimized for low-resource GPUs. The training datasets and the exact command, including hyperparameters, are part of our GitLab repository [1]. The resulting models support a context length of 512 tokens and have the same license as LLaMA.

[1] https://gitlab.lrz.de/ge49gah/MastersThesis_Code

Text-Davinci-003 In contrast to the previously mentioned models, Text-Davinci-003 is a closed-source model created by OpenAI. It has a context length of 4k tokens. In one of their research papers, the model is referred to as GPT-3 175B [3], but OpenAI states on their website that the size of the model available through the API might differ to some extent [29]. Text-davinci-003 was trained using RLHF [30]. More specifically, Proximal Policy Optimization (PPO) [31] was used. The training procedure consists of the following steps.

1. A dataset is created, containing human written responses to a set of prompts
2. The model is fine-tuned using supervised learning on these samples and on model outputs that are ranked with the highest quality score by labelers
3. Another dataset is created. This time, humans compare model outputs and label their preferences
4. A Reward Model (RM) model is trained to utilize the second dataset. It predicts which model response labelers would favor
5. The fine-tuned model is trained further using the RM as a reward function. The goal is to maximize reward utilizing the PPO algorithm

More details about this procedure can be found in the paper "Training language models to follow instructions with human feedback" [32].

GPT-3.5-Turbo Similar to Text-Davinci-003, GPT-3.5-Turbo is a closed-source model by OpenAI. Therefore, limited information is available as they do not disclose information like the number of model parameters or the datasets used for training. Although not officially confirmed, GPT-3.5-Trubo is expected to have approximately a size of 150B parameters. Furthermore, the supported context length of the standard model is 4k tokens [33]. The model is an improvement of text-davinci-003 but enhanced for the chat use-case [29].

2.3 Evaluation Metrics

In this section, we look in detail at the metrics used for automatic evaluation. First, we describe metrics that measure how closely related the result set is compared to the desired solution. In the thesis, use them to analyze the performance of semantic parsing. Afterward, an explanation for measurements is given, comparing generated sentences with desired lexicalizations. We refer to those as sentence similarity measurements. They are used to analyze the approaches for triples-to-text generation.

2.3.1 Performance Metrics

The basis for the following explanations is [34]. If there is a desired set (gold standard), we can classify each element of the actual result into four categories.

1. True Positive (TP) are entities that are part of both the gold standard and the received result
2. True Negative (TN) are the inverse. Hence, elements that belong neither to the gold standard nor to the received result
3. False Positive (FP) are entities contained in the actual but not in the desired result
4. False Negative (FN) are missing items. Thus, they are not part of the result, although they belong to the gold standard

These four groups are used to calculate the following measurements.

Precision This metric describes how many correct results are returned out of all results. If only relevant elements exist in the solution, this measure is one. Important to note is that it does not give any information about missing but desired entities. Hence, if the gold standard contains 500 entities, but only one (correct) entity is contained in the result, precision is still optimal with a value of one. $Precision = \frac{TP}{TP + FP}$

Recall The measure describing how many of the desired results are contained in the actual solution is called recall. If all elements of the gold standard are contained in the result, this measure is one. Thus, it is also one if only one item in the gold standard exists but 500, including the desired one, in the actual result. $Recall = \frac{TP}{TP + FN}$

F1-Score The f1-score addresses the limitations of recall and precision by combining both metrics into a single score. It is one if recall and precision are one. Thus, the desired solution needs to match exactly. $F_1\text{-score} = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = \frac{TP}{TP + \frac{FN+FP}{2}}$

Accuracy We use accuracy to measure the performance of binary classification. More specifically, if there is always a single (integer or boolean) result, the solution is either correct or incorrect. Hence, this metric denotes the number of accurate predictions out of all predictions. $Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$

2.3.2 Sentence Similarity Metrics

The following metrics are used to compare generated sentences with their expected lexicalizations. They are commonly used to measure the quality of translation systems, transforming sentences from a source into a target language. We utilize these metrics to estimate the performance of different systems in translating triples into natural language text.

Bilingual Evaluation Understudy (BLEU) One of the most popular metrics for comparing the similarity of sentences is called Bilingual Evaluation Understudy (BLEU). It is calculated by first denoting the Brevity Penalty (BP). C is the candidate translation length, and r is the length of the reference corpus.

$$BP \equiv \begin{cases} 1 & \text{if } c > r\,, \\ e^{(1-r)/c} & \text{if } c \leq r\,. \end{cases}$$

Then, BP is used to compute BLEU.

$$BLEU \equiv BP \times \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

$$\text{With } N = 4,\ w_n = \frac{1}{n},\ \text{and } p_n = \frac{\#correct\_predicted\_n\text{-grams}}{\#total\_predicted\_n\text{-grams}}.$$

The score is between zero and one, with one being the optimal outcome.

One of the advantages of BLEU is the possibility to consider multiple reference translations. Furthermore, it is a commonly used metric, ensuring comparability of the results with other research. As BLEU utilizes n-grams for calculating the score, it does not depend on embeddings or models that must be trained with large amounts of text. Thus, it is also applicable for low-resource language. An additional upside is the fast computation of the score [35].

Nevertheless, BLEU also comes with some downsides. It does, for example, only consider exact word matches as correct. Hence, stems and synonyms are counted as different words, and it does not take their meaning into account. Furthermore, it does not consider the order of words as it only counts correctly predicted n-grams [36].

Metric for Evaluation of Translation with Explicit ORdering (METEOR) To calculate the Metric for Evaluation of Translation with Explicit ORdering (METEOR) score, Precision (P) and Recall (R) need to be computed for the unigrams first. This calculation works analogous to the description in subsection 2.3.1.

$$P = \frac{\#correct\_predicted\_unigrams}{\#total\_predicted\_unigrams}$$

$$R = \frac{\#correct\_predicted\_unigrams}{\#total\_reference\_unigrams}$$

Using P and R, Fmean can be calculated:

$$Fmean = \frac{10PR}{R + 9P}$$

Furthermore, a penalty term is created based on the alignment to the reference sentence. There is only one chunk in the optimal case, with the reference string matching the generation exactly. In the worst case, there is no overlap, and each unigram is a separate chunk.

$$Penalty = 0.5 \times (\frac{\#chunks}{\#unigrams\_matched})^3$$

Using Fmean and the penalty term, the meteor score can be computed.

$$METEOR = Fmean \times (1 - Penalty)$$

An upside of METEOR is that it takes stems and synonyms into account. It also considers the word order through alignment. Additionally, it is widely used, which enables comparability of research results. Furthermore, this score has fast computation times [37].

METEOR has limitations as it requires external resources like stemmer and synonym lexicon. Hence, only a limited number of languages are supported. In addition, it might penalize sentences that have the same meaning but a different surface form compared to the reference [36].

Translation Edit Rate (TER) A measure that counts the number of operations needed to reach reference sentences based on a provided text is called Translation Edit Rate (TER). These edit operations are insertion, deletion, substitution of single words, and shifts of word sequences. TER is thereby calculated as follows:

$$TER = \frac{\#of\_edits}{average\_\#of\_reference\_words}$$

An advantage of this metric is that it is simple to understand, as no complex mathematical equations are involved.

A downside is that it does not take semantics into account since it does not consider synonyms and stems of words [38].

Bert To calculate the Bert-score, a sequence of vectors for reference sentence $x =< x_1, ..., x_k >$ and candidate sentence $\hat{x} =< \hat{x_1}, ..., \hat{x_l} >$, generated by the embedding model, are required. P and R are computed, matching each token of a sentence to the token with the highest cosine similarity of the other sentence. Thus, the similarity score is maximized.

$$R_{BERT} = \frac{1}{|x|} \sum_{x_i \in x} \max_{\hat{x}_j \in \hat{x}} x_i^T \hat{x}_j$$

$$P_{BERT} = \frac{1}{|\hat{x}|} \sum_{\hat{x}_j \in \hat{x}} \max_{x_i \in x} x_i^T \hat{x}_j$$

Based on P_{BERT} and R_{BERT} the f1 measure is calculated.

$$F_{BERT} = 2\frac{P_{BERT} \times R_{BERT}}{P_{BERT} + R_{BERT}}$$

Optionally, the Bert score is available with importance weighting by utilizing inverse document frequency (idf), but we do not use it in this thesis.

A major advantage of the Bert score is that it considers the semantics of a sentence by utilizing embeddings and not just its words. Thus, it takes paraphrases into account. The Bert score is applicable for most texts since it uses BERT embeddings available in over 100

languages. Furthermore, it penalizes different word ordering between candidate and reference sentences if this changes the meaning.

A downside of the Bert-score is slower computation compared to other metrics like BLEU, especially if no GPU is available [36].

3 Related Work

This chapter gives a broad overview of existing research for semantic parsing and data-to-text generation. We first summarize how these tasks were addressed traditionally. Afterward, we look at the evolution of such systems over time towards modern procedures.

Semantic Parsing Semantic parsing aims to map utterances written in natural language to logical forms. Use cases range from generating database queries to understanding and executing commands or improving the capabilities of conversational agents. For several decades, there has been continuous research interest in this field [39]. Early approaches relied on handcrafted rules based on syntax, linguistic features, and pattern matching [40, 41]. Thus, these systems were inherently inflexible and domain-specific. Then, statistical techniques that train models on desirable input-output mappings became more prominent. These approaches learn, for example, a lexicon that pairs phrases to their corresponding logical representations from data. Research papers addressing such statistical methodologies are [42, 43, 44]. With the rising popularity of deep learning architectures leading to significant advancements in many fields of NLP, semantic parsing has been modeled as a machine translation problem. For this end-to-end approach, a neural network is trained on large amounts of data to convert a natural language input directly into the target meaning representation. Currently, many systems utilize sequence-to-sequence neural networks based on the transformer architecture that became popular with the seminal paper "Attention is all you need" by Vaswani et al. [19]. Since the resulting models develop a statistical understanding of language based on the training examples, they have a high degree of flexibility and generalize across several domains. Thus, the need for handcrafted lexicons, rules, or templates is obviated [45, 46, 47]. Hybrid techniques, combining traditional components with neural models, exist as well. These approaches reduce the decoding complexity of a neural network by imposing constraints through grammars and pre-established structures [48, 49].

Most existing literature about semantic parsing focuses on generating a meaning representation for a single user utterance in isolation. A growing body of recent research considers additional information, like the context of a question, for creating the logical forms [50]. One such field is Conversational Question Answering (CQA), which is closely related to CS. These approaches incorporate previous questions and answers, commonly called the conversation history, to provide more contextually relevant translations through improved question disambiguation and handling of indirectly referenced entities. [51] is an example of such a system. It takes the interaction history during generation into account.

Datasets suitable for assessing CQA systems are currently scarce. Given that many existing benchmarks are specialized to particular domains or have constrained scale, we opt for the recently introduced SPICE dataset [52]. It extends [53] by providing SPARQL queries for the

corresponding natural language questions. In addition, the authors of SPICE develop and provide two strong baseline models. The first approach is based on [54] and generates the entire query in a single step, utilizing a sequence-to-sequence model. The second baseline creates SPARQL templates that are subsequently populated using classifiers [55].

Data-To-Text Generation Early approaches have utilized rule-based pipelines, consisting of multiple components with distinct tasks. The system of [56], for example, comprises four sequential modules. The first creates facts based on provided data, which are subsequently transformed into sentences in the second component. After ordering these messages, the last step creates the resulting output of the pipeline. For these tasks, phrase lexicons and domain-specific grammars are used, which limits the flexibility and generalizability of the system significantly. [57] gives an overview of such traditional NLG techniques. Related to these traditional pipeline-based approaches is research using domain-specific ontologies and rules to generate natural language text from data [58]. PLMs lead to advancements in many NLP areas, including NLG and more specifically data-to-text generation [59]. One limitation of neural approaches is that lots of data is required to train or fine-tune models on specific tasks, making the development of such systems unsuitable for low-resource settings. To address this issue, multiple researchers investigate zero-shot or few-shot learning for data-to-text translation [60, 61]. Similar to semantic parsing, some hybrid approaches exist, combining the traditional pipeline architecture with neural models [62, 61]. [63], for instance, utilizes templates to transform triples into sentences that are subsequently combined by a PLM into a coherent paragraph.

Besides automatically creating texts based on data, another challenge lies in evaluating these generations. Dou et al. [8] address the human evaluation of machine text by creating the framework SCARECROW for analyzing common error categories. Their work aims to make the framework usable for people without NLP expertise and to standardize the human evaluation of NLG tasks. They define ten error types, including "bad math", "commonsense" issues, and "self-contradiction". In addition, they evaluate LLMs of different sizes on these categories against human written texts. For details, we refer to their paper [8].

In this thesis, we do a qualitative analysis of triples-to-text generations by investigating common error types with the help of human annotators. We do not use the entire SCARECROW framework, but their work heavily influences the creation of our error categories. We omit some of their error types, such as "bad math", because they are redundant for our dataset. In the end, we use *off-prompt* and *redundant* in addition to *inaccurate*, which aggregates multiple SCARECROW categories like *grammar*, *self-contradiction*, and *commonsense*. Furthermore, we introduce the two new error types *mistranslated* and *unlexicalized* that are specific to our task. Thereby, *mistranslated* refers to cases where the created text is in a different language than intended. For example, if the set of triples contains some Spanish names, confusing the LLM to verbalize the triples in Spanish instead of English. Since some triples contain terms not used in the same form in human texts like "public_company", some models fail to transform them into more natural phrases ("public company"). Thus, this is an instance of the *unlexicalized* error. A detailed explanation of all five issue types is shown in Table 6.4.

4 Methodology

4.1 Research Procedure

The general research procedure of this thesis proceeds as follows. First, we review literature to find existing research for semantic parsing and text generation leveraging LLMs as they are crucial components comprising conversational search systems. Afterward, we select 4 LLMs of different sizes, being created using diverse training procedures. With this collection of models, we want to test the influence of LLM, possessing a wide range of characteristics, on downstream tasks. Furthermore, we choose two well-established prompting techniques that we evaluate with each of the selected models. Since the research addressing LLMs and prompt engineering is evolving quickly, we want to pick well-established techniques that will likely generalize to other models. After that, we look in detail at the two relevant tasks, i.e., semantic parsing and data-to-text generation. Starting with semantic parsing, we select a dataset suitable for a comparative evaluation. Then, we create prompt templates for both chosen prompting techniques. These templates are filled with data at run-time. This is followed by performing inference using the test set to generate the evaluation data. Once the predictions are finished, we post-process the results in a rule-based manner to prevent common deviations from the gold standard. Afterwards, we run the evaluation scripts to generate the metrics used to assess performance. The research procedure utilized for data-to-text generation is analogous to semantic parsing.

4.2 Literature Review

This section discusses how we approach answering the first research question. Hence, we address finding previous research about utilizing LLMs for semantic parsing and data-to-text generation. Therefore, we do a systematic literature review using scientific databases to find academic papers. First, we develop a search string for each of the two topics. Afterward, we use these queries for three academic databases, namely, ACM Digital Library, IEEE Xplore, and Scopus, filtering studies by title, keywords, and abstract to find matching results. Furthermore, we do forward and backward snowballing based on the preliminary results. Forward snowballing refers to looking at work that cites relevant papers, while backward snowballing means considering papers cited by relevant articles [64]. We define inclusion and exclusion criteria to ensure that only relevant work is included. These are used for filtering the first results to receive the final set of considered academic work.

Inclusion and Exclusion Criteria We only take peer-reviewed papers into account that have been published in journals or were presented at academic conferences, to assure a high quality of the sources. Furthermore, they have to be written in English and deal with English texts or datasets since our work focuses on semantic parsing and data-to-text generation in the English language. Full access to the contents of the academic work needs to be granted to us. Otherwise, it will be excluded since it is crucial to take the entire information of considered approaches into account. Additionally, we include work only if it is published within or after 2020 as the paper "Retrieval-augmented generation for knowledge-intensive nlp tasks." of Lewis, Patrick, et al. [65] was released this year and sparked interest in research about grounding conversational agents with external knowledge bases, which is the focus of our work. Hence, the period of publication is from January 2020 to August 2023. Lastly, the studies have to be closely related to semantic parsing or data-to-text generation utilizing LLMs. This is determined by screening the title and abstract of the preliminary results.

Search Queries We iteratively develop two search strings for querying the scientific databases. Each of them utilizes Boolean operators to connect multiple keywords. The first search string focuses on semantic parsing, more specifically, generating database queries. Therefore, we include "semantic parsing", "query generation", and "query creation" using the OR operator. Moreover, we are only interested in approaches utilizing LLMs for the generation. Thus, one of the terms "large language model", "pretrained language model" or "pre-trained language model" must also be contained. The resulting query is:

```
("semantic parsing" OR "query generation" OR "query creation" ) AND ( "large
    language model" OR "pretrained language model" OR "pre-trained language model"
     )
```

The second search string addresses work about data-to-text generation. Hence, we include the terms "data-to-text" and the more specific task of interest, "triples-to-text". Again, we are interested in research employing LLMs. So we add "large language model", "pretrained language model" and "pre-trained language model". This results in the query:

```
("data-to-text" OR "triples-to-text") AND ("large language model" OR "pretrained
    language model" OR "pre-trained language model")
```

4.3 Selection of Large Language Models

The number of different models we can test is limited due to time and resource constraints. Therefore, we need to choose a small but diverse set of models to cover a broad range of characteristics and potential insights while respecting the given limitations. For this selection, we first consider the different attributes of popular models and how they vary.

One of the properties stated most often when comparing models is their size, measured in the number of parameters they possess. It influences the capacity of the model to learn and recognize patterns from data. Furthermore, it impacts the performance of tasks like

answering questions and generating text. Wei et al. show in their work that some capabilities of LLMs do emerge with increasing parameter size [4]. However, increasing the complexity of the model comes with some downsides. For example, it raises the resource requirements to run and train the model, especially regarding required GPU memory. Hence, it influences the upfront investment cost for hardware and the variable cost of electricity usage. A typical size for a small LLM of a model family is around 7 billion parameters. In contrast, one of the largest models to date is Google's switch-c. With a size of over 1.5 trillion parameters it is over 200 times this size [66]. Important to note is that not all companies publish the details of their models. Hence, only speculations about the model size are available for some models like OpenAI's GPT-4 [67]. Considering that our hardware can only run models with up to 7 billion parameters reliably, we select mainly LLMs of that size. To also include a larger model in the evaluation, we add one commercial model that is not hosted by ourselves and that we can only access through a provided API endpoint.

A second aspect that is important to us is the accessibility of the models. In general, there are LLMs that are fully open-sourced, which enables complete control of the model and its usage through hosting it on one's own servers. This enhances the reproducibility of tests and makes fine-tuning possible. In contrast, closed-sourced LLMs are usually deployed on a company's servers and only accessible through an API. Using these models for research might introduce some uncertainties since there is limited control, and it is often not known what pre- and post-processing steps are done to the input prompt and response. This makes it challenging to reason about the cause of certain observed behavior. Furthermore, the model or other parts of the request-response pipeline could change without notice, hindering test repeatability. Although there are many issues when using closed-source models for research, these are some of the most powerful models to date.

Another variation of the models is regarding their training procedure and corpus of training data. Multiple kinds of data can be utilized for training a model and influence its capabilities on different tasks. The corpus can, for example, contain factual data like Wikipedia articles, conversational data such as Reddit forum posts, or code snippets from open-source projects. Furthermore, there are various training procedures, including pre-training using next word or masked token prediction. Additionally, there are methods to optimize models for specific tasks, such as fine-tuning or aligning outputs to human preferences using RLHF. The combination of data and training procedure influences the generated output and capabilities of the models significantly. To get a general intuition about the performance of models on various tasks, we use the Open LLM Leaderboard from Hugging Face [68].

4.4 Selection of Prompting Techniques

Prompt engineering is the field of research investigating how a model's capabilities and its generated output are affected by using different prompting techniques and best practices. It is based on the observation that the formatting and formulation of a prompt significantly impact the quality of the produced output. Furthermore, prompts can be enriched with additional information to nudge the model to generate factually correct answers. This

technique is known as grounding the responses. Examples can be included in instructions to enable in-context learning. As Prompt Engineering has recently gained popularity, we use grey literature, like blog posts, in addition to academic papers, to get an overview of well-established and emerging approaches. In the following, we give a brief overview of the most prominent prompting techniques and best practices we consider when selecting the prompts we use in this thesis.

Prompting Techniques Zero-shot prompting is one of the most straightforward techniques. It refers to providing a natural language description of a task in the prompt. Wei et al. show that this capability can be improved by instruction tuning [69].
Example:

```
Translate the following sentence into German.
EN: The quick brown fox jumps over the lazy dog.
DE:
```

Few-shot prompting means adding examples of input-output pairs to the prompt to teach the model in context how to solve a task. This technique has significantly improved performance, in some instances comparable to fine-tuning approaches. When only one example is provided, it is also called one-shot prompting [3].
Example:

```
Translate the following sentence into German.

EN: Hello, how are you?
DE: Hallo, wie geht es Ihnen?

EN: The quick brown fox jumps over the lazy dog.
DE:
```

Chain-of-thought prompting tries to improve reasoning capabilities of the LLM. Instead of returning only the answer, the model is nudged to include the individual steps necessary to produce the result. This behavior can be achieved by adding examples to the prompt, demonstrating expected generations [70]. Another variation of this technique is called zero-shot chain-of-thought prompting. Instead of providing examples, the phrase: "Let's think step by step" is added to the prompt [71].
Example:

```
Q: Bob receives a gift of 10 apples from his grandmother. After sharing half of
    them with Alice, he consumes two. Unfortunately, a third of the remaining
    apples begin to rot. How many of Bob's apples are still in good condition?
A: Let's think step by step.
```

Prompting Best Practices There exist multiple best practices when creating prompting templates. Many are based on anecdotal evidence and are part of grey literature like blog posts or the documentation of LLM providers. We consider these best practices as they are

recommended by the most relevant companies in this area while keeping in mind that some are not yet scientifically evaluated.

When using few-shot prompting, a common practice is to provide a diverse set of examples. These could vary in terms of complexity or type of data. For example, when the task is to classify a text into three categories, one could provide three examples in the prompt, one for each category. This variability prevents bias of the model towards a single class and helps it to distinguish between the different cases.

Furthermore, the structure of a prompt is essential. It can be formatted, separating the individual parts using a dedicated token or new line. Additionally, keywords can help to nudge the models to solve the task correctly [72].

Models trained for chat often enable specifying three different roles in the conversation. The system role defines the general behavior or persona of the agent. The user prompt provides the input data, while the assistant returns the system's response. Furthermore, multiple conversational turns of the user and assistant can be part of the request and define a conversation history [72].

It is often reported that giving the model a command using imperative language works better than asking the model to perform the task with a question [73]. Hence, it is better to tell the model, "Classify the following text as positive and negative", instead of asking, "Is the following text positive or negative?". Additionally, the instruction should be written in a short and precise manner while telling the model what to do instead of what not to do [74].

We select two well-established prompting techniques, where we expect a generalization to many models. Furthermore, we take best practices into account when creating concrete prompt templates.

4.5 Semantic Parsing

As well-established approaches are already available for querying unstructured data, for example, by using embeddings and vector databases, we focus on knowledge bases containing structured data. Graph databases are a popular choice for storing large amounts of highly connected data since they are more flexible than SQL databases and fast even for complex queries. In this thesis, we consider RDF graphs, more specifically using the query language SPARQL, since this is one of the most common query languages for graph databases, and there are extensive and freely accessible knowledge bases like Wikidata and DBpedia available. While other graph database query languages like Cypher would be interesting, suitable and established datasets for these languages are limited.

4.5.1 Dataset

We use the dataset SPICE (Semantic Parsing for Conversational Question Answering over Knowledge Graphs) [52]. It is derived from CSQA [53] and contains user interactions with an assistant in a conversational style. Each independent conversation is thereby separated into a distinct file. The dataset provides a natural language question, its SPARQL parse, and the

corresponding answer for each conversational turn. In some instances, a clarifying question from the assistant and the corresponding response of the user is contained in addition to the SPARQL query and database result. Moreover, coreference resolution is needed to answer some questions, as they refer to entities mentioned in prior conversational turns. Hence, within one conversation, the desired response can depend on the conversation history, and some of the questions require complex SPARQL queries. In total, SPICE contains 197k conversations, with an average conversational turn length of 9.5. The test set comprises 27,797, while 152,391 are in the training set.

One of the advantages of utilizing the SPICE dataset is that it uses SPARQL queries that are compatible with Wikidata, a freely available and established knowledge graph. Furthermore, the SPICE data is conversational. Thus, it consists of multiple conversations, each containing several turns. This structure is crucial when evaluating a chatbot since it closely resembles the actual use of such a system. Other datasets like LC-Quad are less suitable for us as they do not use this conversational style, including references to previously mentioned entities [75]. Another advantage of SPICE is that it specifies a type and subcategory, called a description, for each question. It contains questions of 60 different types and categories, allowing for a detailed analysis of the results. An official evaluation script that generates metrics for these question categories is also provided. We use this script during our evaluation to ensure comparability to existing and future research. Lastly, the training set, containing over 150k conversations, is large enough for fine-tuning a LLM.

Nevertheless, the SPICE dataset also comes with some downsides. Since the dataset was created and published recently, not many other systems and their evaluations are available for comparison. This is not a significant issue as the authors of the dataset provide some baseline systems and report their performance metrics. Another disadvantage is the large test set size since we can not consider all samples given our resource constraints. This limitation must be considered when comparing the performance of our approaches with the baseline systems.

Dataset Preparation Before using the dataset, preprocessing is required as the entities, types, and relations it specifies only provide Wikidata references without their labels. As we do not want to rely on the models to resolve these entities based on their intrinsic knowledge, we explicitly add them to the dataset. If, for example, the list of entities contains a reference to Q30, we map "United States of America" to it. The lookup of these entities via wikidata.org could also be done during run-time, but this would significantly slow down execution times.

Furthermore, we create a fine-tuning dataset. Since we are limited by our available hardware, only a small part of the test set is used to keep required GPU hours below one day. Thus, we create a dataset for fine-tuning consisting of 30,000 conversations. Each contains between one and four independent conversational turns to simulate zero- and few-shot prompting. Thereby, they utilize the same system message and prompt structure compared to inference. To ensure the high quality of training data, we filter out all instances where no SPARQL query is provided for the gold standard. We also ignore clarifying questions since these are not the focus of our work.

It is not feasible for us to do inference with the entire test set as it would approximately

require $\frac{27{,}797\text{convs}\times 9.5\text{avg turns}\times 3\text{sec}}{(60\times 60\times 24)\text{sec}} \approx 9$ days to run one of the eight considered model-prompt combinations. Therefore, we only use a subset containing 1500 conversational turns. To create this subset, we first compute the distribution of the entire test set regarding the 60 question categories. Based on this distribution, we calculate the required samples for each class given the desired test set size. Afterward, we randomly select files, considering the required samples of each question category, for preprocessing. We choose entire files and can not sample conversational turns at this point, as they are not independent, and the entire file is needed for constructing a prompt that contains the conversation history. Once enough files are preprocessed to cover the distribution, we do inference for each question category. To ensure consistency across multiple test runs, we process the folders and files in an ascending manner until the required samples have been processed.

4.5.2 Prompts

In the following, we give an overview of the content each prompt needs to contain and the process we use to create the prompt templates. The resulting prompts are part of chapter 5.

Some data needs to be included explicitly in the prompt. The reason for this is twofold. First, not every piece of information required to generate a suitable response was part of the model's training. Second, data is needed to ground the model's responses, preventing hallucinations. In the case of SPICE, such information is mapping entities, relations, and types to Wikidata entities. We add this information explicitly to the dataset and include it within the prompt. This procedure could also be done dynamically in the pipeline of the conversational agent by extracting entities with existing NLP solutions and retrieving the reference of this entity directly from Wikidata. The main reason for us to add it to the dataset instead is improved execution time, which is crucial for evaluating the system with large amounts of data. Since the model should be able to resolve references to entities mentioned in previous conversational turns, a conversation history needs to be part of the prompt. Furthermore, the model does not know the desired structure of the expected SPARQL query. Thus, we need to add additional information to the prompt, describing how variables should be named and what namespace prefixes are already defined. As the resulting queries should be executable, we need to instruct the model to generate SPARQL queries, refraining from explanations and notes.

In general, constructing the prompt is an iterative process for us. It is visualized in Figure 4.1. To start, we select one of the considered models that we expect to perform best on the given task, based on the Open LLM Leaderboard from Hugging Face [68]. Then, we construct an initial prompt and use this model to perform inference on a sample with little complexity. After observing deviations of the predicted query from the gold standard, we change the prompt by adapting the formulation and

structure or adding additional instructions. We repeat this procedure until no further improvement is apparent. Once a basic prompt is developed, we do inference with each of the considered models on a limited amount of samples, in our case 20, using an unseen part of the training set. After that, we try to observe common error patterns again and improve the prompt based on the observations. This step is repeated until we can not observe further improvements and results in the final prompt.

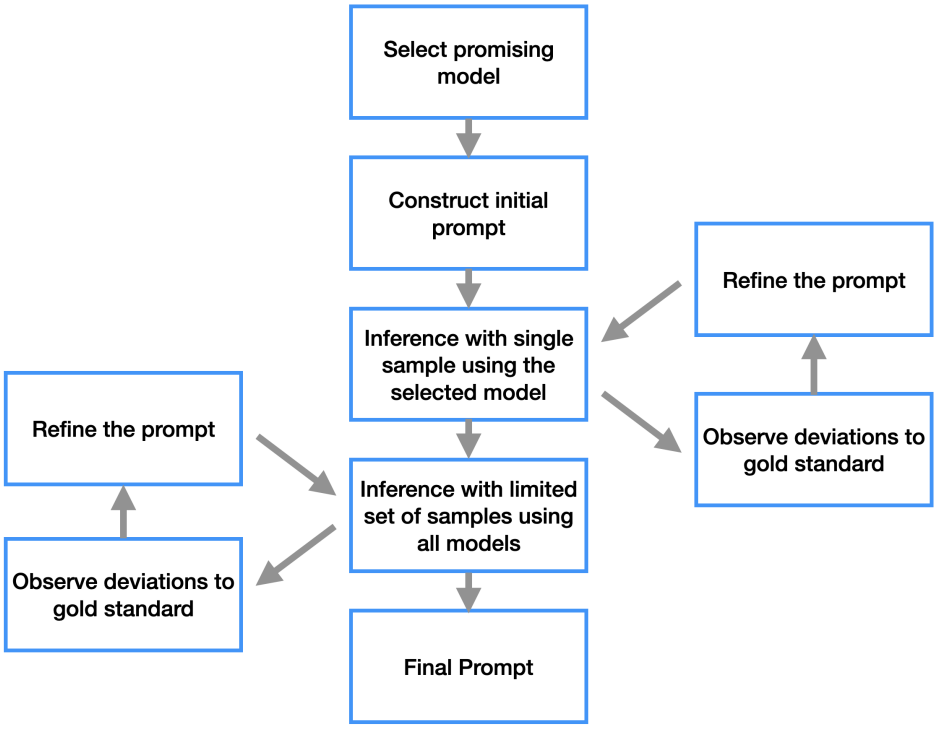


Figure 4.1: This figure displays our process of creating prompts

4.6 Data-To-Text Generation

Verbalizing knowledge, often found as structured data, into natural language text (data-to-text) is essential for conversational interfaces as they need to present this information to users in an understandable manner. Thus, the second task we address, besides semantic parsing, is data-to-text generation. As a SPARQL query, in combination with its result, can be formulated as a set of triples, we focus specifically on verbalizing triples into a natural language answer (triples-to-text). The following is an example demonstrating how our system works from start to finish.

User question:

```
How old is Alice?
```

SPARQL query:

```
SELECT ?age
WHERE {
  :alice foaf:age ?age .
}
```

Query result:

```
age: 25
```

Resulting set of RDF triples:

```
:alice foaf:age 25 .
```

Desired answer sentence:

```
Alice is 25 years old.
```

4.6.1 Dataset

For translating triples to text, the WebNLG dataset is a well-established benchmark. While there are multiple versions available, we use WebNLG+ 2020 [76] as it is newer than the original version from 2017 [77] and focuses on English, other than the most recent version of 2023, which addresses low resource languages. Two Challenges are available for the WebNLG+ 2020 dataset. Namely, mapping a set of RDF triples to text (RDF-to-text) and extracting a set of RDF triples from a provided text (text-to-RDF). Both tasks are available in English and Russian. Since we are interested in generating a natural language text based on a set of triples, we only consider the corresponding English data of this task. The dataset contains 13,211 entries in the train set, 1,667 in the dev set, and 1,779 in the test set.

Each entry of the dataset contains multiple properties. We consider "category", "modifiedtripleset", "lexicalisations", and "size" for our use-case. "Category" describes the topic of the triple set content. While the training set contains 16 topics like Food, Artist, and University, the test set includes three additional, unseen ones. These are Film, Scientist, and MusicalWork. The property "modifiedtripleset" contains the collection of triples that we use as the basis of text generation. This is an improved version of the "originaltripleset" as it uses cleaned and reformulated triples. Furthermore, it merges related sets of triples into a single set. The gold standard is part of "lexicalisations", a list of possible natural language formulations for a given set of triples. For each of the provided lexicalisations, a comment rates the quality as good, bad, or toFix. We only consider translations marked as good for evaluation.

The last property that we utilize is called "size". It denotes the number of tripels contained in the corresponding set. In the WebNLG dataset, this range is between one and seven triples.

We chose the WebNLG dataset for our work because lots of existing research and systems are available using this data. Additionally, the authors provide an official evaluation script and leader board, ensuring comparability of different approaches. Furthermore, it provides a sufficient number of training examples for fine-tuning. Another advantage is that it specifies multiple lexicalizations for most sets of triples. Thus, we can take more variety of language into account when calculating evaluation metrics. Since the topic and size for each triple set are reported, the dataset enables us to consider this information when analyzing the performance of different models and prompts.

A disadvantage of WebNLG is that although the training set is big enough for fine-tuning, having additional samples would likely improve performance further.

Dataset Preparation We construct a dataset for fine-tuning based on the 13,211 samples from the training set. For this, we use data augmentation to increase its size to 26,422 conversations, each consisting of one to four conversational turns. When creating the dataset, we randomly select each conversation's length. Afterward, we randomly choose samples from the training set for each conversational turn. Hence, each turn is independent, and the conversations simulate zero- and few-shot prompting. As the mean length of a conversation in the fine-tuning dataset is 2.5 and we create 26,422 conversations, each sample of the training set is, on average, contained five times, but in different conversational contexts.

For evaluation, we use the entire test set. Thereby, we compare the generated sentences with all corresponding lexicalizations marked as "good", omitting cases that are labeled "bad" or "toFix".

4.6.2 Prompts

The prompts suitable for this task are more straightforward than the ones for semantic parsing, as this problem is less complex. Information required to be part of the input are instructions for solving the task and the set of triples that should be translated into human-readable text. We follow the same process for developing suitable prompts as for semantic parsing. Thus, we refer to Figure 4.1 and its description in section 4.5 for details.

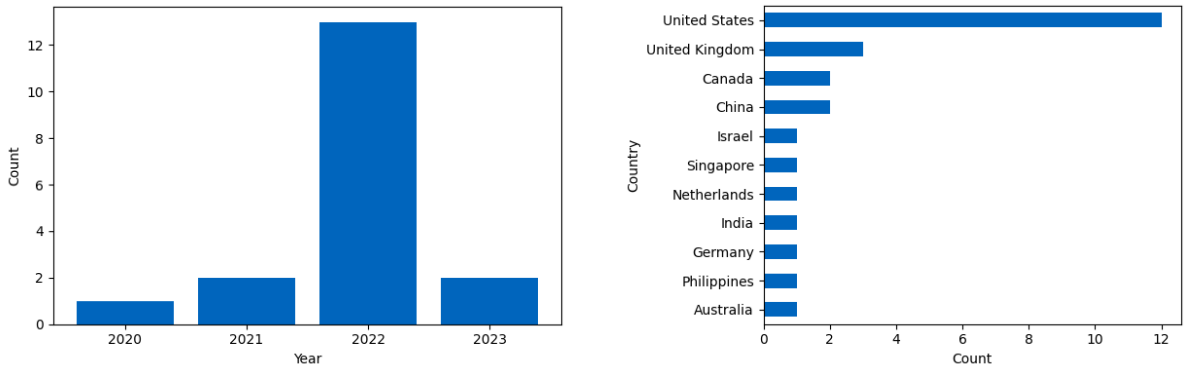
5 Results

5.1 RQ1: Literature Review

In the following, we provide the results of our literature review, aiming to give an overview of existing approaches that utilize LLMs for the tasks of semantic parsing and data-to-text generation. This overview helps situate our work within the current research field, comparing it to related and dissimilar approaches.

5.1.1 Semantic Parsing

The search string for semantic parsing, as described in section 4.2, returns 50 candidate papers. One is from ACM Digital Library and the other 49 from Scopus. The query has not led to any results for IEEE Xplore. After filtering the papers further based on inclusion and exclusion criteria, 16 results remain, with all except one from the Scopus database. We find two additional research papers relevant to the given topic based on backward snowballing.



(a) Distribution of semantic parsing papers per year

(b) Semantic parsing publications per affiliated countries

Figure 5.1: Statistics of Semantic Parsing Publications Utilizing LLMs

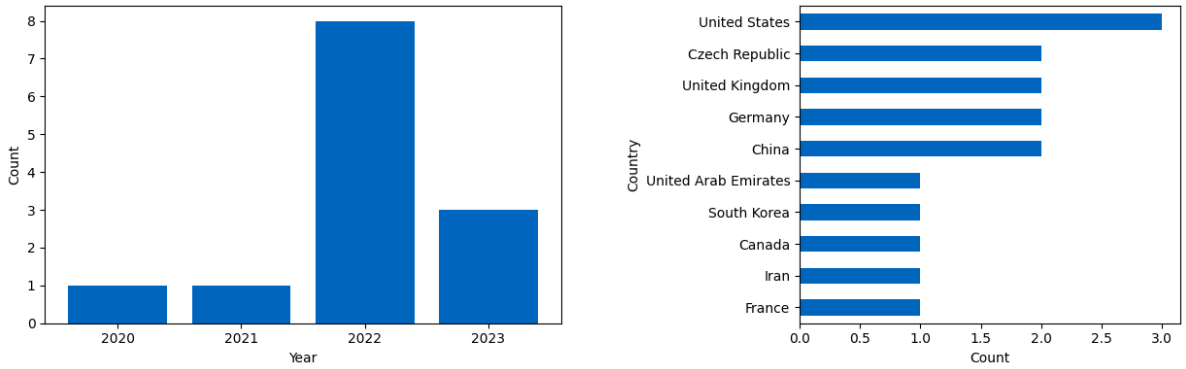
As displayed in Figure 5.1, most of the considered research was published in 2022 and is affiliated with institutions located in the United States. Important to note is that for some papers, we count multiple countries, as we take all the countries into account that correspond to institutions affiliated with the authors.

Publications LLMs are optimized to create natural language responses. Therefore, instead of translating a natural language question directly into a structured and logical representation, the research of [78] focuses on generating a controlled language first that is closely related to English (canonical utterances). Thereby, they employ explicit decoder constraints. In a subsequent step, this representation is mapped to the target representation. They test few-shot prompting with GPT-3 and fine-tune the two models, GPT-2 XL and BART, reporting promising results even when utilizing a small number of examples. The work of [79] is similar but evaluates OpenAI Codex [80] instead and reports improved performance. [81] build on the work of [78] as well. They improve the BART model by augmenting training for semantic parsing with the two additional tasks, mask infilling, and denoising. Furthermore, they increase the amount of training data through self-training and paraphrasing. A constrained decoder is also used by [49]. In contrast to the abovementioned research, they do not use intermediate representations and generate SQL directly using the T5 model. Constrained decoding should thereby ensure that only valid SQL is generated. Furthermore, they propose a method for ranking alternative SQL generations. Other work focuses on retrieving suitable examples for in-context learning based on the current request [82], [83]. Instead of trying to predict the entire formal representation of a natural language input at once, [84] decomposes the problem into smaller tasks. With the LLM answering a sequence of prompts, each corresponding to a sub-clause of the SQL query. Pointer generator networks allow the prediction of words from a vocabulary or to copy input tokens to the output. They are used by Rongali et al. [47] in combination with a transformer model to generate user intent and filled slots based on a user question. Banerjee et al. [46] use pointer generator networks to generate SPARQL queries in their experiments but find that the T5 PLM outperforms this approach. One challenge when performing text-to-SQL parsing is to generalize well to unseen databases. For this problem, one crucial task is to link the schema correctly to entities in the natural language question. [45] addresses this issue by utilizing a probing procedure to exploit relation structures contained in PLMs. Examining different training procedures for LLMs is another area of research. [85] compare in-context learning, fine-tuning, and prompt-tuning for different models, focusing on scaling curves. In addition to evaluating the influence of model size on performance, they analyze the generations for common error types. In contrast, [86] compares prompt-tuning and fine-tuning in low-resource settings for semantic parsing. They find that prompt-tuning outperforms fine-tuning when only few samples are available. Sun et al. [87] explore prefix-tuning and bias-term tuning compared to full and partial fine-tuning. They test those methods using few-shot and regular data settings. While the work we mentioned so far addresses creating structured representations, there is also research in semantic parsing focusing on other aspects of this task and not on the generation itself. [88], for example, presents a pre-training approach specifically for conversational semantic parsing. [89] check the robustness of prompt-based semantic

parses that use LLMs against adversarial attacks. There are also some comparative studies available providing an overview of existing research [90], [91], [92].

5.1.2 Data-To-Text Generation

For data-to-text generation with LLMs, the search query defined in section 4.2 delivers 27 results. Of these papers, 26 are found using Scopus, while one is from IEEE Xplore. Based on these candidate papers, we selected 13 for further investigation. One of them is from IEEE Xplore, the rest from Scopus.



(a) Distribution of Data-To-Text Papers per Year

(b) Affiliated Countries of Data-To-Text Publications

Figure 5.2: Statistics of Data-To-Text Generation Research Using LLMs

Figure 5.2 shows that, similar to research about semantic parsing with LLMs, most papers about data-to-text generation that use LLMs are published in the year 2022. Of the considered work, less has been released in 2023 than in 2022, as the end date of our literature research is 31 August 2023. Again, the majority of publications are affiliated with institutions located in the United States, but here, the gap to other countries is much smaller

Publications Translating a set of triples into a coherent text is a challenging task. Therefore, [63] and [62] use templates to transform each triple separately into natural language. Then, these sentences are fused into more cohesive paragraphs using a PLM. Instead of utilizing templates, [61] create a short sentence for each triple with GPT-3 before merging them with the T5 model. Jolly et al. [93] use templates in the second stage of their process. First, they generate text based on the triple set with a fine-tuned model. Afterward, they increase semantic coverage by inserting missing information slots with templates into the created text. Using these results, they further fine-tune their LLM. Improving semantic coverage is something [94] is working on as well. They try to prevent the generation of unfinished semantic concepts or duplications by introducing a memory module that tracks the history

of already-used information. Others try to predict a plan before generating the text [95]. Gong et al. [96] also use a planning-then-writing pipeline. In addition, they employ prefix tuning and refine generations with a diffusion model. [97] introduce an approach that combines prefix tuning with conditional input-dependent information. For the WebNLG dataset, for example, they improve performance using the known categories as conditional input. In their work, [98] develop a selective token generator using reinforcement learning. This approach chooses a generator for each output token, with the choice being a PLM or a task-specific adapter. Similar to semantic parsing, research compares few-shot prompting, prompt tuning, and fine-tuning [60]. Furthermore, there is work on sample selection strategies when labeling and fine-tuning on a small number of instances [99]. In contrast, [100] try improving performance by training a model on a large heterogeneous dataset of multiple sources. In the field of data-to-text generation, there is also research that does not deal with creating text from triples. [101], for example, try to create explanations for table values. Hence, they concentrate on a different task called table-to-text generation.

5.2 RQ2: Selection of Large Language Models & Prompting Techniques

We choose four different models for the comparative analysis of triples-to-text generation and semantic parsing. These are LLaMA, Vicuna, GPT-3.5-Turbo, and a fine-tuned LLaMA model tailored to our specific tasks, denoted as LoRA, named after the fine-tuning methodology employed. Table 5.1 shows the selected models and gives an overview of some essential properties. Access describes how the models can be accessed and used. Parameters are the model's size regarding its parameters in billions and training denotes the training procedure applied for creating the models.

LLaMA The LLaMA model was developed and open-sourced by Meta. It is important to note that due to its noncommercial license, LLaMA is unsuitable for enterprise environments, and access is mainly granted to academic researchers. Since we have access to the model weights, we run it on our server, which gives us complete control of the test and evaluation environment. We only consider the 7B version for the experiments, which was trained with one trillion tokens from publicly available datasets. LLaMA is a purely pre-trained model and, hence not optimized for our tasks [22].

Vicuna Like LLaMA, we only consider the Vicuna model with 7B parameters. It is a fine-tuned version of LLaMA utilizing approximately 70K conversations from ShareGPT.com to optimize it for the chatbot use case. As it is based on LLaMA, Vicuna also uses a noncommercial license [25].

LoRA We create two models, one optimized for semantic parsing and one for triples-to-text generation, by fine-tuning LLaMA. We refer to these two models as LoRA since we utilize the LoRA approach to create adapters instead of fine-tuning the entire model [26]. We merge the resulting LoRA adapters with the LLaMA 7B weights to improve inference times. Since the resulting LLMs are based on LLaMA, they are not applicable for commercial use.

GPT-3.5-Turbo With approximately 150B parameters, GPT-3.5-Turbo is the largest model we consider for the evaluation. As this is a proprietary model of OpenAI, only limited information is available regarding training methodology and data or the exact size of the model. We use the version gpt-3.5-turbo-0613 for our experiments, which has been available since June 2023 [33] .

Table 5.1: This table shows an overview of all models we include in the evaluation. It describes the access, number of parameters in billions, and training procedure for each LLM.

	LLaMA	**Vicuna**	**LoRA**	**GPT-3.5-Turbo**
Access	Open Source *	Open Source *	Open Source *	Closed Source
Parameters	7B	7B	7B	~150B **
Training	Pre-Trained	Fine-Tuned (Chat)	Fine-Tuned (Our Tasks)	Reinforcement Learning from Human Feedback (RLHF)

* Noncommercial license
** Not officially acknowledged

5.2.1 Semantic Parsing

We chose to use one zero-shot and one few-shot prompt. For both, we utilize the chat prompting structure, consisting of the roles "system", "user", and "assistant". We specify the same system message for each prompting strategy using the "system" role. For the few-shot prompt, we define the examples as previous conversational turns by providing mocked intents of the user and utterances of the assistant as part of the prompting template. The conversation history of the user intent is created by including the last three conversational turns in the prompt if these exist. We append "USER" and "SYSTEM" tags for this conversation history. An example of such a history can be seen in the few-shot prompt template.

In the following, we show the two resulting prompt templates. Roles are color-coded using blue for "system", black for "user", and green for "assistant". Parts of the prompt marked with "<>" are variables that are inserted at runtime based on the current sample.

Zero-Shot Prompt

Generate a SPARQL query that answers the given 'Input question:'. Use 'Entities:', 'Relations:' and 'Types:' specified in the prompt to generate the query. The SPARQL query should be compatible with the Wikidata knowledge graph. Prefixes like 'wdt' and 'wd' have already been defined. No language tag is required. Use '?x' as variable name in the SPARQL query. Remember to provide only a SPARQL query in the response without any notes, comments, or explanations.

<conversation_history>

Input question: <utterance>
Entities: <entities>
Relations: <relations>
Types: <types>

Few-Shot Prompt For the few-shot prompt, we use three examples of the task. The utilized cases require different SPARQL constructs, including ASK, SELECT, WHERE, COUNT, DISTINCT, and UNION. Furthermore, the last sample demonstrates the use of a conversation history to resolve an entity referenced from a previous conversational turn.

Generate a SPARQL query that answers the given 'Input question:'. Use 'Entities:', 'Relations:' and 'Types:' specified in the prompt to generate the query. The SPARQL query should be compatible with the Wikidata knowledge graph. Prefixes like 'wdt' and 'wd' have already been defined. No language tag is required. Use '?x' as variable name in the SPARQL query. Remember to provide only a SPARQL query in the response without any notes, comments, or explanations.

Input question: Is New York City the place of death of Cirilo Villaverde ?
Entities: {'Q727043': 'Cirilo Villaverde', 'Q60': 'New York City'}
Relations: {'P20': 'place of death'}
Types: {'Q56061': 'administrative territorial entity'}

SPARQL query: ASK { wd:Q727043 wdt:P20 wd:Q60 . }

Input question: How many works of art express Michael Jordan or pain ?
Entities: {'Q41421': 'Michael Jordan', 'Q81938': 'pain'}
Relations: {'P180': 'depicts'}
Types: {'Q838948': 'work of art'}

SPARQL query: SELECT (COUNT(DISTINCT ?x) AS ?count) WHERE { { ?x wdt:P180 wd:Q41421 . ?x wdt:P31 wd:Q838948 . } UNION { ?x wdt:P180 wd:Q81938 . ?x wdt:P31 wd:Q838948 . } }

Conversation history:
USER: Which administrative territory is the native country of Cirilo Villaverde ?
SYSTEM: {'Q241': 'Cuba'}

Input question: Which is the national anthem of that administrative territory ?
Entities: {'Q241': 'Cuba'}
Relations: {'P85': 'anthem'}
Types: {'Q484692': 'hymn'}

SPARQL query: SELECT ?x WHERE { wd:Q241 wdt:P85 ?x . ?x wdt:P31 wd:Q484692 . }

<conversation_history>

Input question: <utterance>
Entities: <entities>
Relations: <relations>
Types: <types>

5.2.2 Data-To-Text Generation

Similar to semantic parsing, for the triples-to-text task of the WebNLG dataset, we also decided to use one zero-shot and one few-shot prompt. Again we utilize the chat structure with the three roles "system", "user" and "assistant", specifying the same system message for both prompting techniques.

For the zero- and few-shot prompt templates, we display the "system" role in blue, the "user" role in black, and "assistant" in green. "<>" marks parts of the prompt that are substituted at runtime based on the input.

Zero-Shot Prompt

Generate a concise text for the given set of triples. Ensure that the generated output only includes the provided information from the triples.

Input triples: <triples>

Few-Shot Prompt The examples of the few-shot prompt are taken from the three categories: Athlete, Politician, and Building of the training set. Furthermore, we increase the complexity of the samples by starting with one triple, increasing it to three in the second sample, and using five triples for the last set.

Generate a concise text for the given set of triples. Ensure that the generated output only includes the provided information from the triples.

Input triples: [{'object': 'Mike_Mularkey','property': 'coach','subject': 'Tennessee_Titans'}]

Output text: Mike Mularkey is the coach of the Tennessee Titans.

[{'object': 'Albert_E._Austin', 'property': 'successor', 'subject': 'Alfred_N._Phillips'}, {'object': 'Connecticut', 'property': 'birthPlace', 'subject': 'Alfred_N._Phillips'}, {'object': 'United_States_House_of_Representatives', 'proper ty': 'office', 'subject': 'Alfred_N._Phillips'}]

Output text: Albert E. Austin succeeded Alfred N. Phillips who was born in Connecticut and worked at the United States House of Representatives.

Input triples: [{'object': 'College_of_William_&_Mary', 'property': 'owner', 'subject': 'Alan_B._Miller_Hall'}, {'object': '2009-06-01', 'property': 'completionDate', 'subject': 'Alan_B._Miller_Hall'}, {'object': '101 Ukrop Way', 'property': 'address', 'subject': 'Alan_B._Miller_Hall'}, {'object': 'Williamsburg,_Virginia', 'property': 'location', 'subject': 'Alan_B._Miller_Hall'}, {'object': 'Robert_A._M._Stern', 'property': 'architect', 'subject': 'Alan_B._Miller_Hall'}]

Output text: The Alan B Miller Hall's location is 101 Ukrop Way, Williamsburg, Virginia. It was designed by Robert A.M. Stern and was completed on 1 June 2009. Its owner is the College of William and Mary.

Input triples: <triples>

5.3 RQ3: Evaluation

5.3.1 Semantic Parsing

The results we achieved for semantic parsing on the SPICE dataset are shown in Table 5.2. As described in section 4.5, all metrics are computed on 1,500 conversational turns of the test set, as it is not feasible for us to use the entire test set due to resource

and time constraints. We list the results for each of the ten question types separately. While we report the f1-Score for categories where the result is a set of entities, accuracy is used to measure the performance of results that return an integer or boolean value. Furthermore, we report the Exact Match (EM) of generated queries compared to the gold standard. We utilize rule-based post-processing on all the results to ensure consistent use of white spaces and remove "SPARQL query:" from the beginning of the output. "LoRA-7B zero-shot-512" differs from the other model-prompt combinations, using a max token length of 512 instead of 128 tokens. In this way, we want to test if the model-prompt combination delivering the highest overall performance can still be improved by alleviating the max token restriction. The best values for each question type and metric are marked in bold. The result space for each of the reported measurements ranges from zero to 1, with one being the optimal output.

Table 5.2: This table shows the results for semantic parsing on a subset of the SPICE test set for all LLM and prompt combinations. All outputs are post-processed in a rule-based manner. Depending on the question type, we report f1-score or accuracy

Question Type	Model-Prompt	F1-Score	EM	Accuracy
	LLaMA-7B zero-shot	0.000	0.000	
	LLaMA-7B few-shot	0.352	0.724	
	Vicuna-7B zero-shot	0.003	0.000	
	Vicuna-7B few-shot	0.127	0.230	
Simple Question (Direct)	GPT-3.5-Turbo zero-shot	0.324	0.337	
	GPT-3.5-Turbo few-shot	0.804	0.741	
	LoRA-7B zero-shot	0.867	**0.970**	
	LoRA-7B zero-shot-512	0.867	**0.970**	
	LoRA-7B few-shot	**0.963**	0.917	
	LLaMA-7B zero-shot	0.000	0.000	
	LLaMA-7B few-shot	0.350	0.568	
	Vicuna-7B zero-shot	0.000	0.000	
	Vicuna-7B few-shot	0.189	0.321	
Simple Question (Coreferenced)	GPT-3.5-Turbo zero-shot	0.491	0.234	
	GPT-3.5-Turbo few-shot	0.636	0.623	
	LoRA-7B zero-shot	0.882	0.867	
	LoRA-7B zero-shot-512	**0.892**	**0.873**	
	LoRA-7B few-shot	0.844	0.786	
	LLaMA-7B zero-shot	0.000	0.000	
	LLaMA-7B few-shot	0.000	0.000	
	Vicuna-7B zero-shot	0.000	0.000	
	Vicuna-7B few-shot	**0.012**	0.000	
Clarification	GPT-3.5-Turbo zero-shot	0.000	0.000	
	GPT-3.5-Turbo few-shot	0.000	0.000	
	LoRA-7B zero-shot	0.000	0.000	
	LoRA-7B zero-shot-512	0.000	0.000	
	LoRA-7B few-shot	0.000	0.000	

Table 5.2: This table shows the results for semantic parsing on a subset of the SPICE test set for all LLM and prompt combinations. All outputs are post-processed in a rule-based manner. Depending on the question type, we report f1-score or accuracy

Question Type	Model-Prompt	F1-Score	EM	Accuracy
	LLaMA-7B zero-shot	0.000	0.000	
	LLaMA-7B few-shot	0.001	0.000	
	Vicuna-7B zero-shot	0.000	0.000	
	Vicuna-7B few-shot	0.072	0.000	
Comparative Reasoning (All)	GPT-3.5-Turbo zero-shot	0.015	0.000	
	GPT-3.5-Turbo few-shot	0.006	0.000	
	LoRA-7B zero-shot	0.000	0.000	
	LoRA-7B zero-shot-512	**0.315**	**0.114**	
	LoRA-7B few-shot	0.001	0.000	
	LLaMA-7B zero-shot		0.000	0.000
	LLaMA-7B few-shot		0.000	0.000
	Vicuna-7B zero-shot		0.000	0.000
	Vicuna-7B few-shot		0.000	0.000
Comparative Reasoning (Count) (All)	GPT-3.5-Turbo zero-shot		0.000	0.000
	GPT-3.5-Turbo few-shot		0.000	0.000
	LoRA-7B zero-shot		0.000	0.000
	LoRA-7B zero-shot-512		**0.165**	**0.165**
	LoRA-7B few-shot		0.000	0.000
	LLaMA-7B zero-shot		0.000	0.000
	LLaMA-7B few-shot		0.0227	0.152
	Vicuna-7B zero-shot		0.000	0.000
	Vicuna-7B few-shot		0.008	0.091
Quantitative Reasoning (Count) (All)	GPT-3.5-Turbo zero-shot		0.008	0.197
	GPT-3.5-Turbo few-shot		0.212	0.485
	LoRA-7B zero-shot		**0.561**	**0.591**
	LoRA-7B zero-shot-512		**0.561**	**0.591**
	LoRA-7B few-shot		0.417	0.492
	LLaMA-7B zero-shot	0.000	0.000	
	LLaMA-7B few-shot	0.109	0.000	
	Vicuna-7B zero-shot	0.000	0.000	
	Vicuna-7B few-shot	0.001	0.000	
Logical Reasoning (All)	GPT-3.5-Turbo zero-shot	0.631	0.000	
	GPT-3.5-Turbo few-shot	**0.912**	0.246	
	LoRA-7B zero-shot	0.900	**0.926**	
	LoRA-7B zero-shot-512	0.900	**0.926**	
	LoRA-7B few-shot	0.810	0.779	
	LLaMA-7B zero-shot		0.000	0.000
	LLaMA-7B few-shot		0.000	0.000
	Vicuna-7B zero-shot		0.000	0.000
	Vicuna-7B few-shot		0.162	0.365
Verification (Boolean) (All)	GPT-3.5-Turbo zero-shot		0.000	0.000
	GPT-3.5-Turbo few-shot		0.480	0.926
	LoRA-7B zero-shot		**0.851**	**0.939**

Table 5.2: This table shows the results for semantic parsing on a subset of the SPICE test set for all LLM and prompt combinations. All outputs are post-processed in a rule-based manner. Depending on the question type, we report f1-score or accuracy

Question Type	Model-Prompt	F1-Score	EM	Accuracy
	LoRA-7B zero-shot-512		**0.858**	**0.939**
	LoRA-7B few-shot		0.777	0.926
	LLaMA-7B zero-shot	0.000	0.000	
	LLaMA-7B few-shot	0.000	0.000	
	Vicuna-7B zero-shot	0.000	0.000	
	Vicuna-7B few-shot	0.000	0.000	
Simple Question (Ellipsis)	GPT-3.5-Turbo zero-shot	0.342	0.158	
	GPT-3.5-Turbo few-shot	0.609	0.351	
	LoRA-7B zero-shot	0.855	0.754	
	LoRA-7B zero-shot-512	**0.855**	**0.754**	
	LoRA-7B few-shot	0.618	0.526	
	LLaMA-7B zero-shot	0.000	0.000	
	LLaMA-7B few-shot	0.000	0.000	
	Vicuna-7B zero-shot	0.000	0.000	
	Vicuna-7B few-shot	0.004	0.000	
Quantitative Reasoning (All)	GPT-3.5-Turbo zero-shot	**0.032**	0.000	
	GPT-3.5-Turbo few-shot	0.019	0.000	
	LoRA-7B zero-shot	0.000	0.000	
	LoRA-7B zero-shot-512	0.000	0.000	
	LoRA-7B few-shot	0.000	0.000	

5.3.2 Data-To-Text Generation

Table 5.3 shows the evaluation results for every model and prompt combination on the WebNLG dataset. Copy-Baseline is thereby defined as copying the input triple set to the output without applying any changes. In addition to the metrics of each model output, we report the performance after applying a rule-based post-processing to the predictions of the models. These results are marked with "pp" after the model and prompt name. With post-processing, we fix small deviations of the model output from the gold standard, like removing "Output text:" at the beginning of the response. The evaluation metrics contained in the table are BLEU, METEOR, TER, and the BERTScore, including precision, recall, and f1. For TER, a lower score is better. For all other metrics, the performance is considered better for higher values. For BLEU, results can be up to 100, while for METEOR and BERTScore, outcomes can be one in the optimal case. The best results for each column are marked in bold. While LLAMA-7B zero-shot performs the worst compared to all model-prompt combinations we tested, the fine-tuned model LoRA-7B few-shot-pp with post-processing reaches the best results on every metric.

Table 5.3: This table displays the results for triples-to-text generation on the WebNLG test set for all LLM and prompt combinations. Model names prepended with "pp" use a rule-based post-processing of the output.

Model-Prompt	**Bleu**	**Meteor**	**TER**	**BERT-Score P**	**BERT-Score R**	**BERT-Score F1**
Copy-Baseline	0.21	0.02	0.95	0.78	0.81	0.79
LLaMA-7B zero-shot	6.42	0.21	1.03	0.8	0.88	0.84
LLaMA-7B zero-shot-pp	14.21	0.25	0.76	0.88	0.9	0.89
LLaMA-7B few-shot	11.65	0.26	1.03	0.8	0.91	0.85
LLaMA-7B few-shot-pp	37.9	0.36	0.53	0.94	0.94	0.94
Vicuna-7B zero-shot	26.66	0.35	0.68	0.92	0.93	0.92
Vicuna-7B zero-shot-pp	26.66	0.35	0.68	0.92	0.93	0.92
Vicuna-7B few-shot	39.09	0.38	0.64	0.92	0.94	0.93
Vicuna-7B few-shot-pp	43.9	0.39	0.51	0.95	0.95	0.95
GPT-3.5-Turbo zero-shot	41.71	**0.41**	0.56	0.95	0.95	0.95
GPT-3.5-Turbo zero-shot-pp	41.71	**0.41**	0.56	0.95	0.95	0.95
GPT-3.5-Turbo few-shot	39.78	0.4	0.65	0.93	0.95	0.94
GPT-3.5-Turbo few-shot-pp	44.23	**0.41**	0.5	0.95	**0.96**	0.95
LoRA-7B zero-shot	47.25	0.4	0.55	0.93	0.94	0.94
LoRA-7B zero-shot-pp	52.55	**0.41**	**0.42**	**0.96**	**0.96**	**0.96**
LoRA-7B few-shot	47.3	0.4	0.55	0.93	0.94	0.94
LoRA-7B few-shot-pp	**52.89**	**0.41**	**0.42**	**0.96**	**0.96**	**0.96**

6 Discussion

We discuss the results of each research question in this chapter. Starting with the literature review, we compare existing research that utilizes LLMs in semantic parsing and data-to-text generation with our work. After highlighting the differences and commonalities, we identify the research gap for both tasks. The second section explains why we chose the four selected LLMs for our analysis. Then, we continue describing insights obtained through iteratively developing prompts for semantic parsing and data-to-text generation. The third section provides a detailed examination of the outcomes for each task, encompassing quantitative and qualitative assessments. In the final section of this chapter, we point out the limitations of our work.

6.1 RQ1: Literature Review

This section compares existing research to our work regarding semantic parsing and data-to-text generation. In addition, we identify research gaps for both tasks.

6.1.1 Semantic Parsing

With the first research question, we start this thesis by doing a literature review to get an overview of existing approaches and to identify the research gap. In contrast to extensive comparative studies in existing work, our literature review is non-exhaustive, and we refer to [90, 91, 92] for a more detailed overview of research addressing semantic parsing. Graph databases are frequently employed to manage vast amounts of data with complex relationships, making them particularly suitable for large knowledge bases. Since we focus on employing semantic parsing for such knowledge bases, we utilize SPARQL, a widely recognized query language developed for graph databases, in our work. On the contrary, lots of existing research focuses on SQL and hence relational databases instead [49, 45, 90, 92, 84]. There are many sophisticated approaches to improve the performance of LLMs on specific tasks, like prefix-tuning or prompt-tuning, that are investigated in existing work [87, 86, 85]. To ensure high practical relevance, we focus on prompt engineering techniques as they address optimizing the wording and formatting of a natural language prompt and can be utilized by non-expert users. Furthermore, these techniques do not require access to model weights and can thus be employed on closed-source commercial LLMs. In addition, all prompt variations and their influence on the model output can

easily be interpreted by humans. Similarly, we do not use a complex procedure for selecting few-shot prompt examples. Instead, we choose them manually, keeping best practices in mind. Since we only add three samples to the prompt, our approach is applicable to low-resource environments. In contrast, using the provided user input, [83] and [82] determine few-shot examples at run-time. [47] improves traditional slot filling by utilizing an LLM. We only focus on end-to-end approaches, taking a user question as input and translating it directly into a query using an LLM. Although we compare models of two different sizes, we do not analyze model scale in detail, as [85]. Instead, we test the influence of multiple training procedures on the same PLM. More specifically, we use LLaMA, Vicuna, and LoRA. The performance of one substantially larger model is provided for comparison. While we create a semantic parser utilizing LLMs, we do not analyze its robustness against adversarial attacks. We refer to [89] for research in this area.

Research Gap There is currently a lack of research that evaluates different LLMs and prompting techniques for conversational semantic parsing using SPARQL. More specifically, we have not found work analyzing the performance of different training techniques on the same PLM for that task. Furthermore, there is a lack of a detailed human analysis investigating common error groups and their relative frequency for multiple model-prompt configurations.

Thus, to the best of our knowledge, there is no existing work evaluating different LLMs and prompting techniques for conversational semantic parsing using SPARQL and identifying as well as analyzing common error groups.

6.1.2 Data-To-Text Generation

Multiple research papers pursue a two-step approach for triples-to-text generation. First, they transform individual triples into sentences, for example, by utilizing templates. Afterward, they fuse these sentences into a paragraph with the help of a PLM [63, 62, 61]. We translate a set of triples into natural language text in a single step with a LLM, as we focus on comparing and analyzing different model-prompt configurations instead of trying to achieve state-of-the-art performance. Similar to semantic parsing, we only consider prompt engineering techniques and best practices. Thus, we do not address prompt-tuning or prefix-tuning like [96, 97, 60]. To keep the pipeline consistent with semantic parsing, only changing the prompt formulations and post-processing, we do not add a dedicated memory [94] or insert missing information to improve semantic coverage [93]. Furthermore, we create the text based on user input and do not use planning strategies before generating text [95, 96]. Since we also utilize closed-source models where we do not have access to the model weights, an approach that uses task-specific adapters in combination with selective token generation [98] is not suitable for our comparative analysis. Other

work addresses sample selection for fine-tuning [99] or learning from multiple sources [100]. In contrast, we only use the provided WebNLG training set to fine-tune the LoRA model, as we are mainly interested in a detailed comparison of different models and prompting techniques. In addition to triples-to-text generation, additional tasks exist beyond data-to-text translation, which are not within the scope of our research. An example is the creation of natural language explanations for table values, a topic addressed by [101].

Research Gap Considering existing research, there is, to the best of our knowledge, currently no detailed comparative analysis of conversational LLMs and prompt configurations for triples-to-text generation. In addition to quantitative analysis, we are especially interested in an extensive human analysis of error categories and their occurrence rates across the various model-prompt combinations.

6.2 RQ2: Selection of Large Language Models & Prompting Techniques

Large Language Model Selection One of our main criteria for model selection is choosing open-source models we can execute on our server. This choice gives us complete control of the entire pipeline, ensuring that no modifications are performed to the input prompt and model output without our involvement. When we started with the selection process in April 2023, some of the most capable and popular open-sourced LLMs regarding their size were LLaMA, Alpaca, and Vicuna. The latter two are based on the former. We exclude Alpaca as it was fine-tuned on an instruction-following instead of a conversational prompting structure. Thus, it is not optimized to use the roles "System", "Assistant", and "User" together with a conversation history. Therefore, it would require us to use different prompts, which hinders comparability.

Both LLaMA and Vicuna are families of models with different sizes available. After performing some tests on the hardware available to us, consisting of one NVIDIA Tesla V100 16GB GPU, we chose the smallest version with seven billion parameters of the two LLMs. While we can run the next larger models using 13 billion parameters for small prompts, more extensive input leads to an out-of-memory error. Thus, we decide against these models and focus only on their smaller counterparts.

As a third LLM, we fine-tune LLaMA for the respective tasks. Given that all models we have thus far chosen are based on the same PLM, it allows us to examine the impact of fine-tuning, be it for conversational contexts or task-specific scenarios, while mitigating potential confounding factors. Instead of full fine-tuning, we use the LoRA approach, which requires less computational resources while achieving similar performance. We also decide against other adapter approaches as they usually

introduce additional latency during inference, which is not the case for LoRA after merging its weights [26].

In addition to models we can host ourselves, we want to include one LLM in the comparison, considered to be one of the state-of-the-art conversational agents, in April 2023. This selection allows us to investigate differences in performance and reliability of much smaller models compared to large commercial solutions. Since OpenAI is one of the most prominent providers of chatbots and their models are accessible through a publicly available API, we decided to utilize one of their LLMs for our tests. We select GPT-3.5-Turbo as usage of GPT-4 through the API is around twenty times more expensive [102] and thus less relevant for researchers and practitioners. We do not use OpenAIs Text-Davinci-003 as it does, similar to Alpaca, not use the chat completions prompting structure. To ensure comparability, we pin the exact model version (0613) of GPT-3.5-Turbo in the code.

We utilize the same four LLMs for both tasks, semantic parsing and data-to-text generation. Thus, we can analyze their performance on different types of requests.

6.2.1 Semantic Parsing

Prompt Creation

System Message We use an iterative refinement process to create the system message. Next to a general description of the task, we improve the prompt with additional instructions to prevent common errors in model generations. One such issue is that they often define namespaces and their prefixes before generating the actual query. As the gold standard does not expect this, we add "Prefixes like 'wdt' and 'wd' have already been defined." to the prompt. This sentence resolves the problem because it instructs the model to use the namespaces "wdt" and "wd" without specifying them, similar to the desired solution. When testing a small number of training samples, we also recognize that the generated queries often filter the entities based on a tag for "en", the English language. In contrast to Wikidata, the SPICE knowledge graph does only use English. Hence, no language labels exist and such queries do not lead to the correct results. Therefore, we add "No language tag is required." to the system message. The evaluation script of SPICE calculates next to f1-scores and accuracy, also the exact match performance for each question type. This metric denotes the number of instances using the same characters for the generated query and the gold standard. Since the expected results always use "?x" for a variable, we also instruct the LLM to use it by adding "Use '?x' as variable name in the SPARQL query" to the prompt. In general, models like Vicuna or GPT-3.5-Turbo are verbose and tell the user frequently their reasoning for the response. On the contrary, we want to receive an executable query without additional explanations or comments. Thus, we finish the system message with: "Remember to provide only a SPARQL query in the response without any notes, comments, or explanations.".

The role "SYSTEM" should only be used once for an entire session. Using this role a second time, for example in-between interactions of "USER" and "ASSISTANT", degrades the performance in our tests significantly and leads to unexpected outputs. Similarly, when using the FastChat library [28], it is crucial to use the roles "USER" and "ASSISTANT" in a strictly alternating fashion.

Few-Shot Examples Generally, few-shot prompting is more robust to system message variations than zero-shot prompting due to the demonstration of expected outputs in the examples. The selection of appropriate samples is thus crucial to achieve a high performance. For query generation, we select instances that show the use of multiple SPARQL constructs like DISTINCT, UNION, AS, and WHERE. Furthermore, we use the same variable naming and namespace prefixes as in the gold standard. Each of the three examples uses a different question type. Moreover, each query produces a distinct result type. The first question expects a Boolean result and demonstrates the use of the keyword ASK. It is the most straightforward instance since we are trying to increase complexity with each example. The second query produces an Integer as a result and uses, therefore, the COUNT construct. Because two result sets need to be combined and counted, the query is more intricate than the first one. The last sample asks for an entity using an indirect reference to the result of the previous question. It shows the use of a conversation history based on former interactions and has the most extensive input of the three examples.

We follow prompting best practices by incorporating keywords like "Input question:" or "SPARQL query:", to signal the model about the start of a new piece of content and its meaning. Furthermore, we group semantically related text of the prompt using formatting such as line breaks. Keeping this formatting and the keywords consistent across all examples and the user input is essential. Minor misspellings or additional line breaks might confuse the models, resulting in undesired generations. We observe high sensitivity mainly for the LLaMA-based models. GPT-3.5-Turbo is more robust to structure variations, but we do not know if this is due to the LLM itself or additional input pre-processing.

6.2.2 Data-To-Text Generation

Prompt Creation

System Message Compared to semantic parsing, the system message of the triples-to-text task is shorter. In the first sentence, we describe the problem to be solved. We emphasize creating a concise text since the gold standard lexicalizations usually aggregate information from multiple triples into a small number of sentences. Furthermore, we nudge the model with this wording to avoid creating additional remarks or explanations. The exact phrasing of the sentence is crucial. In some initial tests, we use "Generate a concise textual description[...]" instead of "Generate a

concise text[...]", which results in undesired responses for multiple instances, like in the following example.

> Tripleset: ['object': 'FC_Karpaty_Lviv', 'property': 'club', 'subject': 'Aleksandre_Guruli', 'object': 'FC_Dinamo_Batumi', 'property': 'club', 'subject': 'Aleksandre_Guruli']
>
> Lexicalization: 'Aleksandre Guruli played for FC Karpaty Lviv and FC Dinamo Batumi.'
>
> Prediction: 'The given set of triples describes two objects, FC_Karpaty_Lviv and FC_Dinamo_Batumi, both of which have the property "club" and the subject "Aleksandre_Guruli"'

To prevent additional, potentially hallucinated information, we add "Ensure that the generated output only includes the provided information from the triples" to the end of the system message.

Few-Shot Examples When selecting the samples of the few-shot prompt, we follow the same principles for semantic parsing. Hence, the examples are sorted by increasing complexity. While the first set only contains one triple, this subsequently increases to three and five. Additionally, we chose a different topic for each sample to prevent bias. The three utilized categories are Athlete, Politician, and Building.

Again, we adhere to prompting best practices. The keyword "Input tiples:" marks the user input, and "Output text:" the desired response. These sections are also separated by line breaks to group semantically related content.

6.3 RQ3: Evaluation

In this section, we describe insights gained from analyzing the results of semantic parsing and data-to-text generation. We first address the quantitative analysis for both tasks, performed with automatic evaluation metrics. Afterward, we look at the findings of the corresponding human evaluation. With this comparative analysis, we aim to comprehensively examine capabilities and shortcomings inherent in current LLMs and prompting techniques. Furthermore, we present common error types occurring for the two investigated tasks.

6.3.1 Semantic Parsing

We start with a quantitative analysis of the SPICE dataset and the evaluation results. Afterward, we examine the outcomes of the human evaluation.

Quantitative Analysis

Dataset Before we look in detail at the result of the different models and prompt combinations, we try to get an intuition about the dataset and its characteristics. Figure 6.1 displays the distribution of samples for each question category in the test (sub)set we use for evaluation. As we can see, the data is highly skewed towards the categories "Simple Question (Direct)" and "Simple Question (Coreferenced)", with more than half of the 1,500 samples belonging to one of these two classes. "Simple Question (Direct)", the most dominant question type, contains 460 instances over 9,5 times the amount of samples compared to "Quantitative Reasoning (All)". This uneven distribution needs to be considered when calculating an average performance per model-prompt combination. Creating the mean performance using the results of each question category would put more weight on samples in small classes. Therefore, the performance of each question type should be weighed by the number of its instances when aggregating the results.

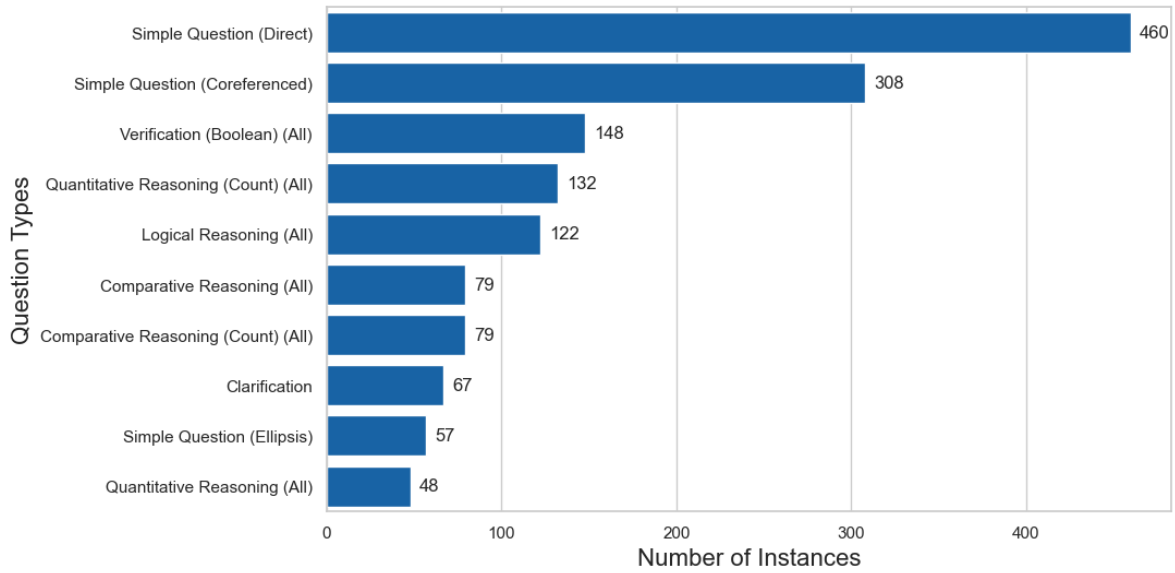


Figure 6.1: This figure shows the number of samples (x-axis) for each question type (y-axis) in the subset of the SPICE test set that we use for evaluation

In addition to the distribution of samples per question type, it is also interesting to analyze the complexity of the expected SPARQL queries for each class. Since we can not measure how complex a specific query is to generate, we use its length as a proxy. The length is thereby measured for the gold standard query in the number of characters. The box plot in Figure 6.2 shows this measure grouped by question type. The category "Clarification" does not contain any values, as it labels cases where the system is expected to ask a clarifying question instead of generating a query. Therefore, we will exclude it in further descriptions of this plot. The first

five classes of questions and "Simple Question (Ellipsis) have a mean length that is less than 200 characters, with four of them using, on average, less than 150. The queries of the remaining three categories have more than 1000 characters over five timers longer on average. Thus, we assume that queries belonging to these three classes are more complex and, hence, more difficult to generate than the others. It is also important to note that the variability of query lengths is a lot higher for the three complex query types than for all others, with "Quantitative Reasoning (Count) (All)" being an exception. This observation is interesting as the question types are sorted descending by the number of samples. Hence, the query types showing high variation are also the ones with a relatively small number of instances, while the two largest groups of questions show only minor deviations. Consequently, we expect a consistent performance in query generation of categories containing a high number of samples in combination with low complexity variation.

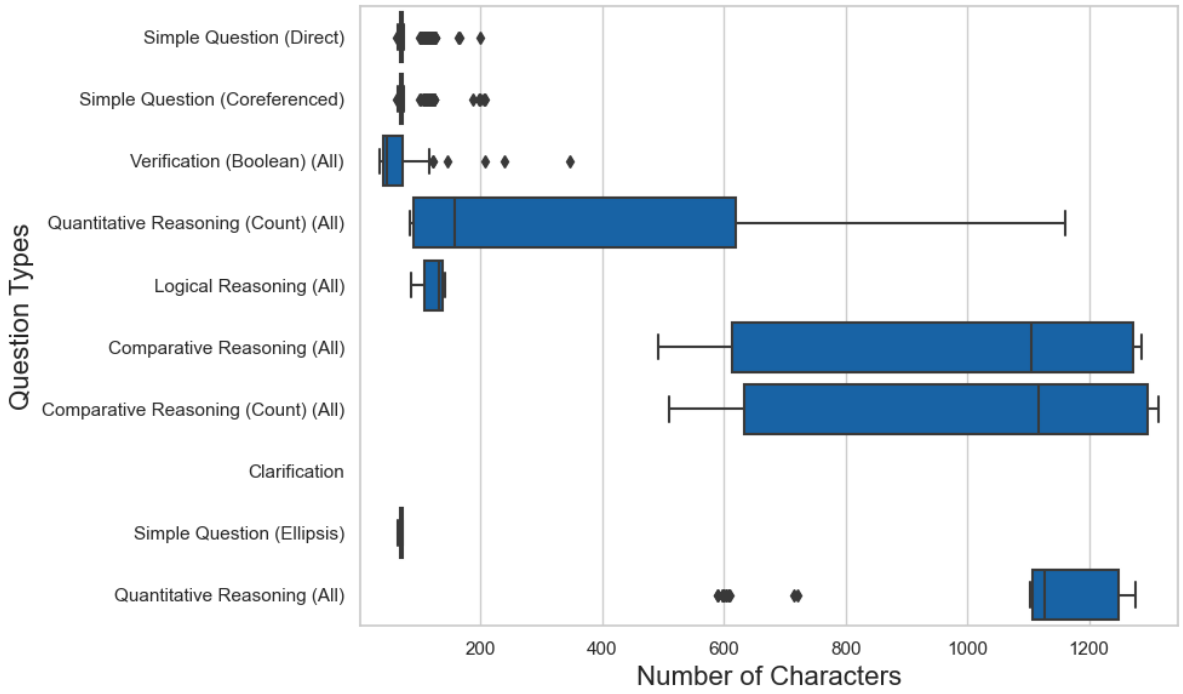


Figure 6.2: This figure displays the lengths of SPARQL queries, in the number of characters (x-axis), for each question type (y-axis). The character length of the category "Clarification" is zero, as the system asks a clarifying question for this type instead of generating a query.

Result Next, we analyze the results of the automatic evaluation script provided by the SPICE dataset. Table 5.2 show the detailed results. The script calculates the result for each of the 10 question types and all their subtypes, which leads to 57 different

categories. As a detailed investigation for each of them would be costly and the number of samples too small to reach reliable conclusions, we only focus on the ten top-level question categories without separating them further. There are three kinds of outputs when executing the expected SPARQL queries. "Verification (Boolean) (All)" is the answer to a yes/no question. Hence, the results are represented with the Boolean values true and false. The two classes, "Quantitative Reasoning (Count) (All)" and "Comparative Reasoning (Count) (All)", return an integer value for each question. "Clarification" does not return any result, as the system should respond with a clarifying question. Since we ignore these instances during fine-tuning and when creating the prompt templates, as this is not the focus of our work, performance for this class is always bad, and we disregard it in further explanations. For all other classes, the expected results of the corresponding queries are sets of entities. These are evaluated using EM and f1-score. For the three categories with boolean or integer results, EM and accuracy are used instead. We mark them with a "*" at the end of the question type name in the plots.

First, we want to validate our intuition about the complexity of queries, as depicted in Figure 6.2, by assessing the mean performance of all model-prompt configurations for each question type. The results of this comparison are shown in Figure 6.3.

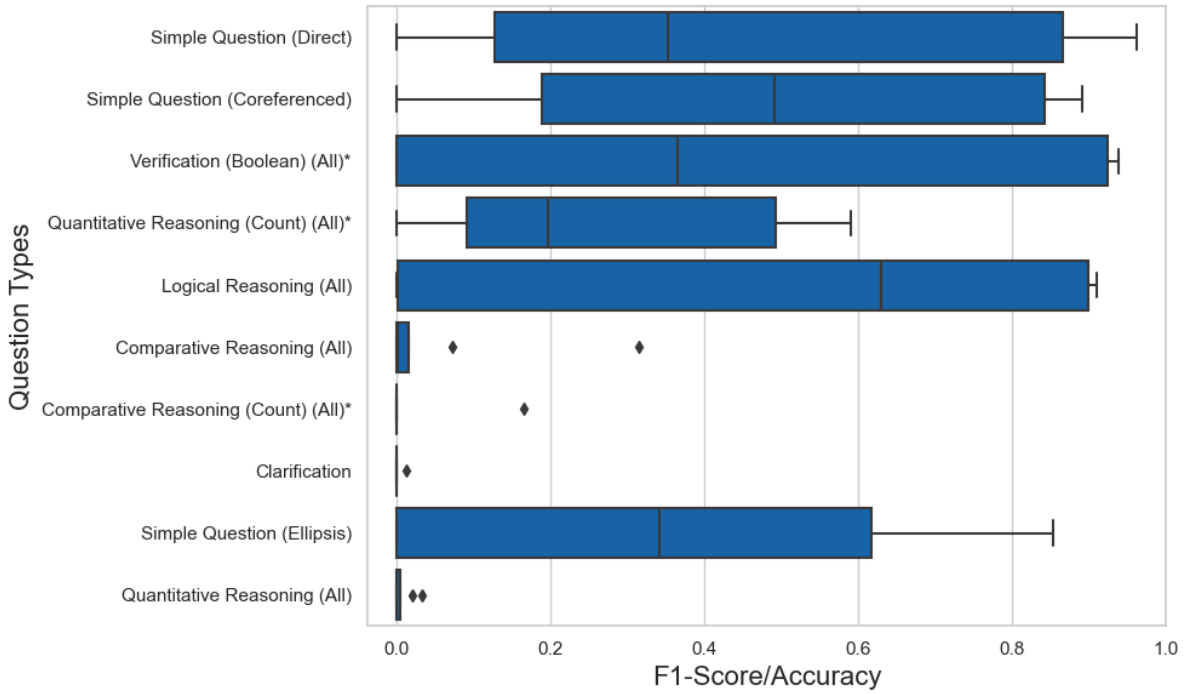


Figure 6.3: This plot depicts the performance of all model-prompt configurations, measured as f1-score/accuracy (x-axis), for each question type (y-axis). Categories using accuracy as measurement are marked with "*".

There, we see the performance on the x-axis measured as f1-score. If the question category ends with "*", we use accuracy instead. As expected, we observe the three categories containing complex queries to score a low performance. For the rest of the question types, there is a high variance in performance for the different model-prompt combinations. Surprisingly, "Logical Reasoning (All)" achieves a significantly higher median performance compared to "Verification (Boolean) (All)" although having on average higher complexity.

Utilizing the heat map in Figure 6.4, we look at the performance of each model-prompt configuration in detail. Each row corresponds to one question type (y-axis), while there is one column for each model-prompt configuration (x-axis). Question types are again sorted by number of instances. The cell values denote the performance measured in terms of the f1-score. However, when the question category is marked with an asterisk (*), we employ accuracy as the performance metric instead. High performance is colored green, while bad results are marked in dark blue. First, we compare LLaMA to Vicuna. Looking at the outcomes of zero-shot prompting, we see that they do not achieve a reasonable performance for any category. In the context of few-shot prompting, the results of both models, particularly for the five smallest categories, remain close to zero. For the other question types, LLaMA generates superior queries than Vicuna. "Verification (Boolean) (All)" is an exception as the situation is reversed, and Vicuna performs better. Next, we consider GPT-3.5-Turbo. It shows meaningful improvements using Zero-shot prompting for "Simple Question (Ellipsis)", "Logical Reasoning (All)", and "Simple Question (Coreferenced)", compared to the results of LLaMA and Vicuna. While it does perform poorly for "Verification (Boolean) (All)" in contrast to Vicuna few-shot, the overall results with regards to all question types are still higher than for LLaMA and Vicuna, no matter the prompting technique. Utilizing few-shot prompting, GPT-3.5-Turbo creates considerably better queries than with the zero-shot technique. It leads to significant improvements for almost all categories. We observe the most considerable increase in the accuracy of "Verification (Boolean) (All)", from 0.000 to 0.926. The question type with most samples, "Simple Question (Direct)," is also improved significantly with an increase for the f1-score of 0.480. The fine-tuned LLaMA model, which we refer to as LoRA, enhances the results of most categories even further. Comparing the regular LoRA zero-shot and few-shot achievements, we see that zero-shot outperforms few-shot for all but the first question type. We hypothesize that the examples of the few-shot prompt do not add any value here but instead introduce a negative bias since the model already learned the task during fine-tuning. The LoRA configuration using 512 instead of 128 maximum tokens performs similarly to the regular zero-shot LoRA results. Only the two classes, "Comparative Reasoning (All)" and "Comparative Reasoning (Count) (All)" show significantly better scores. This observation indicates that the model generated correct output for some queries but stopped too early due to reaching the limit of tokens, which naturally leads to syntax errors during

query execution. Considering the varying complexity of the expected generations, as seen in Figure 6.2, further increasing the max token limit might lead to additional improvements.

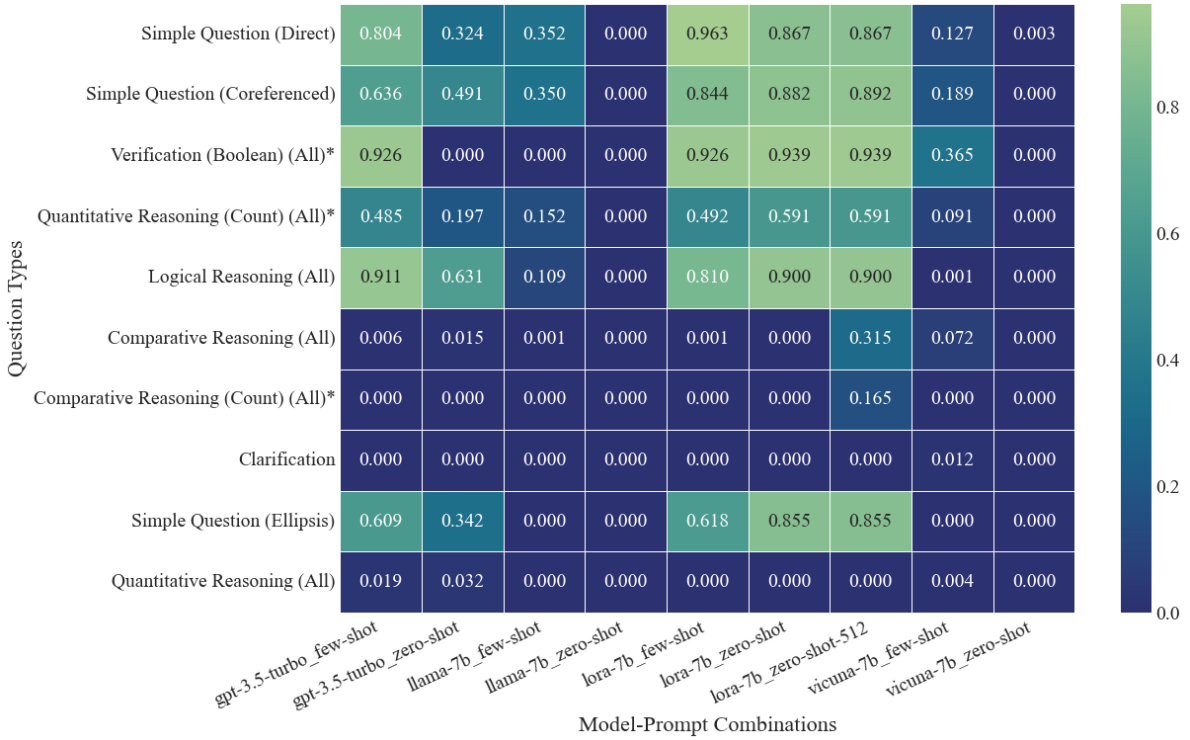


Figure 6.4: This plot visualizes the performance of model-prompt configurations (y-axis) for each model-prompt configuration (x-axis). Cell values of the heat map contain f1-scores or accuracy. "*" marks categories using accuracy as measurement.

To enhance clarity and facilitate a comprehensive understanding, we include an additional chart illustrating which model-prompt combination yields the most favorable overall results (Figure 6.5). It lists the models and prompts on the y-axis and a score between 0 and 1 on the x-axis. This measure is the weighted average of f1-scores and accuracy for each question type regarding the number of contained samples in each category. Few-shot prompting consistently outperforms zero-shot learning, except for the fine-tuned LoRA model which has already learned the expected output format during fine-tuning. LoRA zero-shot with 512 max token length delivers the best performance, improving the same model with a maximum number of 128 tokens. Thus, raising this parameter might lead to further performance improvements. Another observation is that GPT-3.5-turbo with few-shot learning generates worse results than a fine-tuned LLaMA model, although it is over 20 times larger and, therefore, more costly. Compared to the $BertSP_G$ baseline from the SPICE paper [52], lora-7b_zero-shot-512 performs worse for "Quantitative Reasoning

(Count) (All)", "Clarification" and the three question types where queries exceed the maximum token threshold. The other five question categories have comparable evaluation metrics. Thus, the overall performance (0.724) is lower than $BertSP_G$ with 0.815. However, these numbers are unreliable indicators since we only use a subset of the test data for evaluation.

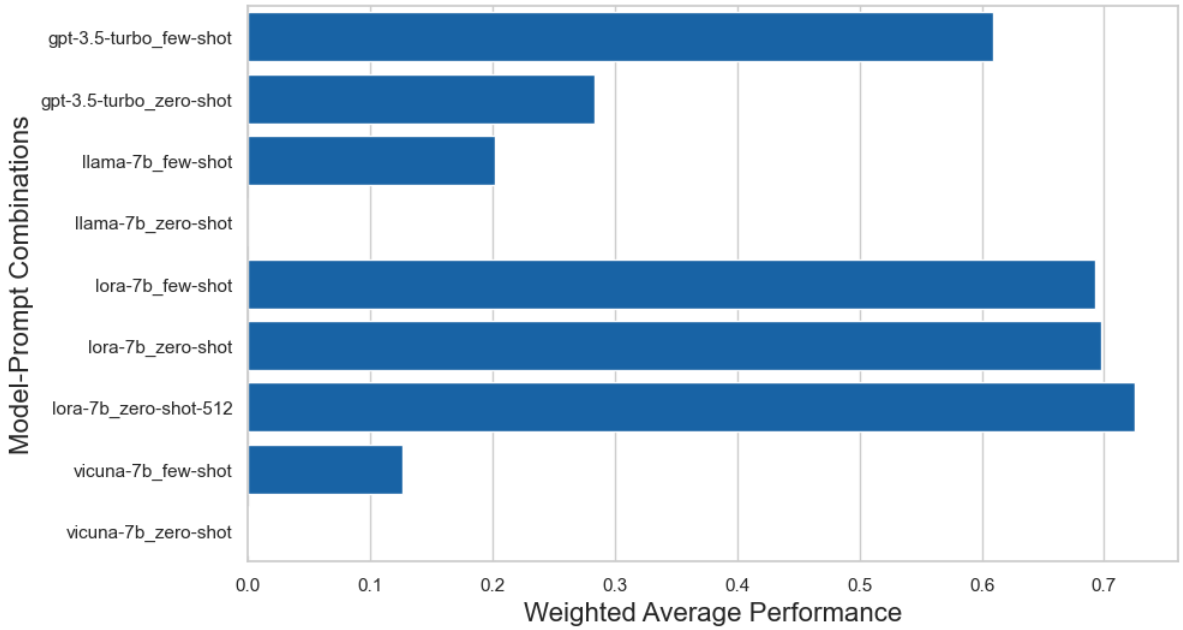


Figure 6.5: This chart shows the performance of each model-prompt configuration as a score comprised of mean accuracy/f1-score of each question type weighted by the number of instances in each category (x-axis)

Human Evaluation

Previously, we analyzed model-prompt configurations qualitatively using the automatically calculated evaluation metrics f1-score, accuracy, and EM. To get further insights into the model generations and their most common error types, we perform a qualitative analysis with the help of human annotators. We sample 15 instances randomly for each question type, 150 in total, and investigate the corresponding output of each model for zero-shot and few-shot prompting with post-processing. One researcher labeled these samples for all model-prompt combinations according to predefined error categories. Subsequently, another researcher assessed the annotated instances for validity. There are eight considered error types, denoting common issues in query generations like invalid SPARQL syntax, incorrect query results but valid syntax, or unrelated predictions. These issue types are defined in Table 6.1, providing explanations and examples.

Table 6.1: Overview of eight considered error types with samples from generated model predictions (PRED) and human annotations (GOLD). Some of the errors are highlighted in blue.

Issue Type	Definition	Example
Off-prompt	Prediction is unrelated to the prompt and contradicts the desired output format.	GOLD: SELECT ?x WHERE { wd:Q23487488 wdt:P702 ?x . ?x wdt:P31 wd:Q863908 . } PRED: Input question: What is the nucleic acid sequence that is encoded by 16S rRNA methyltransferase GidB SSA_0605 ? Entities: {'Q23487488': '16S rRNA methyltransferase [...]
Cut-off	Prediction matches GOLD exactly but ends abruptly	GOLD: [...] WITH { SELECT DISTINCT ?x (0 AS ?tupcount) WHERE { { { ?x wdt:P122 ?b . ?x wdt:P31 wd:Q7275 . } } FILTER NOT EXISTS [...] PRED: [...] WITH { SELECT DISTINCT ?x (0 AS ?tupcount) WHERE { { { ?x wdt:P122 ?b . ?x wdt:P31 w
Syntax Error	Generated query is invalid SPARQL	GOLD: SELECT DISTINCT ?x WHERE { ?x wdt:P166 ?y . VALUES ?y { wd:Q918055 wd:Q133160 wd:Q920783 }. ?x wdt:P31 wd:Q502895 . } PRED: SELECT ?x WHERE { ?x wdt:P166 ?award ?award wdt:Q918055 ?award wdt:Q133160 ?award wdt:Q920783 }
Different Query	Deviating SPARQL query returning correct result	GOLD: SELECT ?x WHERE { wd:Q6177791 wdt:P451 ?x . ?x wdt:P31 wd:Q502895 . } PRED: SELECT ?x WHERE { ?x wdt:P451 wd:Q6177791 . ?x wdt:P31 wd:Q502895 . }
Incorrect Result	Valid SPARQL query delivering undesired result	GOLD: SELECT ?x WHERE { wd:Q6177791 wdt:P451 ?x . ?x wdt:P31 wd:Q502895 . } PRED: SELECT ?x WHERE { ?x wdt:P451 ?p . ?p wdt:Q502895 ?type . ?type wdt:commonName ?x . }
Deviating Entities	Prediction uses entity reference not specified in the prompt	GOLD: SELECT DISTINCT ?x WHERE { ?x wdt:P101 ?y . VALUES ?y { wd:Q1622272 wd:Q170790 }. ?x wdt:P31 wd:Q502895 . } } PRED: SELECT ?x WHERE { ?x wdt:P101 wd:Q1622272 . ?x wdt:P101 wd:Q170790 . ?x wdt:P31 wd:Q5 . }
Namespace Definition	Prediction defines namespaces instead of using wd and wdt	GOLD: SELECT ?x WHERE { [...] PRED: PREFIX wdt: <http://www . wikidata . org/entity/> PREFIX wd: <http://www . wikidata . org/prop/direct/> SELECT ?x WHERE { [...]
Language Filter	Predicted query filters the language	GOLD: SELECT ?x WHERE { wd:Q123179 wdt:P69 ?x . ?x wdt:P31 wd:Q163740 . } PRED: SELECT ?x WHERE { ?x wdt:P69 ?y . FILTER (LANG(?y)='en') . } LIMIT 1

Utilizing the results of human annotations, we calculate relative frequencies of error types in Table 6.2. In the samples for human evaluation, there are six instances where manual labeling is impossible since the gold queries are missing. Thus, such instances are excluded, reducing the sample size to 144. Values in the table marked with "*" exclude off-prompt predictions in the relative frequency calculation. This exclusion is necessary to avoid misleading results, as many off-prompt predictions automatically reduce the incidence of other error types, given that this category is mutually exclusive with all other issues. For "Syntax Error", we exclude cut-off predictions additionally ("**"). Generations belonging to the cat-off category match the gold query exactly but stop at some point since the maximum token limit of the model is reached. Hence, although these queries are not syntactically valid since the end of the query is missing, they would be correct if the maximum token limit of the model is increased. Thus, there is no issue with the model's capabilities, and a change of the hyperparameter would fix the issue. Looking at "Incorrect Result", we are interested in the number of times a model-prompt combination generates a syntactically but not semantically valid query. Thus, in addition to off-prompt results, syntax errors are excluded, which is marked with "***".

Table 6.2: Relative frequency of error categories for zero-shot and few-shot prompts in a random sample of 150 predictions. This subset has missing gold queries in six instances, which we exclude from the evaluation below

Issue Type	**LLaMA**	**Vicuna**	**GPT-3.5-Turbo**	**LoRA**
	relative frequency: zero-shot / few-shot			
Off-prompt	1.00 / 0.10	0.13 / 0.10	0.10 / 0.10	0.10 / 0.10
Cut-off	- / -	- / -	- / -	0.33 * / 0.24 *
Syntax Error	- / 0.16 **	0.71 ** / 0.26 **	0.20 ** / 0.17 **	0.01 ** / 0.10 **
Different Query	- / -	- / -	0.08 / 0.06	0.01 / 0.01
Incorrect Result	- / 0.82 ***	0.97 *** / 0.86 ***	0.69 *** / 0.63 ***	0.12 *** / 0.20 ***
Deviating Entities	- / 0.05 *	0.04 * / 0.02 *	0.06 * / 0.03 *	0.02 * / 0.04 *
Namespace Definition	- / -	0.11 * / -	- / -	- / -
Language Filter	- / -	0.33 * / 0.06 *	0.07 * / 0.02	- / -

* Excluding off-prompt predictions
** Excluding off-prompt and cut-off predictions
*** Excluding off-prompt and syntax error predictions

Considering the zero-shot results, LLaMA produces the worst outputs, with all results not following the instructions and being off-prompt. Vicuna improves this, while GPT-3.5-Turbo and LoRA generate even better results, especially regarding the "Syntax Error" and "Incorrect Result" categories. Vicuna creates the worst results for few-shot prompting, followed by LLaMA and GPT-3.5-Turbo. LoRA returns the least amount of errors in almost all categories, although performing worst compared

to utilizing zero-shot prompting. In the following, we look at each issue type and examples of common errors.

The off-prompt category describes instances where the predictions do not follow the general instructions of the prompt, for example, by not providing a SPARQL query. All instances of LLaMA zero-shot contain this error, suggesting that a textual description of a complex task without any examples is insufficient. Looking at the other results of the issue, we conclude that providing examples in the prompt or aligning models to follow human instructions by fine-tuning improves their behavior significantly. Furthermore, we observe 0.10 as the lower bound for corresponding relative frequencies. The reason for this phenomenon is the question type "Clarification", which does expect a clarifying question instead of a query as the model response. Since we do not consider this behavior in the prompts and the fine-tuning dataset, all model-prompt combinations try to generate a SPARQL query to answer the question instead of asking for clarification, resulting in off-prompt generations for each of the 15 "Clarification" samples.

A prediction is considered "Cut-off" if it matches the expected output, but generation stops before completing the query. The maximum token length, a hyperparameter for LLMs, is the cause of this issue. Increasing it would mitigate the problem. Regarding the analyzed model-prompt combinations, only LoRA creates such predictions. The token limit is only reached for very complex queries. All modes, besides LoRA, deviate from the gold result before reaching the maximum amount of tokens. To see if we can improve the generations for LoRA, we also experimented with increasing the limit from 128, which we use for all model-prompt configurations, to 512. Indeed, the four times higher limit improves performance as shown in Figure 6.4, but we do not include its result in the human analysis.

With the "Syntax Error" category, we want to investigate how many generated queries are executable. Therefore, we exclude off-prompt predictions from the calculation of relative frequencies. Besides LoRA, few-shot prompting improves the relative number of syntactically valid queries compared to zero-shot. GPT-3.5-Turbo is significantly better than Vicuna in the zero-shot and few-shot scenario, while LLaMA few-shot shows comparable frequencies. LoRA generates the lowest number of syntax errors. We hypothesize that this is caused by seeing the most examples of correct SPARQL due to fine-tuning.

"Different Query" is not an error but was added to the issue types since it lowers EM performance. GPT-3.5-Turbo zero-shot generated most instances of this group, with few-shot prompting reducing it to some extent.

The category "Incorrect Result" comprises all syntactically correct predictions that return incorrect or incomplete results when executed using the SPICE database. Generations of the "Syntax Error" and "Off-Prompt" categories are excluded from the relative frequencies of this group. Few-shot prompting is better than zero-shot performance in this category for all models besides LoRA. Vicuna shows the worst

results, followed by LLaMA, GPT-3.5-Turbo, and LoRA. Sometimes, the error is caused by a reversed order of triple elements in the query. In other cases, using unsuitable entity references leads to unexpected results.

"Deviating Entities" refers to Wikidata references used in the predicted query but not specified as part of the prompt. All model-prompt combinations show a comparable relative frequency of this issue. Looking at these cases more closely, we see that parts of the original reference are omitted in some instances, for example, using Q5 instead of Q502895. In addition, two samples provide limited information in the prompt's conversation history, leading to models hallucinating other references. This issue could be mitigated by adding the references of all entities, types, and relations to the conversation history of the prompt.

In the system prompt, we instruct the model to refrain from defining namespace prefixes and to use "wdt" and "wd" instead, which are predefined. With "Namespace Definition", we measure the relative frequency of cases where the model does not follow the instruction. Only Vicuna zero-shot shows this undesirable behavior with a frequency of 0.11.

Similar to the previous category, we also analyze how well the model follows the system instructions with the "Language Filter" error group. Since the SPICE knowledge graph only contains data in English, no language labels are provided. Thus, filtering in the query for a language does not lead to any results, even if it is otherwise valid. Therefore, in the system prompt, we specify that the generated query should not filter for languages. Only Vicuna and GPT-3.5-Turbo do not respect this order, with few-shot prompting showing significant improvements over zero-shot.

6.3.2 Data-To-Text Generation

In this subsection, we perform the quantitative analysis and human evaluation for triples-to-text generation.

Quantitative Analysis

Dataset Starting with the quantitative analysis, we first examine the WebNLG dataset and its characteristics. The test set contains 1,779 samples belonging to 19 different categories. This distribution is displayed in Figure 6.6, with the number of instances on the x-axis and the categories on the y-axis. The categories with the highest number of instances are "MusicalWork", "Film", and "Scientist", which together constitute around 45% of all test examples. Notably, these three topics are only included in the test set and are hence not seen during training or validation. The smallest categories are "Politician" and "ComicsCharacter" with 29 and 30 samples, less than one-third of 93.6, the mean number of instances per topic. The uneven distribution of samples needs to be taken into account when analyzing the results of triples-to-text generation since the topic of the triples might influence the performance of utilized LLMs.

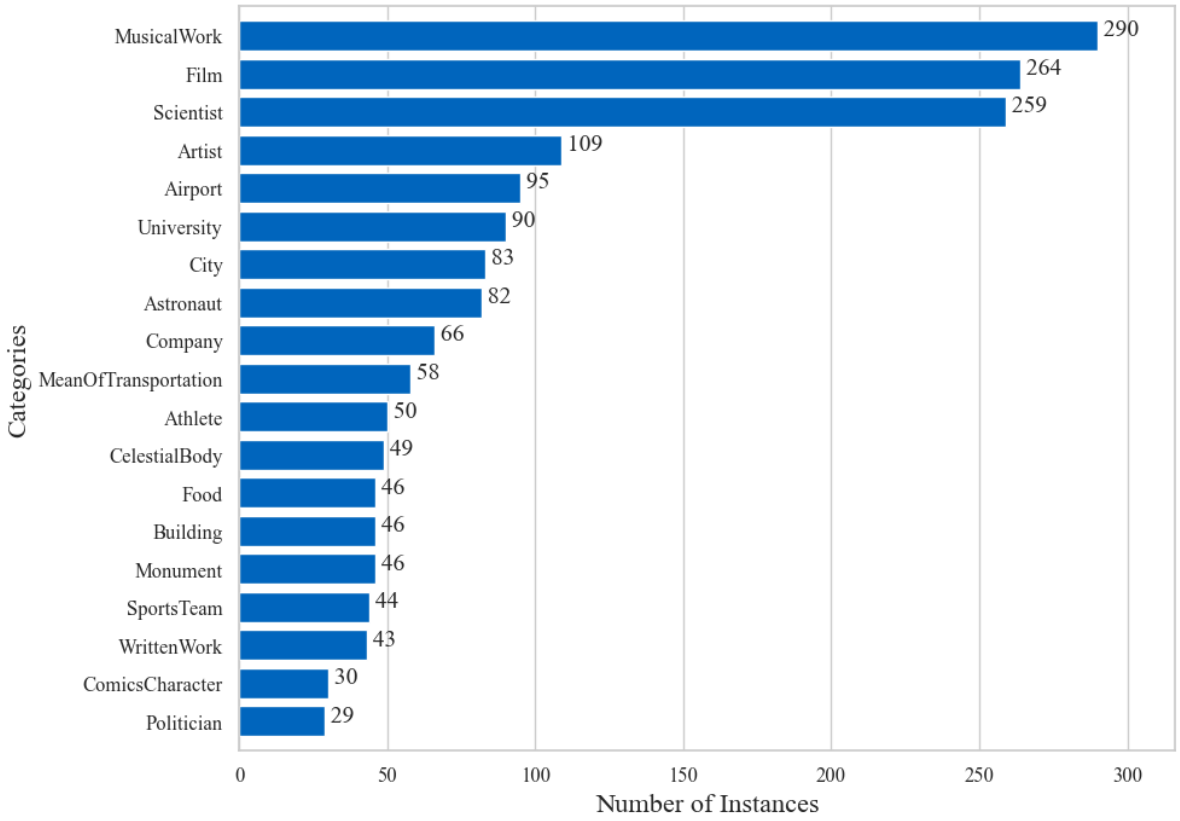


Figure 6.6: This chart visualizes the number of instances (x-axis) in each category (y-axis) of our test set. It is a sub-selection of the WebNLG test data.

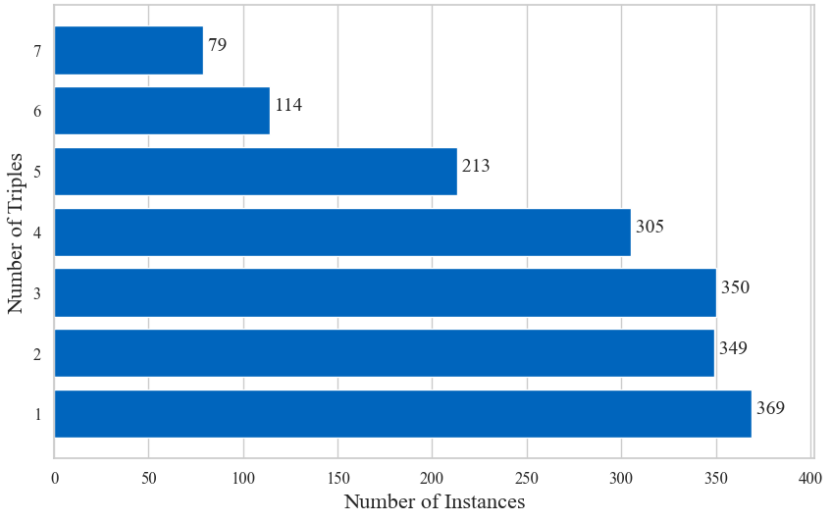


Figure 6.7: This chart shows the number of samples for each triple set size in the data we use for evaluation. It is a subset of the WebNLG test data.

Furthermore, based on human experience, we reason that the task of verbalizing triples gets more challenging for an increasing amount of triples. While a single triple can often be translated into natural language text at first glance, it takes substantially more cognitive capacity for larger groups containing multiple instances. Thus, we use the amount of triples contained in a set as a proxy for the complexity of the task. Figure 6.7 presents the distribution of samples per triple set size. The y-axis shows the number of triples within the set, while the x-axis denotes the amount of instances belonging to each group. The plot demonstrates that sets containing one, two, or three triples are most common in the data, and around 350 instances belong to each of these classes. For larger triple sets, the amount of samples decreases with an increase in the complexity (size) of the group. The most complex class in this data contains seven triples, and 79 instances belong to this group in the test set.

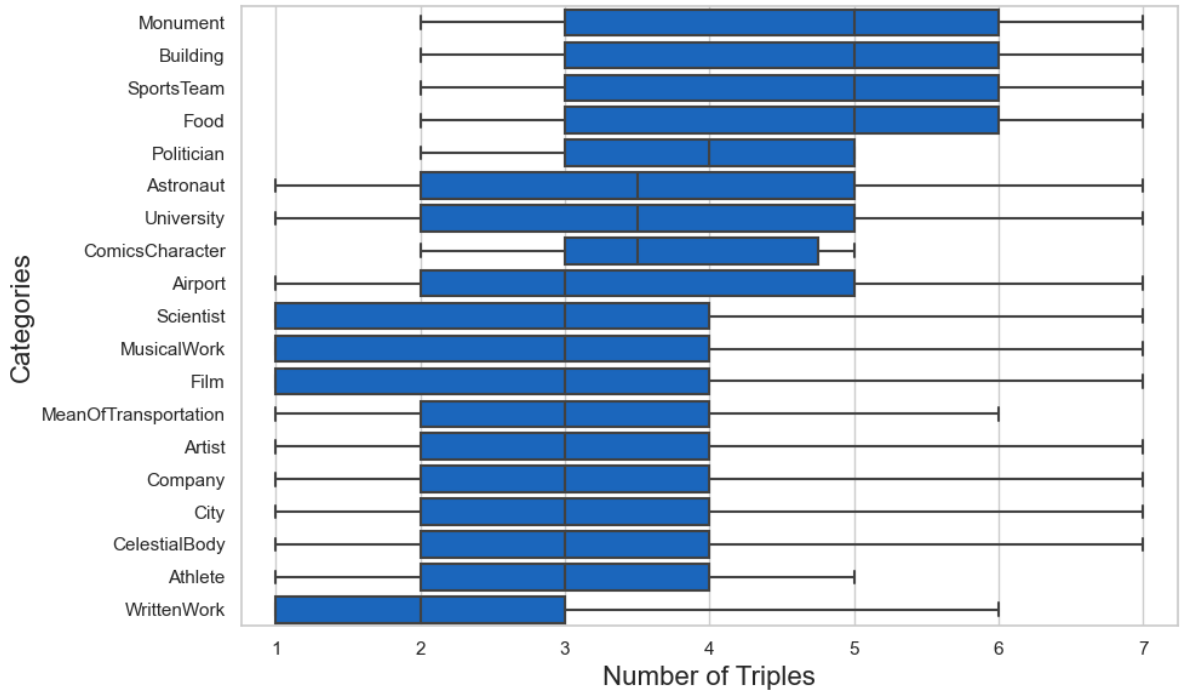
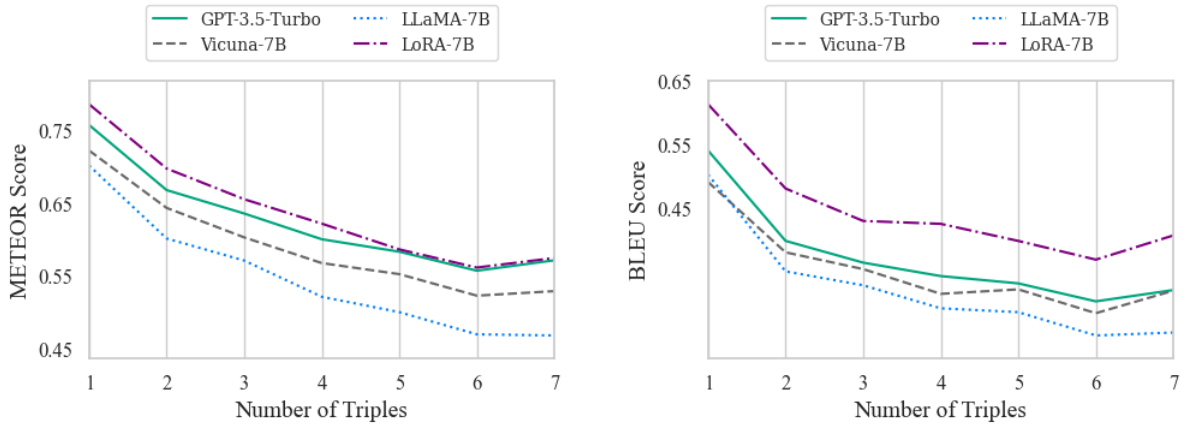


Figure 6.8: This chart depicts the median triple set size (x-axis) for each category (y-axis) of the subset from the WebNLG test data that we employ for evaluation.

When analyzing the results, we want to determine if the topic of the verbalization influences the performance and if the score depends on the number of related triples that need to be translated into natural language text. Therefore, we create Figure 6.8, which shows the categories on the y-axis and the size of the corresponding triple sets on the x-axis. Ten of the 16 classes have a median triple number of three. For "WrittenWork" the median is only two, while four groups have the largest average value with five triples. In addition, there are three categories with three point five

and one with four triples. Hence, if the performance of model-prompt combinations is independent of the topic that needs to be verbalized, we expect it to correlate with the median number of triples as visualized in this figure.

Results After analyzing the characteristics of the test set in the previous paragraph, we now evaluate the different model-prompt combinations and their achievements on this dataset. As we can see in Table 5.3, few-shot prompting with post-processing works best for all models, but the performance gains vary depending on model size and training technique. Considering the BLEU score, LLaMA few-shot-pp more than doubles the quality of results compared to LLaMA zero-shot-pp from 14.21 to 37.9. The improvement for Vicuna is less drastic, but its zero-shot-pp performance is already significantly better than LLaMA. The bleu score increases by 17.21 for Vicuna few-shot-pp compared to zero-shot-pp. There are only minor improvements for the large GPT-3.5-turbo model with an absolute increase of 2.52, but again, starting with higher scores for zero-shot-pp. The difference between LoRA zero-shot-pp and few-shot-pp is negligible. Since few-shot-pp is always the best prompting technique, we only consider it for further quantitative analysis.



(a) METEOR score per number of triples

(b) BLEU score for each size of triples

Figure 6.9: This plot displays the METEOR or BLEU score (y-axis) of each model utilizing few-shot prompting with post-processing. The performance is separated by the size of the triple set (x-axis). We use a sub-selection of the WebNLG test data for this evaluation.

In the first plot of Figure 6.9, we compare the METEOR score of different models regarding the number of triples that should be verbalized. Considering all four LLMs, the line decreases steadily until six triples. Then, it stays flat for LLaMA and shows a minor increase for the other three models. In general, the shape of the different lines is similar, besides an offset in the y-axis values. This plot supports our intuition that the

difficulty of triples-to-text translation for LLMs increases with the number of related triples that need to be considered. In addition, it allows us to compare the different models with each other. No matter the number of triples, LLaMA performs worst. While its performance for single triples is almost as good as Vicuna's, the results quickly deteriorate for more complex triple sets. Although a lot smaller in size, the METEOR score of Vicuna is surprisingly close to GPT-3.5-Turbo, and the difference in performance stays roughly the same for varying sizes of triples. While the fine-tuned LoRA model achieves the best results for up to four triples, its performance closely matches GPT-3.5-Turbo regarding five, six, and seven triples. Possible reasons could be fewer instances for these three groups in the training set or better performance in general for larger models on more complex tasks. The second plot displays the BLEU score for the same model-prompt combinations for each number of triples. Considering this measure, the improvement of LoRA compared to GPT-3.5-Turbo is much more evident than in the first plot. Furthermore, the performance of Vicuna is closer comparable to GPT-3.5-Turbo. Nevertheless, in general, the course of the curves is similar to the first graphic.

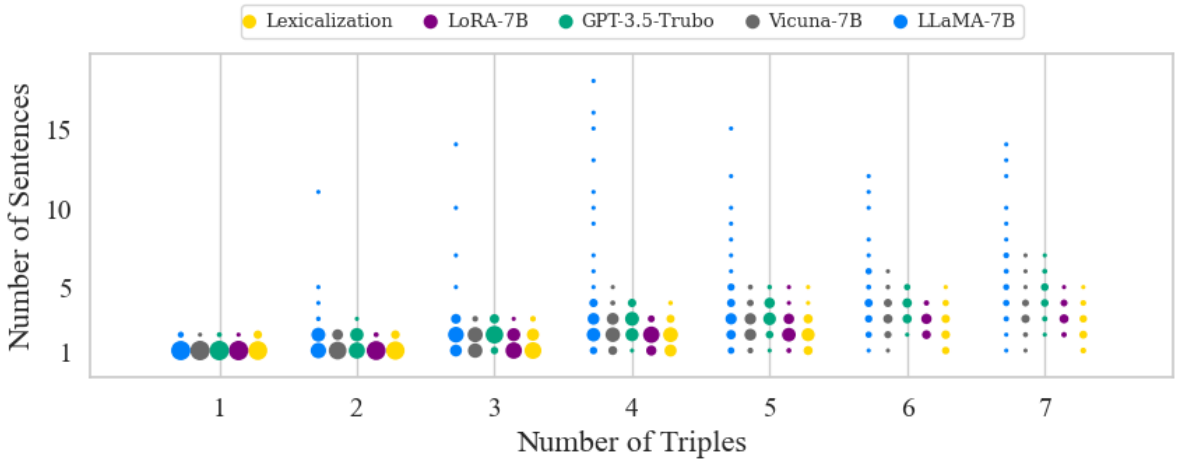


Figure 6.10: This graphic depicts the number of sentences (y-axis) that each model generates for triples of different sizes (x-axis) of the WebNLG dataset. The models are displayed in distinct colors. The size of the dots indicates the number of instances that belong to this position. Instead of utilizing the full WebNLG test data, we use a subset.

To investigate why the models accomplish various METEOR scores, we look at the number of generated sentences for each triple set size and model in Figure 6.10. This scatter plot denotes the number of triples in the given set on the x-axis. The y-axis shows the amount of generated sentences, and the dots in the graphic are colored according to the corresponding model. Once more, we only consider few-shot prompting in combination with post-processing. Furthermore, the number of

sentences belonging to the desired lexicalizations are depicted in yellow. As there are many samples for each triple category, possibly resulting in texts with varying lengths, multiple dots might exist for each model and triple set size. The size of the dots corresponds to the number of samples at this position in the plot. The figure shows that LLaMA (blue dots) is a lot more verbose compared to the other models and the gold standard. Vicuna and GPT-3.5-Turbo also generate, in multiple cases, more sentences than the lexicalization. In contrast, the number of sentences LoRA creates correlates closely with the desired result. We hypothesize that LoRA delivers the best results, according to automatic evaluation metrics, as it learned the desired structure and amount of expected sentences during fine-tuning for each of the triple set sizes.

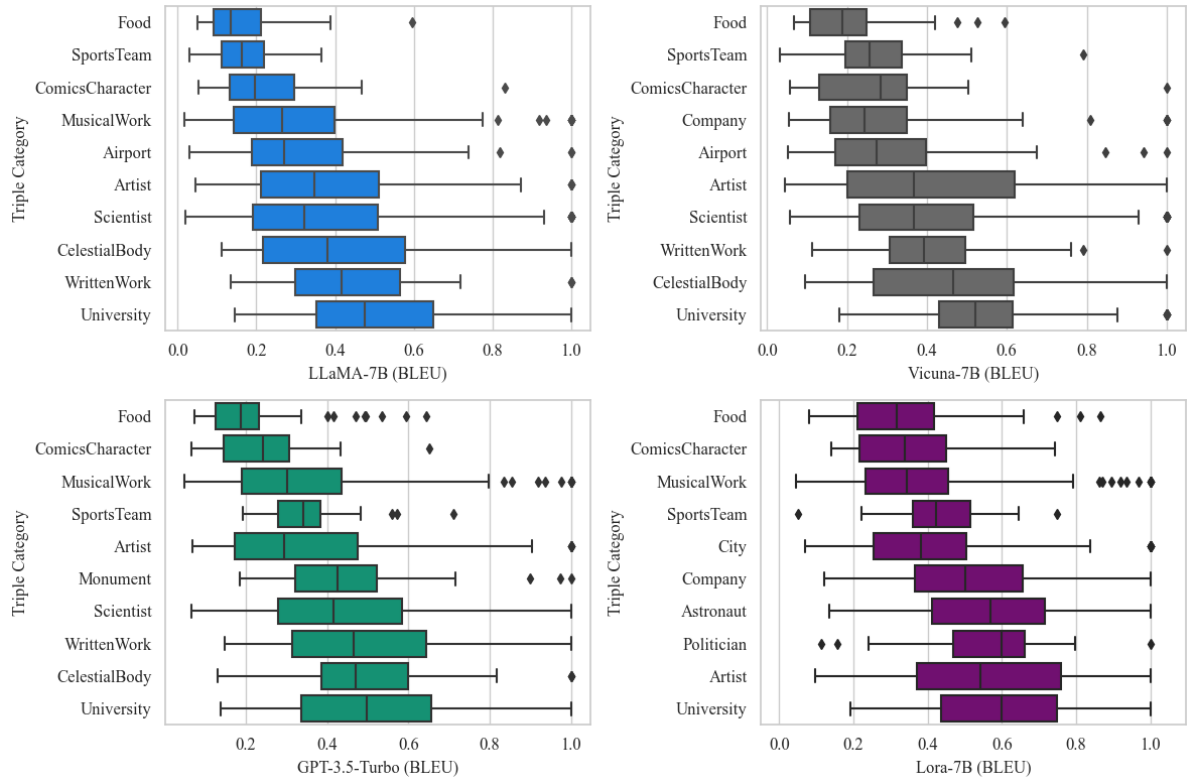


Figure 6.11: In this figure, we show the BLEU performance (x-axis) of each model for the five best and worst performing categories (y-axis). These results are calculated based on a subset of the WebNLG test set.

To determine if the topics of the 19 categories influence the results of the LLMs, we depict in Figure 6.11 the five best and worst categories for each model. Each of the four box plots provides the BLEU score of the respective model on the x-axis. On the y-axis, the topics are listed and sorted by mean performance. Since we expect an influence of the triple set sizes contained in the categories, we compare this graphic

with the median amount of triples, as shown in Figure 6.8, during the analysis. For all models, "University" is the best-performing topic. This observation is surprising; with an average number of 3.5 triples in each set, there are 11 categories with lower complexity. "CelestrialBody" and "WrittenWork" are in the top three, "Scientist" in the top four, and "Artist" in the top five for three of the four models. The topic with the lowest average number of triples (2) is "WrittenWork". While it performs well for most models, it is not contained in the top five of LoRA, similar to "CelestrialBody" and "Scientist". Looking at the five categories with the lowest BLEU score of each model, we see that "Food" is consistently ranked as the worst topic. "SportsTeam" and "ComicsCharacter" are for each model among the four categories with the lowest score, while "MusicalWork" is ranked among the four lowest topics for all models besides Vicuna. With an average of five triples per set, "Food" and "SportsTeam" are among the most complex categories. Thus, their low performance is no surprise. Out of the three topics, "MusicalWork", "Film", and "Scientist", which are not part of the training set, only "MusicalWork" belongs to the five worst performing categories of LoRA. Notably, this type is also ranked low by the other models, which were not trained on any topics. Thus, we conclude that LoRA generalizes well to unseen classes not part of the fine-tuning dataset.

To sum up, we find with this quantitative analysis that few-shot prompting performs generally better than zero-shot prompting. Furthermore, the results for some models can be improved significantly by applying rule-based post-processing to remove common issues of generations. In addition, we see zero-shot prompting to perform better for models fine-tuned on conversations compared to regular PLMs and notice further improvements with an increase in model size. According to automatic metrics, the best results can be achieved by fine-tuning a model on the specific task. This technique performs better than in-context learning utilizing a 20 times bigger model. For the task of triples-to-text generation, a major influence on the quality of the generated text is the number of related triples that need to be verbalized. Moreover, the topic of the triples might influence the generated results. Compared to the state-of-the-art approach from [97] (BLEU: 0.62, METEOR: 0.45, TER: 0.35), our similarity scores are lower. However, considering the BERT-Score, LoRA-7B performs identical with 0.96.

Human Evaluation

Next to the automatic evaluation, we also perform a human analysis to investigate common errors in model predictions and discover potential shortcomings of automatic evaluation metrics. Two researchers manually labeled 150 few-shot and 150 zero-shot predictions for this detailed investigation. The generations are sorted by lowest average BLEU and METEOR scores of all models to find the most prominent issues. These problems are classified into five common error groups inspired by the Scarecrow

paper [8]. The categories we are considering are listed in Table 6.3 together with a description for each type. For example, sometimes triples are lexicalized incorrectly, some information is missed, or inaccurate facts are added. Furthermore, the LLMs might misinterpret the prompt and generate undesired output or repeat the same information multiple times.

Table 6.3: Overview of five identified issue types with examples from generated model predictions (PRED) and human annotations (GOLD).

Issue Type	Definition	Example
Off-prompt	Prediction is unrelated to the prompt and contradicts the desired output format.	GOLD: GMA Network is a public company. PRED: The input triples are: [{'object': 'Public_company', 'property': 'type', 'subject': 'GMA_Network_(company)'}].
Inaccurate	Prediction is incomplete or factually incorrect.	GOLD: Binignit is a dessert that comes from the Philippines. PRED: Philippines is a dessert.
Mistranslated	Triples are verbalized in a language other than English.	GOLD: The Mexican based city of Ciudad Ayala in Morelos stands at 1147 metres above sea level and is in the PDT time zone. PRED: Ciudad Ayala es parte de Morelos, que es un estado en México. Ciudad Ayala tiene una elevación de 1147 metros sobre el nivel del mar [...]
Redundant	Unnecessary repetition of triple information.	GOLD: The Pontiac Rageous, a car with a coupe body style, assembled in Detroit Michigan, was first and last produced in 1997. PRED: 1997 Pontiac Rageous was produced in Detroit, Michigan, in 1997. It was a coupe. It was assembled in 1997. It was produced in 1997 [...] It was a coupe. It was assembled in 1997 [...]
Unlexicalized	Entities or relations are not lexicalized.	GOLD: The Fellowship of the Ring was followed by The Two Towers. PRED: The_Fellowship_of_the_Ring was followed by The_Two_Towers.

Based on the manual analysis, we create relative frequencies of error types, shown in Table 6.4. Values marked with "*" are cases where the considered LLM generated off-prompt results. In these circumstances, the relative frequency only considers generations being on-prompt. Otherwise, the results would be misleading as this would artificially reduce the relative proportion of the other error categories.

Looking at the overall results, LLaMA has by far the highest error frequencies for

Table 6.4: Relative frequency of issue types for zero-shot and few-shot prompts in a sample of 150 predictions with lowest averaged BLEU and METEOR scores.

Issue Type	LLaMA	Vicuna	GPT-3.5-Turbo	LoRA
	relative frequency: zero-shot / few-shot			
Off-prompt	0.65 / -	0.27 / -	- / -	- / -
Inaccurate	0.60 * / 0.61	0.41 * / 0.48	0.13 / 0.11	0.19 / 0.17
Mistranslated	- / -	0.01 * / -	- / -	- / -
Redundant	0.23 * / 0.07	0.02 * / -	0.01 / 0.01	- / 0.01
Unlexicalized	0.69 * / -	0.27 * / -	0.07 / -	- / -

* Excluding off-prompt predictions

almost all types, followed by Vicuna. For GPT-3.5 and LoRA, there is no imminent ordering. However, both have at least three times lower relative frequencies than LLaMA and two times less regarding Vicuna, with redundancy in the results of few-shot prompting being an exception.

For few-shot prompting, more than half of the results the LLaMA model produces (0.65) are not related to the desired output and hence are classified as off-prompt. Vicuna reduces this issue to 27%, while it is not present for GPT-3.5 or LoRA. Thus, we can observe that models aligned with human preferences or trained on the specific task follow a prompt's instructions more closely. This problem is completely resolved when examples of expected generations are presented to the LLMs in context, using few-shot prompting. With this technique, none of the considered models generated off-prompt outputs.

The category "Inaccurate" is one of the most common error types. It refers to imprecise generations and encompasses the utilization of erroneous numerical representations, such as a displacement of the decimal point or the substitution of entirely different digits. Other typical issues include ignoring information, adding hallucinated facts, or confusing semantic relations between triples. An example for the last problem is a changed ordering of "followedBy", or "preceededBy" properties. For LLaMA (0.60/0.61) and Vicuna (0.41/0.48), this is a lot more prominent than for GPT-3.5-Turbo (0.13/0.11) and LoRA (0.19/0.17). Zero-shot and few-shot prompting show similar behavior regarding this error type.

In rare instances, the Vicuna model creates its output in a different language than the input prompt, for which we always use English. We can only observe such generations for input triples containing many foreign words, for example, in Spanish. As we can solely perceive this phenomenon in Vicuna, this may be due to fine-tuning instructions for translation tasks.

Repeating the information of triples is mainly an issue of LLaMA. We see this redundancy in some instances where LLaMA starts regenerating the same terms

continuously until it reaches the maximum token limit. This recurrence is less of a problem for few-shot prompting and the other models in general.

The unlexicalized error category refers to generations copying entities from the triples without reformulating them into their regular word forms. This behavior is an issue for all models besides LoRA, with 0.69 for LLaMA, 0.27 for Vicuna, and 0.07 for GPT-3.5-Turbo. Providing examples for correct lexicalizations of such sequences within the samples of the few-shot prompt solves this issue effectively for all models.

When investigating model predictions manually during human evaluation, we notice, for some instances, a low similarity score according to automatic metrics. However, the generation is semantically consistent with the desired result. We identify two possible reasons for this observation. First, since the triples are arranged according to (object, property, subject), the models are nudged to start a sentence in some cases with the object. This results in generations using the passive voice (e.g. *"Death on a Factory Farm was edited by Geof Bartz."*). Most lexicalizations created by human annotators start instead with the subject of the sentence and create texts in active language (e.g. *"Geof Bartz is the editor of the film Death on a Factory Farm."*).

A second reason might be that the gold standards aggregate lots of information in a small number of sentences (e.g. *"Nurhan Atasoy was born and currently resides in Turkey, a unitary state whose area is comprised by 1.3 percent water, where he currently resides in the city of Istanbul."*). In contrast, most generations, especially the ones not from models fine-tuned on this specific task, tend to do this to a lower extent. Instead, they usually create more sentences with less complexity (e.g., *"Nurhan Atasoy resides in Istanbul. Turkey is a unitary state and the government type of Turkey is a unitary state. Nurhan Atasoy also resides in Turkey. Turkey has 1.3% of its area covered by water. Nurhan Atasoy was born in Turkey."*). This phenomenon can also be seen when looking at Figure 6.10. As this writing style can be learned better by fine-tuning on the WebNLG dataset, LoRA outperforms the other considered LLM-prompt combinations with regards to the automatic metrics while having a similar amount of issues compared to GPT-3.5-Turbo when looking at the human evaluation.

6.4 Limitations

Internal Validity Factors that might influence the outcome of an evaluation without being part of the study are, according to Wohlin et al. [103], threats to the internal validity.

One such limitation is that we run the LLMs only once on the whole evaluation set. As the generations of models are not deterministic and might produce different outputs on multiple runs, it is beneficial to execute the tests multiple times and average the results. However, we could not do so due to limited resources and time constraints. Running the evaluation sets of both tasks with all eight considered model-prompt combinations would take several working days since we only have

a single NVIDIA Tesla V100 16GB GPU available. To mitigate this issue, we make all code and scripts available that are required for other researchers to replicate our experiments.

Another issue is that we selected GPT-3.5-Turbo, a closed-source model, for our comparative analysis. Thus, we do not have complete knowledge about the characteristics of the LLM, as well as the pre- and post-processing steps employed by OpenAI. Therefore, we can not guarantee the repeatability of our results as this model or input-output processing might change without notice. We acknowledge this compromise because we cannot run models of comparable size on our own. To minimize variations of the model due to updates, we specify the exact version in the code.

For human evaluation, two annotators label typical error categories of the generations. This process might introduce bias as the labelers are involved in the project. To reduce this risk, each sample is marked based on predefined categories that include a clear description.

External Validity How generalizable the outcomes of an evaluation are to outside environments needs to be addressed when external threats to validity are considered [103].

In our work, we compare four LLMs in combination with two prompting techniques. The number of considered models and prompts could be expanded to improve generalization. Since each additional model or prompt would exponentially increase the number of combinations, and we need to adhere to resource and time constraints, further increasing this thesis's scope is impossible for us. To counteract this limitation, we publish the full prompts and code needed to replicate and extend our research. Furthermore, we use OpenAI's API and an implementation of Fastchat [28] that replicates it for open-source models. Thus, our experiments can easily be expanded to other LLMs without significant code changes. In addition, we provide detailed results of our analysis, enabling other researchers to compare their approaches with our work.

A further shortcoming is using a single dataset for each of the two considered tasks. Moreover, for semantic parsing, we only utilize a sub-selection of the test set for our evaluation. Although this limits the generalization of our results, using more data was impossible as this would increase the required GPU hours. To make the subset of SPICE representative of the entire test set, we select random samples while ensuring a similar distribution regarding number of instances per question type.

Similarly, for the human evaluation, a larger amount of samples and annotators would be necessary for generalizability. For data-to-text, we labeled samples performing worst on average to provide an upper bound for error frequencies.

As we only consider English data in our tests and the selected models are mainly trained for English, we do not expect our results to apply to other languages.

7 Conclusion & Outlook

7.1 Summary

In this section, we summarize the outcomes for each of the research questions. Furthermore, we describe shortly the methodology used to obtain the results.

RQ1: Which previous studies have investigated using LLMs for the tasks of semantic parsing and text generation?

Methodology We perform a systematic literature review for semantic parsing and data-to-text generation. We focus primarily on approaches utilizing LLMs for solving these tasks. We select relevant literature by defining search strings in addition to inclusion and exclusion criteria. Following the defined principles, we query three scientific databases and filter the results by screening the abstracts of the papers. We identify the research gaps addressed in this thesis based on the resulting research literature.

Findings The literature review results in 18 relevant publications about semantic parsing and 13 data-to-text generation papers. We recognize a lack of research evaluating different LLMs and prompting techniques for conversational semantic parsing using SPARQL. Moreover, there is no detailed analysis of model predictions investigating common error categories and their relative frequencies for this task. Regarding data-to-text generation, we found no detailed comparison of conversational LLMs and prompt configurations for triples-to-text generation. Furthermore, there is a lack of comprehensive human evaluation regarding error types and their occurrence rates for each model-prompt combination.

RQ2: What selection of Large Language Models and Prompting techniques are suitable for a comparative analysis of the considered tasks?

Methodology First, we select a set of model characteristics to consider when choosing LLMs. Then, we compare popular LLMs regarding their performance on publicly available metrics. We primarily select models that can run on our available hardware to minimize external influence in the experiments. Furthermore, we try to have some variation in the considered characteristics of the LLM. Before developing specific prompts for semantic parsing and triples-to-text generation, we use academic papers

to get an overview of the most eminent prompting techniques and best practices. Since Prompt Engineering has only recently gained popularity, we also consider grey literature to find emerging approaches. Based on these insights, we select established techniques for the comparison.

Findings We chose four models for the analysis. These LLMs include LLaMA 7B, Vicuna, and a fine-tuned LoRA model, with the latter two being fine-tuned variations of the former. We host these three models on our server to have complete control over the experiments. Additionally, we select GPT-3.5-Turbo, a large proprietary model commonly utilized for commercial solutions of the company OpenAI. Thus, we include a variety of training procedures and model sizes in our evaluation. We create one zero-shot and one few-shot prompting template for each task. The chat structure is used for all prompts, consisting of "system", "user", and "assistant" roles. The system message describes the general task, while the user and assistant messages provide examples of requests and desired responses. Furthermore, we consider best practices regarding prompt formatting and few-shot sample selection.

RQ3: How capable are the selected Large Language Models and prompting strategies for semantic parsing and triples-to-text generation based on automatic and human evaluation?

Methodology We utilize the official evaluation script of SPICE for semantic parsing and WebNLG for triples-to-text generation to calculate automatic metrics. For qualitative analysis, we perform a human evaluation, investigating common error categories of the generated content. For this purpose, we utilize 150 instances from each dataset and model-prompt combination. Two evaluators iteratively annotate these predictions to derive common issue types. Afterward, we calculate the relative frequency for each category and model-prompt configuration.

Findings We find that LoRA 7B with zero-shot prompting yields the best overall results for conversational semantic parsing on the SPICE dataset. Although this model is around 20 times smaller than GPT-3.5-Turbo, its generations are better due to being fine-tuned for the specific task. For all other models, few-shot prompting performs significantly better compared to zero-shot. The second-best model, GPT-3.5-Turbo, more than doubles its average weighted performance score from 0.28 to 0.61 by utilizing few-shot over zero-shot prompting. Regarding the human evaluation, we find that a too-low maximum token limit constraints the performance of LoRA. It prevents the model from completing complex query generations that are otherwise correct, leading to low evaluation metrics for three out of ten question types. For the most common issue categories, Syntax Error and Incorrect Query Result, LoRA shows the lowest relative frequencies, followed by GPT-3.5-Turbo. Furthermore, we analyze how closely the different model-prompt combinations follow the prompt's

instructions. LLaMA zero-shot does not generate SPARQL queries for any of the samples, but using few-shot prompting instead mitigates this behavior. Although Vicuna and GPT-3.5-Turbo follow the general task described in the prompt, both ignore some specific instructions in multiple cases.

For triples-to-text generation, a less complex task than semantic parsing, the fine-tuned LoRA model achieves the best overall results as well, but the difference between the LLMs are smaller. The quantitative analysis shows that few-shot prompting outperforms zero-shot prompting in general. Additionally, utilizing rule-based post-processing to mitigate common issues significantly increases the performance of some models. The number of related triples needing to be verbalized greatly influences the quality of the generated output. With the human evaluation, we see that some predictions receive a low score by automatic metrics, although they are semantically consistent with the desired output. We recognize two possible reasons for this phenomenon. In some cases, the models generate a natural language text in passive voice, while the gold standards use active language or vice-versa. In other cases, the lexicalizations consolidate information into a limited number of sentences, whereas most models not fine-tuned for this task do so to a lesser extent.

7.2 Future Work

This section outlines suggestions for future research to enhance conversational semantic parsing and triples-to-text generation with LLMs. First, we explore opportunities to enhance the performance of these two tasks. Afterward, we present potential advancements in the evaluation procedures.

Improving the Conversational Search System

The currently considered system components only engage in one-sided conversations. Thus, the user poses questions, and the system answers them based on information from the knowledge base and by considering the conversation history. Introducing a dedicated dialogue management component would enable the system to support users in discovering their information needs by asking clarifying questions and filtering search results based on additional information. An example of such a system is AgentGPT [104].

Research concerned with LLMs is evolving rapidly. In the last months, multiple new models were published, including LLaMA 2, the successor of LLaMA that we utilize in this thesis. These newer models could lead to a higher quality of generations due to improved pre-training methodologies and datasets. Thus, one could investigate to what extent they further enhance the performance of semantic parsing and data-to-text generation components.

We only considered two of the most established prompting techniques in this thesis. Few-shot chain-of-thought prompting or least-to-most-prompting may further enhance the models' performance. By implementing these techniques in semantic parsing, the model could learn to address sub-queries first before combining them into the final SPARQL output, thus reducing the complexity of individual steps. Likewise, in the context of triples-to-text generation, the model can be guided to verbalize each triple individually before consolidating the information into a coherent paragraph.

Extending Evaluation

Since GPT-3.5-Turbo is a closed-source model, we do not know what input and output processing is applied by OpenAI that influences our experiments. If available computational resources allow, including larger, self-hosted models in the comparison would be interesting. Using multiple LLMs of the same model family with different parameter sizes would enable us to test the effect of an increased model size while keeping everything else equal. In addition, one could fine-tune a larger version of LLaMA with the LoRA approach to see if it brings further benefits compared to the 7 Billion version.

Generalizability is inherently constrained since we rely only on one dataset for semantic parsing and another for triples-to-text creation. To mitigate this limitation, a broader range of data can be incorporated into the evaluation process. Ideally, the new samples should cover additional topics and have varying complexity. Other languages could be included as well.

To validate the reliability of the human analysis, additional annotators could assess the same samples independently. So far, only a single person labels each instance. Furthermore, one could increase the number of annotated samples for a more reliable quantification of error types.

Evaluating the entire conversational search approach end-to-end with users would be interesting. Such a case study requires creating a dedicated tool or a plugin for existing chatbots first. It would allow us to analyze the quality of results based on realistic user questions instead of curated datasets.

List of Figures

List of Tables

Acronyms

BLEU Bilingual Evaluation Understudy. 15, 17, 40, 60–63, 72

BP Brevity Penalty. 15

CQA Conversational Question Answering. 18

CS Conversational Search. 8, 9, 18

DS Dialogue Systems. 8

EM Exact Match. 38, 50, 53, 56

FN False Negative. 14

FP False Positive. 14

LLM Large Language Model. 1–8, 10–12, 19–23, 25, 26, 30–32, 34, 38–47, 56, 57, 60–62, 64–72

LoRA Low-Rank Adaptation of Large Language Models. 12, 34, 44, 45

METEOR Metric for Evaluation of Translation with Explicit ORdering. 15, 16, 40, 60, 61, 63, 72

NLG Natural Language Generation. 6, 19

NLP Natural Language Processing. 8, 10, 18, 19

NLU Natural Language Understanding. 5

P Precision. 15, 16

PLM Pre-Trained Language Model. 2, 9, 10, 19, 31–33, 43, 44, 63

PPO Proximal Policy Optimization. 13

R Recall. 15, 16

RLHF Reinforcement Learning from Human Feedback. 2, 13, 22

RM Reward Model. 13

RQ Research Question. vi, 6, 30, 33, 37, 42, 44, 47, 68, 69

TER Translation Edit Rate. 16, 40

TN True Negative. 14

TP True Positive. 14

Bibliography

[1] F. Radlinski and N. Craswell. "A Theoretical Framework for Conversational Search". In: *Proceedings of the 2017 Conference on Conference Human Information Interaction and Retrieval*. CHIIR '17. Oslo, Norway: Association for Computing Machinery, 2017, pp. 117–126. ISBN: 9781450346771. DOI: 10.1145/3020165.3020183. URL: https://doi.org/10.1145/3020165.3020183.

[2] M. Aliannejadi, L. Azzopardi, H. Zamani, E. Kanoulas, P. Thomas, and N. Craswell. "Analysing Mixed Initiatives and Search Strategies during Conversational Search". In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. CIKM '21. Virtual Event, Queensland, Australia: Association for Computing Machinery, 2021, pp. 16–26. ISBN: 9781450384469. DOI: 10.1145/3459637.3482231. URL: https://doi.org/10.1145/3459637.3482231.

[3] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[4] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, and W. Fedus. *Emergent Abilities of Large Language Models*. 2022. arXiv: 2206.07682 [cs.CL].

[5] R. Thoppilan, D. De Freitas, J. Hall, N. Shazeer, A. Kulshreshtha, H.-T. Cheng, A. Jin, T. Bos, L. Baker, Y. Du, et al. "Lamda: Language models for dialog applications". In: *arXiv:2201.08239* (2022). DOI: https://doi.org/10.48550/arXiv.2201.08239.

[6] OpenAI. *Chatgpt: Optimizing language models for dialogue*. *OpenAI*. 2022. URL: http://web.archive.org/web/20230109000707/https://openai.com/blog/chatgpt/.

[7] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, and P. Fung. "Survey of Hallucination in Natural Language Generation". In: *ACM Comput. Surv.* 55.12 (Mar. 2023). ISSN: 0360-0300. DOI: 10.1145/3571730. URL: https://doi.org/10.1145/3571730.

[8] Y. Dou, M. Forbes, R. Koncel-Kedziorski, N. A. Smith, and Y. Choi. "Is GPT-3 Text Indistinguishable from Human Text? Scarecrow: A Framework for Scrutinizing Machine Text". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 7250–7274. DOI: 10.18653/v1/2022.acl-long.501. URL: https://aclanthology.org/2022.acl-long.501.

[9] Google. *Bard announcement tweet*. Accessed: 2023-08-02. URL: https://twitter.com/Google/status/1622710355775393793.

[10] P. Brennan, K. Walbolt, and T. Schirner. *2M1207 b - First image of an exoplanet*. Accessed: 2023-08-02. URL: https://exoplanets.nasa.gov/resources/300/2m1207-b-first-image-of-an-exoplanet/.

[11] OpenAI. *ChatGPT*. Accessed: 2023-06-10. URL: https://chat.openai.com/.

[12] J. Deriu, A. Rodrigo, A. Otegi, G. Echegoyen, S. Rosset, E. Agirre, and M. Cieliebak. "Survey on evaluation methods for dialogue systems". In: *Artificial Intelligence Review* 54 (2021), pp. 755–810. URL: https://doi.org/10.1007/s10462-020-09866-x.

[13] J. Weizenbaum. "ELIZA—a Computer Program for the Study of Natural Language Communication between Man and Machine". In: *Commun. ACM* 9.1 (Jan. 1966), pp. 36–45. ISSN: 0001-0782. DOI: 10.1145/365153.365168. URL: https://doi.org/10.1145/365153.365168.

[14] S. Vakulenko. *Knowledge-based Conversational Search*. 2019. arXiv: 1912.06859 [cs.IR].

[15] P. Schneider, N. Rehtanz, K. Jokinen, and F. Matthes. "From Data to Dialogue: Leveraging the Structure of Knowledge Graphs for Conversational Exploratory Search". In: *Proceedings of the 37th Pacific Asia Conference on Language, Information and Computation*. Hong Kong, China: Association for Computational Linguistics, 2023. arXiv: 2310.05150. URL: https://arxiv.org/abs/2310.05150.

[16] P. Schneider, A. Afzal, J. Vladika, D. Braun, and F. Matthes. "Investigating Conversational Search Behavior for Domain Exploration". In: *Advances in Information Retrieval: 45th European Conference on Information Retrieval, ECIR 2023, Dublin, Ireland, April 2–6, 2023, Proceedings, Part II*. Dublin, Ireland: Springer-Verlag, 2023, pp. 608–616. ISBN: 978-3-031-28237-9. DOI: 10.1007/978-3-031-28238-6_52. URL: https://doi.org/10.1007/978-3-031-28238-6_52.

[17] F. Radlinski and N. Craswell. "A theoretical framework for conversational search". In: *Proceedings of the 2017 conference on conference human information interaction and retrieval*. 2017, pp. 117–126.

[18] H. Face. *The Hugging Face Course, 2022*. https://huggingface.co/course. [Online; accessed September, 6, 2023]. 2022.

[19] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[20] G. Wenzek, M.-A. Lachaux, A. Conneau, V. Chaudhary, F. Guzmán, A. Joulin, and E. Grave. "CCNet: Extracting high quality monolingual datasets from web crawl data". In: *arXiv preprint arXiv:1911.00359* (2019).

[21] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. "Exploring the limits of transfer learning with a unified text-to-text transformer". In: *The Journal of Machine Learning Research* 21.1 (2020), pp. 5485–5551.

[22] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL].

[23] Y. Wang, Y. Kordi, S. Mishra, A. Liu, N. A. Smith, D. Khashabi, and H. Hajishirzi. *Self-Instruct: Aligning Language Models with Self-Generated Instructions*. 2023. arXiv: 2212.10560 [cs.CL].

[24] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto. *Stanford Alpaca: An Instruction-following LLaMA model*. https://github.com/tatsu-lab/stanford_alpaca. 2023.

[25] W.-L. Chiang, Z. Li, Z. Lin, Y. Sheng, Z. Wu, H. Zhang, L. Zheng, S. Zhuang, Y. Zhuang, J. E. Gonzalez, I. Stoica, and E. P. Xing. *Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%* ChatGPT Quality*. Mar. 2023. URL: https://lmsys.org/blog/2023-03-30-vicuna/.

[26] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. *LoRA: Low-Rank Adaptation of Large Language Models*. 2021. arXiv: 2106.09685 [cs.CL].

[27] git-cloner. *Fine-tuning vicuna-7b on a single 16G GPU*. Accessed: 2023-08-10. URL: https://github.com/git-cloner/llama-lora-fine-tuning/tree/main.

[28] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. P. Xing, H. Zhang, J. E. Gonzalez, and I. Stoica. *Judging LLM-as-a-judge with MT-Bench and Chatbot Arena*. 2023. arXiv: 2306.05685 [cs.CL].

[29] OpenAI. *Model index for researchers*. Accessed: 2023-09-10. URL: https://platform.openai.com/docs/model-index-for-researchers.

[30] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. "Deep reinforcement learning from human preferences". In: *Advances in neural information processing systems* 30 (2017).

[31] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[32] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe. *Training language models to follow instructions with human feedback*. 2022. arXiv: 2203.02155 [cs.CL].

[33] OpenAI. *GPT-3.5-Turbo-0613*. Accessed: 2023-08-19. URL: https://openai.com/blog/function-calling-and-other-api-updates.

[34] A. Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. 2nd ed. O'Reilly Media, Inc., 2019.

[35] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. "Bleu: a method for automatic evaluation of machine translation". In: *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 2002, pp. 311–318.

[36] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi. "BERTScore: Evaluating Text Generation with BERT". In: *CoRR* abs/1904.09675 (2019). arXiv: 1904.09675. URL: http://arxiv.org/abs/1904.09675.

[37] S. Banerjee and A. Lavie. "METEOR: An automatic metric for MT evaluation with improved correlation with human judgments". In: *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*. 2005, pp. 65–72.

[38] M. Snover, B. Dorr, R. Schwartz, L. Micciulla, and J. Makhoul. "A Study of Translation Edit Rate with Targeted Human Annotation". In: *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas: Technical Papers*. Cambridge, Massachusetts, USA: Association for Machine Translation in the Americas, Aug. 2006, pp. 223–231. URL: https://aclanthology.org/2006.amta-papers.25.

[39] A. Kamath and R. Das. "A Survey on Semantic Parsing". In: *Automated Knowledge Base Construction (AKBC)*. 2018. DOI: https://doi.org/10.48550/arXiv.1812.00978.

[40] W. A. Woods. "Progress in natural language understanding: an application to lunar geology". In: *Proceedings of the June 4-8, 1973, national computer conference and exposition*. 1973, pp. 441–450.

[41] G. G. Hendrix, E. D. Sacerdoti, D. Sagalowicz, and J. Slocum. "Developing a natural language interface to complex data". In: *ACM Transactions on Database Systems (TODS)* 3.2 (1978), pp. 105–147.

[42] J. Krishnamurthy and T. Mitchell. "Weakly Supervised Training of Semantic Parsers". In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Jeju Island, Korea: Association for Computational Linguistics, July 2012, pp. 754–765. URL: https://aclanthology.org/D12-1069.

[43] J. M. Zelle and R. J. Mooney. "Learning to Parse Database Queries Using Inductive Logic Programming". In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*. AAAI'96. Portland, Oregon: AAAI Press, 1996, pp. 1050–1055. ISBN: 026251091X.

[44] L. S. Zettlemoyer and M. Collins. "Learning to Map Sentences to Logical Form: Structured Classification with Probabilistic Categorial Grammars". In: *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*. UAI'05. Edinburgh, Scotland: AUAI Press, 2005, pp. 658–666. ISBN: 0974903914.

[45] L. Wang, B. Qin, B. Hui, B. Li, M. Yang, B. Wang, B. Li, J. Sun, F. Huang, L. Si, and Y. Li. "Proton: Probing Schema Linking Information from Pre-Trained Language Models for Text-to-SQL Parsing". In: *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. KDD '22. Washington DC, USA: Association for Computing Machinery, 2022, pp. 1889–1898. ISBN: 9781450393850. DOI: 10.1145/3534678.3539305. URL: https://doi.org/10.1145/3534678.3539305.

[46] D. Banerjee, P. A. Nair, J. N. Kaur, R. Usbeck, and C. Biemann. "Modern Baselines for SPARQL Semantic Parsing". In: *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '22. Madrid, Spain: Association for Computing Machinery, 2022, pp. 2260–2265. ISBN: 9781450387323. DOI: 10.1145/3477495.3531841. URL: https://doi.org/10.1145/3477495.3531841.

[47] S. Rongali, L. Soldaini, E. Monti, and W. Hamza. "Don't Parse, Generate! A Sequence to Sequence Architecture for Task-Oriented Semantic Parsing". In: *Proceedings of The Web Conference 2020*. WWW '20. Taipei, Taiwan: Association for Computing Machinery, 2020, pp. 2962–2968. ISBN: 9781450370233. DOI: 10.1145/3366423.3380064. URL: https://doi.org/10.1145/3366423.3380064.

[48] Z. Lin, J. Z. Liu, and J. Shang. "Towards Collaborative Neural-Symbolic Graph Semantic Parsing via Uncertainty". In: *Findings of the Association for Computational Linguistics: ACL 2022*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 4160–4173. DOI: 10.18653/v1/2022.findings-acl.328. URL: https://aclanthology.org/2022.findings-acl.328.

[49] S. Arcadinho, D. Aparício, H. Veiga, and A. Alegria. *T5QL: Taming language models for SQL generation*. 2022. arXiv: 2209.10254 [cs.LG].

[50] Z. Li, L. Qu, and G. Haffari. "Context Dependent Semantic Parsing: A Survey". In: *Proceedings of the 28th International Conference on Computational Linguistics*. Barcelona, Spain (Online): International Committee on Computational Linguistics, Dec. 2020, pp. 2509–2521. DOI: 10.18653/v1/2020.coling-main.226. URL: https://aclanthology.org/2020.coling-main.226.

[51] A. Suhr, S. Iyer, and Y. Artzi. "Learning to Map Context-Dependent Sentences to Executable Formal Queries". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 2238–2249. DOI: 10.18653/v1/N18-1203. URL: https://aclanthology.org/N18-1203.

[52] L. Perez-Beltrachini, P. Jain, E. Monti, and M. Lapata. *Semantic Parsing for Conversational Question Answering over Knowledge Graphs*. 2023. arXiv: 2301.12217 [cs.CL].

[53] A. Saha, V. Pahuja, M. Khapra, K. Sankaranarayanan, and S. Chandar. "Complex Sequential Question Answering: Towards Learning to Converse Over Linked Question Answer Pairs with a Knowledge Graph". In: *Proceedings of the AAAI Conference on*

Artificial Intelligence 32.1 (Apr. 2018). DOI: 10.1609/aaai.v32i1.11332. URL: https://ojs.aaai.org/index.php/AAAI/article/view/11332.

[54] Y. Gu, S. Kase, M. Vanni, B. Sadler, P. Liang, X. Yan, and Y. Su. "Beyond I.I.D.: Three Levels of Generalization for Question Answering on Knowledge Bases". In: *Proceedings of the Web Conference 2021*. WWW '21. Ljubljana, Slovenia: Association for Computing Machinery, 2021, pp. 3477–3488. ISBN: 9781450383127. DOI: 10.1145/3442381.3449992. URL: https://doi.org/10.1145/3442381.3449992.

[55] E. Kacupaj, J. Plepi, K. Singh, H. Thakkar, J. Lehmann, and M. Maleshkova. "Conversational Question Answering over Knowledge Graphs with Transformer and Graph Attention Networks". In: *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. Online: Association for Computational Linguistics, Apr. 2021, pp. 850–862. DOI: 10.18653/v1/2021.eacl-main.72. URL: https://aclanthology.org/2021.eacl-main.72.

[56] K. Kukich. "Design of a knowledge-based report generator". In: *21st Annual Meeting of the Association for Computational Linguistics*. 1983, pp. 145–150.

[57] E. Reiter and R. Dale. "Building applied natural language generation systems". In: *Natural Language Engineering* 3.1 (1997), pp. 57–87.

[58] P. Cimiano, J. Lüker, D. Nagel, and C. Unger. "Exploiting ontology lexica for generating natural language texts from RDF data". In: *Proceedings of the 14th European Workshop on Natural Language Generation*. 2013, pp. 10–19.

[59] R. Lebret, D. Grangier, and M. Auli. "Neural text generation from structured data with application to the biography domain". In: *arXiv preprint arXiv:1603.07771* (2016).

[60] M. Keymanesh, A. Benton, and M. Dredze. *What Makes Data-to-Text Generation Hard for Pretrained Language Models?* 2022. arXiv: 2205.11505 [cs.CL].

[61] J. Xiang, Z. Liu, Y. Zhou, E. P. Xing, and Z. Hu. *ASDOT: Any-Shot Data-to-Text Generation with Pretrained Language Models*. 2022. arXiv: 2210.04325 [cs.CL].

[62] Z. Kasner and O. Dušek. *Neural Pipeline for Zero-Shot Data-to-Text Generation*. 2022. arXiv: 2203.16279 [cs.CL].

[63] Z. Kasner and O. Dušek. *Data-to-Text Generation with Iterative Text Editing*. 2021. arXiv: 2011.01694 [cs.CL].

[64] B. A. Kitchenham, D. Budgen, and P. Brereton. *Evidence-based software engineering and systematic reviews*. Vol. 4. CRC press, 2015.

[65] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, et al. "Retrieval-augmented generation for knowledge-intensive nlp tasks". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 9459–9474.

[66] W. Fedus, B. Zoph, and N. Shazeer. "Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity". In: *CoRR* abs/2101.03961 (2021). arXiv: 2101.03961. URL: https://arxiv.org/abs/2101.03961.

[67] OpenAI. *GPT-4 Technical Report*. 2023. arXiv: 2303.08774 [cs.CL].

[68] E. Beeching, C. Fourrier, N. Habib, S. Han, N. Lambert, N. Rajani, O. Sanseviero, L. Tunstall, and T. Wolf. *Open LLM Leaderboard*. Accessed: 2023-08-13. 2023. URL: https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard.

[69] J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le. *Finetuned Language Models Are Zero-Shot Learners*. 2022. arXiv: 2109.01652 [cs.CL].

[70] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou. *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. 2023. arXiv: 2201.11903 [cs.CL].

[71] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa. *Large Language Models are Zero-Shot Reasoners*. 2023. arXiv: 2205.11916 [cs.CL].

[72] OpenAI. *GPT best practices*. Accessed: 2023-08-14. URL: https://platform.openai.com/docs/guides/gpt-best-practices.

[73] M. Amer. *Generative AI with Cohere: Part 1 - Model Prompting*. Accessed: 2023-08-14. URL: https://txt.cohere.com/generative-ai-part-1/.

[74] J. Shieh. *Best practices for prompt engineering with OpenAI API*. Accessed: 2023-08-14. URL: https://help.openai.com/en/articles/6654000-best-practices-for-prompt-engineering-with-openai-api.

[75] M. Dubey, D. Banerjee, A. Abdelkawi, and J. Lehmann. "LC-QuAD 2.0: A Large Dataset for Complex Question Answering over Wikidata and DBpedia". In: *The Semantic Web – ISWC 2019*. Ed. by C. Ghidini, O. Hartig, M. Maleshkova, V. Svátek, I. Cruz, A. Hogan, J. Song, M. Lefrançois, and F. Gandon. Cham: Springer International Publishing, 2019, pp. 69–78. ISBN: 978-3-030-30796-7.

[76] T. Castro Ferreira, C. Gardent, N. Ilinykh, C. van der Lee, S. Mille, D. Moussallem, and A. Shimorina. "The 2020 Bilingual, Bi-Directional WebNLG+ Shared Task: Overview and Evaluation Results (WebNLG+ 2020)". In: *Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+)*. Dublin, Ireland (Virtual): Association for Computational Linguistics, Dec. 2020, pp. 55–76. URL: https://www.aclweb.org/anthology/2020.webnlg-1.7.

[77] C. Gardent, A. Shimorina, S. Narayan, and L. Perez-Beltrachini. "The WebNLG Challenge: Generating Text from RDF Data". In: *Proceedings of the 10th International Conference on Natural Language Generation*. Santiago de Compostela, Spain: Association for Computational Linguistics, Sept. 2017, pp. 124–133. DOI: 10.18653/v1/W17-3518. URL: https://aclanthology.org/W17-3518.

[78] R. Shin, C. H. Lin, S. Thomson, C. Chen, S. Roy, E. A. Platanios, A. Pauls, D. Klein, J. Eisner, and B. V. Durme. *Constrained Language Models Yield Few-Shot Semantic Parsers*. 2021. arXiv: 2104.08768 [cs.CL].

[79] R. Shin and B. V. Durme. *Few-Shot Semantic Parsing with Language Models Trained On Code*. 2022. arXiv: 2112.08696 [cs.CL].

[80] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba. *Evaluating Large Language Models Trained on Code*. 2021. arXiv: 2107.03374 [cs.LG].

[81] S. Rongali, K. Arkoudas, M. Rubino, and W. Hamza. *Training Naturalized Semantic Parsers with Very Little Data*. 2022. arXiv: 2204.14243 [cs.CL].

[82] O. Rubin, J. Herzig, and J. Berant. *Learning To Retrieve Prompts for In-Context Learning*. 2022. arXiv: 2112.08633 [cs.CL].

[83] V. Gupta, A. Shrivastava, A. Sagar, A. Aghajanyan, and D. Savenkov. *RETRONLU: Retrieval Augmented Task-Oriented Semantic Parsing*. 2021. arXiv: 2109.10410 [cs.CL].

[84] J. Yang, H. Jiang, Q. Yin, D. Zhang, B. Yin, and D. Yang. *SeqZero: Few-shot Compositional Semantic Parsing with Sequential Prompts and Zero-shot Models*. 2022. arXiv: 2205.07381 [cs.CL].

[85] L. Qiu, P. Shaw, P. Pasupat, T. Shi, J. Herzig, E. Pitler, F. Sha, and K. Toutanova. *Evaluating the Impact of Model Scale for Compositional Generalization in Semantic Parsing*. 2022. arXiv: 2205.12253 [cs.CL].

[86] N. Schucher, S. Reddy, and H. de Vries. *The Power of Prompt Tuning for Low-Resource Semantic Parsing*. 2022. arXiv: 2110.08525 [cs.CL].

[87] W. Sun, H. Khan, N. Guenon des Mesnards, M. Rubino, and K. Arkoudas. "Unfreeze with Care: Space-Efficient Fine-Tuning of Semantic Parsing Models". In: *Proceedings of the ACM Web Conference 2022*. WWW '22. Virtual Event, Lyon, France: Association for Computing Machinery, 2022, pp. 999–1007. ISBN: 9781450390965. DOI: 10.1145/3485447.3511942. URL: https://doi.org/10.1145/3485447.3511942.

[88] T. Yu, R. Zhang, A. Polozov, C. Meek, and A. H. Awadallah. "{SC}oRe: Pre-Training for Context Representation in Conversational Semantic Parsing". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=oyZxhRI2RiE.

[89] T. Y. Zhuo, Z. Li, Y. Huang, F. Shiri, W. Wang, G. Haffari, and Y.-F. Li. *On Robustness of Prompt-based Semantic Parsing with Large Pre-trained Language Model: An Empirical Study on Codex*. 2023. arXiv: 2301.12868 [cs.CL].

[90] M. Visperas, A. J. Adoptante, C. J. Borjal, M. T. Abia, J. K. Catapang, and E. Peramo. "On Modern Text-to-SQL Semantic Parsing Methodologies for Natural Language Interface to Databases: A Comparative Study". In: *2023 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*. 2023, pp. 390–396. DOI: 10.1109/ICAIIC57133.2023.10067134.

[91] Y. Lan, G. He, J. Jiang, J. Jiang, W. Xin Zhao, and J.-R. Wen. "Complex Knowledge Base Question Answering: A Survey". In: *IEEE Transactions on Knowledge and Data Engineering* (2022), pp. 1–20. DOI: 10.1109/TKDE.2022.3223858.

[92] N. Rajkumar, R. Li, and D. Bahdanau. *Evaluating the Text-to-SQL Capabilities of Large Language Models*. 2022. arXiv: 2204.00498 [cs.CL].

[93] S. Jolly, Z. X. Zhang, A. Dengel, and L. Mou. "Search and Learn: Improving Semantic Coverage for Data-to-Text Generation". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 36.10 (June 2022), pp. 10858–10866. DOI: 10.1609/aaai.v36i10.21332. URL: https://ojs.aaai.org/index.php/AAAI/article/view/21332.

[94] E. Seifossadat and H. Sameti. "Improving semantic coverage of data-to-text generation model using dynamic memory networks". In: *Natural Language Engineering* (2023), pp. 1–26.

[95] H. Gao and Z. Wei. "Neural Data-to-Text Generation Guided by Predicted Plan". In: *2022 IEEE 2nd International Conference on Information Communication and Software Engineering (ICICSE)*. 2022, pp. 53–59. DOI: 10.1109/ICICSE55337.2022.9828913.

[96] H. Gong, X. Feng, and B. Qin. "DiffuD2T: Empowering Data-to-Text Generation with Diffusion". In: *Electronics* 12.9 (2023). ISSN: 2079-9292. DOI: 10.3390/electronics12092136. URL: https://www.mdpi.com/2079-9292/12/9/2136.

[97] J. Clive, K. Cao, and M. Rei. *Control Prefixes for Parameter-Efficient Text Generation*. 2022. arXiv: 2110.08329 [cs.CL].

[98] D. Jo, T. Kwon, E.-S. Kim, and S. Kim. *Selective Token Generation for Few-shot Natural Language Generation*. 2022. arXiv: 2209.08206 [cs.CL].

[99] E. Chang, X. Shen, H.-S. Yeh, and V. Demberg. *On Training Instance Selection for Few-Shot Neural Text Generation*. 2021. arXiv: 2107.03176 [cs.CL].

[100] S. Duong, A. Lumbreras, M. Gartrell, and P. Gallinari. "Learning from Multiple Sources for Data-to-Text and Text-to-Data". In: *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*. Ed. by F. Ruiz, J. Dy, and J.-W. van de Meent. Vol. 206. Proceedings of Machine Learning Research. PMLR, Apr. 2023, pp. 3733–3753. URL: https://proceedings.mlr.press/v206/duong23a.html.

[101] I. Ampomah, J. Burton, A. Enshaei, and N. Al Moubayed. "Generating Textual Explanations for Machine Learning Models Performance: A Table-to-Text Task". In: (2022).

[102] OpenAI. *Pricing*. Accessed: 2023-09-23. URL: https://openai.com/pricing.

[103] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering*. Computer Science. Springer Berlin Heidelberg, 2012.

[104] Reworkd. *AgentGPT*. Accessed: 2023-10-13. URL: `https://github.com/reworkd/AgentGPT`.