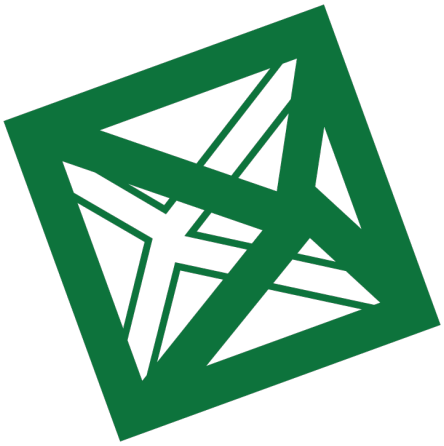
UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA

DEPARTMENT OF INFORMATICS, SYSTEMS AND COMMUNICATION



Master Degree in Data Science

Natural Language Processing

Finetuning Mistral 7B with QLoRA for KG construction

DAVIDE GIARDINI

Index

1 Introduction

Though the term "Knowledge Graph" has been used in literature for a very a long time, its concept gained traction between the public in more recent years. In particular, it was the announcement of "Google Knowledge Graph" [1] in 2012 and the following development of the same tool by major companies like Airbnb, Amazon, eBay, FaceBook, iBM, LinedIn, Microsoft, that made the concept so popular[2].

What makes Knowledge Graphs (KGs) useful and unique is how they structure and represent information: capturing entities, their attributes, and the relationships between them. Knowledge Graphs are a network of interconnected nodes and edges, where nodes represent entities (such as people, places, or concepts) and edges represent the relationships or connections between those entities. This make them, still today, a very powerful tool for representing and organizing complex information, enabling machines to understand and reason about the relationships between entities.

Knowledge Graphs find applications in a wide range of fields, including, but not limited to, search engines, recommendation systems, data governance, fraud detection and dialogue systems. They are also fundamental for decision support systems, particularly in domains such as healthcare, finance, and supply chain management.

Building a Knowledge Graph, though, is often a serious challenge. During the years, many public KGs have been developed through collaborative effort and the development of automatic extraction tools to store as much knowledge as possible. For example, Encyclopedic Knowledge Graphs, like DBpedia[3], Freebase[4] and WikiData[5], have been built to cover factual or event knowledge from different domains. Linguistic Knowledge Graphs, like WordNet[6], are instead built to store semantic relations between words, and are often used to create high-performance word embeddings. With the exception of this collaborative efforts, the vast majority of information still exists in the form of unstructured text, making it challenging to directly integrate it into Knowledge Graphs. For this reason, extracting structured knowledge from unstructured text has become a crucial task in Natural Language Processing and has garnered significant attention from both academia and industry.

Automatic Knowledge Graph Construction (AKGC) is the process of constructing Knowledge Graphs from unstructured text. Traditional approaches to Knowledge Graph construction heavily relied on rule-based methods and hand-crafted features. These methods often required extensive domain expertise and were limited in their ability to scale to large datasets. With the advent of deep learning and the advancements in NLP, the task of AKGC shifted into a multiple-step pipeline that first discovers (Named Entity Recognition) and links (Named Entity Linking) conceptual entities, resolves coreference mentions (Coreference Resolution), and finally extracts relationships among entities (Relation Extraction)[7]. Moreover, deep knowledge representation models have been developed to further enhance Knowledge Graphs. These models can refine existing graphs in several ways: they can complete incomplete or corrupt tuples within the database, discover new tuples by analyzing the internal structure and relationships within the built graph, and merge different knowledge graphs from various sources to construct new, more comprehensive databases.

This shows, though, how complex developing an Automatic Knowledge Graph Construction tool can be, as it requires to build multiple models (one for each step) and join them together in a cohesive pipeline. This complexity presents a significant challenge, effectively limiting AKGC implementation to well-resourced industries or research groups. For this reason, we believe that more research on a simpler to implement end-to-end model is required.

Recent advancements in pre-trained language models, such as BERT[8], GPT[9], and their variants, have revolutionized the field of NLP. These models, trained on massive amounts of unlabeled text data, have demonstrated remarkable performance across various downstream tasks and have demonstrated to be excellent few-shot learners. These abilities have brought the community to use them, in combination with clever prompt engineering, as KG constructors from texts[1][2]. Even though the results may not be as precise, this method enables users that do not have the resources to build an entire AKGC pipeline to extract

[1]Rahul Nayak in "Towards Data Science", Nov 10 2023, towardsdatascience.com/how-to-convert-any-text-into-a-graph-of-concepts

[2]LangChain LLMGraphTransformer documentation: langchain.com/en/latest/graph$_transformers$

triples from text in a fast and efficient way. This simpler approach gained so much traction that Neo4j[3] itself implemented LangChain's graph transformers[4] to allow their users to build graphs directly from documents.

In this project, we are going to tackle the problem of Automatic Knowledge Graph Construction by finetuning the Mistral 7B Instruct Language Model, building in this way an easy to implement AKGC tool. We are then going to evaluate its performances against a non-finetuned version of Mistral 7B Instruct, and the same non-finetuned Mistral 7B Instruct with Few Shot prompting. By comparing these three approaches, we aim to assess whether fine-tuning with synthetic data leads to improved Knowledge Graph construction capabilities. This comparison seeks to assess whether fine-tuning could lead to more cost-effective and easily implementable Knowledge Graph (KG) constructors, or if the task's complexities and Graph Databases' characteristics render it currently beyond the capabilities of Large Language Models. Additionally, we seek to identify which problems arise with the implementation of a LLM on this task, and offer insights into their potential solutions.

The rest of the report is organized as follows. In section 2 we are going to review the literature on Automatic Knowledge Graph Construction, with emphasis on Language Model applications. In section 3 we are going to delineate the problem more precisely, exposing in detail the experimental settings and limitations. In section 4 we are going to provide a detailed account of the methodology employed in creating the database for fine-tuning and testing. In section 5 we are going to delve deeper into the instrument used for fine-tuning: the LLM Mistral 7B Instruct, Lora and QLora. In section 6 we are going to describe the procedure used for the extraction of the triples from the test texts. We are then going to use the extracted triples in section 7 to evaluate the performance of our fine-tuned model against the base Mistral 7B Instruct and the same LLM with Few Shot prompting. Leveraging the insights from Section 7, we will revisit our experiment in Section 8. Finally, in section 9 we are going to provide summary of our findings and an outline of potential future research directions.

2 Past Work

Past work on a end-to-end unified framework for KG construction is incredibly scarce. Zhong et. al.[7] provide a detailed survey of Automatic Knowledge Graph Construction methods, but review past works on a per-subtask basis[5]. In the same survey, authors write that "incorporating knowledge acquisition with knowledge refinement tasks to build an integrated joint model remains a formidable bottleneck" on which still not much research has been done. The same work, though, also cites generative models, as a "potential paradigm to unify more KG-related tasks" by "turning multiple construction tasks into seq2seq generation". Ye et. al. [10] provide a comprehensive survey on the implementation of generative models for Knowledge Graph Construction, but most of the cited research is still focused on the implementation of generative models for the resolution of one specific sub-task of the AKGC pipeline, rather than the implementation of an end-to-end paradigm.

Yao et. al.[11] introduced KG-BERT: BERT for Knowledge Graph Completion. Their method takes entity and relation descriptions of a triple as input and computes scoring function of the triple, in order to predict the plausibility of a triple or a relation. To do so, they turn knowledge graph completion into a sequence classification problem, and then fine-tune a BERT model on those sequences.

Modarressi et. al.[12] developed RET-LLM: a General Read-Write Memory for Large Language Models. Their approach enables the LLM to interact with an external triple-based memory through API calls. When a user inputs a statement, the LLM extracts a triple from it and stores this information in the database for future retrieval. Their method shares strong similarities with our approach, as they also fine-tune a LLM (Alpaca-7B[13]). Though, their focus was limited to five specific relationships on which the model was fine-tuned. In contrast, we would like the model to extract triples based on whatever KG schema is provided in the prompt, without being limited to the triples he has already seen during training.

Even though they do not use any LLM, in T2KG[14] the researchers do develop an end-to-end System for creating a Knowledge Graph from unstructured text. In particular, T2KG is composed of a pipeline of

[3]https://neo4j.com/

[4]https://github.com/neo4j-labs/llm-graph-builder

[5]We are not going to delve into the past research cited in the survey as those types of models were not implemented in our work.

five components: entity mapping, coreference resolution, triple extraction, triple integration, and predicate mapping.
In Seq2KG [15] M. Stewart and W. Liu develop an end-to-end neural model for domain agnostic KG construction from text. Their approach differs from other by not relying on the existence of external knowledge bases. Their work demonstrates that neural models can perform this task on par with state-of-the-art rule-based systems.

The closest AKGC implementation to our approach remains the previously cited LangChain's LLM Graph Transformer, which employs a pipeline of cleverly designed prompts[6] to transform texts into a set of triples. However, this tool leaves to the LLM the decision of what entities and relationships to extract and how to name them, providing only some generic guidelines to encourage consistency and co-reference resolution. While this feature is undoubtedly beneficial for many use cases, we want to differentiate our tool by empowering the user with the ability of specifying the structure of the KG. This will allow the person to determine which information to extract and which to disregard. Moreover, by defining a KG structure in advance, we further reduce the possibilities of encountering inconsistent names for relationship types and nodes labels.

3 Problem Statement

Obviously, there is a reason if in literature complex pipelines are still used to face Automatic Knowledge Graph Construction from unstructured text: it is a very complex task. Each of the sub-task alone is still complex enough to require its own specific model. This work does not aim to replace the pipeline in its entirety, as we are very aware that a LLM alone would not be enough to deal with all of it. Rather, we want to evaluate the performances of a fine-tuned LLM on a controlled environment with specific limitations:

1. As we said before, the schema of the desired KG should be provided in the prompt. This simplifies the task by specifying the entities and relationships to look for. It also resolves the issue of consistency: by following a predefined schema the LLM is not in charge of defining a label for each entity or relation that needs to be consistent across multiple documents.

2. The LLM should work with one single document. The extracted triples should be consistent and follow the specified KG schema, but the model is not expected to link them to the entities discovered in other documents.

3. No node's properties will be required to be extracted from the documents, only node's labels and relationship's types.

4. We are going to deal with short text inputs, derived from a graph with a maximum of 12 nodes.

The aim of this work is not to create an end-to-end AKGC pipeline with only a finetuned LLM. What we want to do is:

1. Provide the community with an easy-to-implement, computationally inexpensive (as much as an LLM can be) tool to extract triples from a text following a predefined schema. This is done to replace the current prompt-based method with a new system that is hopefully more accurate but equally easy to implement and computationally inexpensive. Moreover, unlike most of the other tools, we aim at empowering the user with the ability of specifying the structure of the KG.

2. Evaluate the effectiveness of LLMs fine-tuning on a synthetic dataset on a restricted AKGC task, in order to assess whether more research should be developed in this direction.

3. Identifying which problems arise with the implementation of a LLM on the task of AKGC, and offer insights into their potential solutions for future research.

[6]You can view them here

4 Database Construction

One of the major challenges of this project is the construction of the training dataset with which we can fine-tune the LLM. While datasets for triples extraction from unstructured text do exist, they are often composed of single sentences[16] or based on a predefined schema like DBpedia[17]. Instead, our goal is to build a model that is able to extract a complete (though small) graph from a sequence of text, rather than sparse triples from one single sentence. Furthermore, we want to enable the user to specify a custom schema for his Knowledge Graph, and not rely on predefined schemas like those of DBpedia.

Given these considerations, we opted to create our own synthetic dataset. Our methodology involves the following steps:

1. Selecting publicly available Knowledge Graphs
2. Extracting numerous sub-graphs from these KGs
3. Transforming these sub-graphs into textual triples
4. Utilizing LLMs to generate a small text that contains all the information specified in the triples.

While constructing KGs from unstructured text is a complex task, the inverse process is relative straightforward for current state-of-the-art LLMs, as it consists simply in generating some text that contains specific information. This will allows us to create a robust synthetic dataset for our research.

4.1 KG selection and sub-graph extraction

We began by selecting a sample of publicly available Knowledge Graphs. Our selection comprised of six Neo4j KGs that were published as examples, chosen for their simplicity, high quality, and limited node labels and relationship types.

We are going to utilize the four largest databases to fine-tune our model. We are then going to use these same four DBs for a primary evaluation, in which the model's performance will be assessed using new text derived from the same KGs schemas encountered during training. Then, to evaluate the model's ability to generalize to unfamiliar schemas, we will use the two remaining databases exclusively for testing. Here is the detailed selection of our six DBs:

1. Recommendations[7]
   - Contains information about movies, their genres, the actors who performed in them, the users who rated them, and the directors who guided their production.
   - 28863 nodes and 166261 relationships
   - Used for: train and test
2. Graph-data-science[8]
   - Includes details about the kings and knights who were attackers and defenders in all the battles in the "Game of Thrones" TV series. It also establishes connections between each individual and their respective houses, outlines their familial relationships, and describes their interactions with others.
   - 2642 nodes and 16747 relationships.
   - Modifications: removed "similarity" relationship and merged all "interacts" relationships into one.
   - Used for: test only.
3. Legis-graph[9]

[7] https://github.com/neo4j-graph-examples/recommendations
[8] https://github.com/neo4j-graph-examples/graph-data-science
[9] https://github.com/neo4j-graph-examples/legis-graph

- Models the US congress as a graph. Contains all the legislators, the state they represent, their party and the bills (identified by bill Id and subject) they sponsored and voted on.
- 11824 nodes and 491173 relationships
- Used for: train and test

4. wwc-2019[10]
    - Contains information about all the teams that participated in the 2019 Woman Word Cup, their players, the match they played and the tournaments they participated in.
    - 2066 nodes and 4944 relationships.
    - Used for: test only.
5. recipes[11]
    - Incorporates numerous recipes collected from BBC blog posts, their author, their ingredients and their diet type.
    - 16075 nodes and 164880 relationships.
    - Modifications: removed keyword nodes.
    - Used for: train and test.
6. listings[12]
    - Comprises Airbnb listed apartments, their neighborhood, all of their amenities, their hosts, and the users that reviewed them.
    - 66468 nodes and 156093 relationships.
    - Modifications: linked users directly to listing and removed reviews.
    - Used for: train and test.

For each of these databases, we selected one random node and used it as the base for extracting a sub-graph. The sub-graph's size, measured in the number of nodes, was randomly chosen within a range of 6-12. Similarly, the depth of the sub-graph, defined by the number of hops it can make from the original node, was randomly selected within the range 2-6.[13] In total, 800 sub-graphs were extracted for each of the 4 datasets used for training, while 105 sub-graphs were extracted for each of the 2 dataset reserved for test.

The extracted triples were then transformed into text with the format:

```
(subject)-[predicate]->(object)
```

4.2 Text Generation

Once the triples were extracted, we passed on generating the text. To do so, we used the state-of-the-art Gemini 1.5 Pro[18] through the Gemini API[14].

The triples have been passed through the following prompt, created via prompt-engineering and trial and error:

```
Imagine being a text generator from Knowledge Graphs.
Based on the triples provided in the context, generate a short text containing all the
information contained in the triples. Make sure not to add any information of the entities
mentioned in the triples that is not coming from the knowledge graph. Even though the
usage of pronouns is allowed, make sure not to modify the names of the entities.
```

[10] https://github.com/neo4j-graph-examples/wwc2019
[11] https://neo4j.com/graphgists/dd3dedcf-c377-4575-84f4-4d0d30b2a4c5/
[12] https://neo4j.com/docs/getting-started/appendix/example-data/
[13] These ranges were manually selected by considering the size and complexity of the extracted sub-graph.
[14] https://ai.google.dev/gemini-api

```
The text you generate should not be a simple mention of all the facts stored in the
triples, but you should write them in an original way. The text should resemble
a $style.

This is the KB structure:
$KB_structure

Context:
$context
```

In this prompt, the variable `$style` has been replaced with a random element drawn from the list:

- blog article
- wikipedia article
- newspaper article
- reddit post
- YouTube script
- podcast transcript

This is done to produce text written in different forms and with different lexicon, with the aim of making the model capable of generalizing better to all the different content available on the internet.

The variable `KB_structure` is instead replaced with the general structure of the current KB in order to provide more information on what the triples of the context represent.

Lastly, the `context` variable is replaced with the triples from which the model should generate the text.

4.3 Preparing the Dataset for Finetuning

Once we had our triple-text pairs, we went on building the final dataset for fine-tuning the model. To do so, we utilized the following structure:

```
<s>[INST]Imagine being a Knowledge Graph constructor from unstructured text.
Following the schema provided, extract all the triples you can find in the text.

Schema:
$KB_structure

Context:
$Text[/INST]
Extracted Triples:
$Triples</s>
```

The `<s>` and `</s>` tags correspond to the "Begging of Sequence" and "End of Sequence" tokens used by Mistral to identify the beginning and end of a sequence of text. The `[INST]` and `[/INST]` tokens are instead specific for instruction-finetuned models and are used to identify the part of the sequence that corresponds to the instruction the model has to follow.

For each of the four DBs selected for training, we extract 600 observations for training, 100 for validation, and another 100 for testing.

5 Fine-tuning

Now that we have our dataset, we can pass on fine-tuning the model. To do so, we rented an RTX A40 NVIDIA Graphics Card with 40 GB vRAM from RunPod[15]. In the next sections, we are going to have a closer look to the LLM used for fine-tuning and the technique used.

5.1 Mistral 7B Instruct

Mistral 7B[19] is an open-source Large Language Model (LLM) developed by Mistral AI[16], featuring 7 billion parameters. For our research, we specifically employ the Instruct variant of Mistral 7B, which has been fine-tuned for instruction-following tasks. This choice has been based on several factors. First of all, we needed an open source model, since this will allow us to fine-tune it locally. Alongside Meta's LLaMA[20], Mistral 7B is one of the leading open-source LLMs currently available. Secondly, it is of a manageable size (7B parameters), this will allow us to run and fine-tune it on a single GPU. Finally, it is the open source LLM that most of the community uses for KG construction via prompt[17]. Given its widespread application in Knowledge Graph tasks, comparing our fine-tuned model to the base Mistral 7B effectively allows us to benchmark against the performance that most users would typically achieve. This comparison provides a practical and relevant assessment of our model's improvements over the standard implementation.

In order to reduce memory usage and achieve faster computation, we use the 4 bit quantized version of the model. Quantization involves converting the precision of the numbers used in the model from a higher precision (like 32-bit floating point) to a lower precision (like, in this case, 4-bit integers). This can come at the cost of a slight decrease in the model's accuracy, as the reduction in numerical precision can affect the model's ability to represent subtle differences in data, but enables us to run the model faster and on a single GPU.

5.2 Lora and QLora

To further facilitate finetuning on a single GPU, we are going to implement Parameter Efficient Fine Tuning (PEFT for short), a method for adapting pre-trained language models to specific tasks while minimizing the number of trainable parameters. By doing this, PEFT enables faster fine-tuning and lower memory requirements. In particular, we are going to use QLora[21], a variation of Lora[22].

LoRA stands for Low-Rank Adaptation, it has been introduced by Hu et al. in 2021. The core concept of Lora is derived from Aghajanyan et al. (2020)[23], who show that pre-trained language models have a low "intrinsic dimension" and can still learn efficiently despite a random projection to a smaller subspace. Hu et al. bring this idea one step further into the weight update matrices used during fine-tuning, hypothesizing that also the updates to the weights have a low "intrinsic rank" during adaptation. Therefore, instead of updating our weight matrix W by adding a full weight update matrix ΔW, we utilize matrix decomposition to decompose ΔW into the product of B and A and use them instead. In this way, the matrix ΔW, whose dimensionality is the same of the pre-trained weight matrix W_0: $d \times k$, is decomposed into the product of a matrix B of dimensions $d \times r$ and the matrix A of dimensions $r \times k$. Since the rank $r << min(d, k)$ we will have a smaller number of parameters to tune. This will ultimately result in lower computational needs..

QLoRA then goes one step further in terms of memory efficiency by utilizing 4-bit quantization on the weight values of the original network. More specifically, QLora adopts three strategies on top of LoRA:

1. Paged Optimizer: since not all optimizer states need to be in GPU memory simultaneously, they are moved between CPU and GPU memory as needed.

2. NormalFloat: assumes weights follow a normal distribution, and therefore performs quantization with buckets that are not equally spaced, but that are designed to be equally sized. In practice, buckets near

[15] runpod.io
[16] mistral.ai
[17] Rahul Nayak in "Towards Data Science", Nov 10 2023, towardsdatascience.com/how-to-convert-any-text-into-a-graph-of-concepts

0 are going to be smaller because more weights will fall into their range, while buckets towards the edges of the normal distribution are going to be wider.

3. Double Quantization: Once weights are quantized to 4-bit precision, the quantization parameters (scales and zero-points) use to quantize the weights are themselves quantized.

QLoRA, just as LoRA, has two main hyper parameters: α and r. The r parameter represents the rank of the low-rank decomposition matrices used in LoRA, while α is a scaling factor applied to the LoRA weights before they're added to the frozen weights. In our experiment we used a rank equal to 8 and an α of 16, which are fairly standard values for fine-tuning. An higher rank could potentially aid the model to understand deeper patterns, which could be beneficial in our scenario. Though, due to resource limitations, we are deferring these experiments to future works. It is also important to notice that we applied QLoRA to all the linear layers, leading to a total of 21.260.288 trainable parameters.

6 Inference

Now that we have our finetuned LLM, we can pass on generating the texts for evaluating its performances against its base version.

As we have written before, we are going to firstly utilize the four largest databases to perform a primary evaluation, in which the model's performance will be assessed using new text derived from the Knowledge Graph (KG) schemas encountered during training. We do this by asking the non-finetuned Mistral 7B Instruct to extract the triples from each of the texts. To do so, we use the the same prompt we used when we prepared our dataset for finetuning with the addition of a sentence: `Make sure to mantain the formatting of the schema: "(subject)-[PREDICATE]->(Object)"`. In this way we explicitly instruct the LLM to maintain the correct formatting. This is an important step as the evaluation will not be format-agnostic.

Then, we repeat the operation implementing the Few-Shot prompting technique. To do so, we randomly extract five examples from the training set coming from the same KG as the input observation and insert them in the prompt. In this way, after the instructions about the schema of the graph, the LLM is presented with 5 text-triple pairs coming from the same database as the text it has to process.

Lastly, the operation is repeated with our fine-tuned LLM, with the same prompts we used for fine-tuning.

These three operations are then repeated for the other 200 texts coming coming from the 2 datasets that were not used for training. The 10 remaining texts (5 for each database) were used as examples in the few-shot prompts. For these 200 texts, we were interested also in evaluating the performances of the fine tuned model with some examples in its prompt. For this reason, the prompts with examples were not only used for the base model, but also for the fine-tuned one. This approach allowed us to compare the effectiveness of the base model against the fine-tuned model, both with and without the benefit of few-shot prompting.

7 Results

We are going to evaluate the performances of our LLM with two measures: Precision and Recall.

With Precision, we are going to evaluate how many of the triples extracted by the LLMs are correct, i.e. have a correspondence in our ground truth. To compute it, we are going to divide the number of matches by the total number of triples extracted by the LLM, where the number of matches corresponds to the number of triples extracted by the LLM that are present in the ground truth. With recall, instead, we are going to evaluate how many, between the triples in our ground truth, have been extracted by the LLM. To compute it, we are going to divide the number of matches by the total number of triples in our ground truth. In essence, Precision answers the question "How many of the extracted triples are correct?", while Recall addresses "How many of the correct triples were successfully extracted?".

We are going to repeat this operation from each of the 400 + 200 texts and then compute the average precision and recall both for the 400 texts coming from the original databases and for the 200 texts comping from the 2 new databases.

To be more precise, we define a "match" between a triple extracted from the LLM and a ground-truth triple when the Levenshtein score between the two is greater than 0.95. The Levenshtein distance is a string metric for measuring the difference between two sequences. Informally, the Levenshtein distance between two words is the minimum number of single-character edits (i.e. insertions, deletions or substitutions) required to change one word into the other. This is done to make the evaluation tolerant to a minimal change in spelling, spaces, symbols or anything else that is trivial.

It is important to notice, though, that the evaluation is not agnostic to formatting: the text generated by the LLMs is going to be parsed by regex function in search for a `(subject)-[predicate]->(object)` match. We do this because we believe that formatting is a major step in AKGC. In this scenario, in fact, text generated by LLMs will have to be parsed by regex functions in order to insert the information into a graph database. If the LLM is not able to format the triples in a correct and consistent way, than its output cannot be parsed and inserted into the database. For this reason, a triple that presents the correct subject, predicate and object will not be considered a match if its formatting is wrong.

The results are presented in the following tables:

Table 1: Result for the 4 original databases

Method	Average Precision (%)	Average Recall (%)
Base Model	25.34	23.68
Few Shot	62.62	54.10
Fine Tuned	81.16	76.74

The first table testifies an evident increase in performances when passing from the base model to the few shot prompting technique, and further increase when passing to our fine tuned model. More precisely, our model achieves 81.16% average precision and 76.74% average recall. This means that, on average, 81% of the triples extracted by the LLM are true and backed up by the ground-truth, and that, on average, 77% of the original triples from which the text was generated have been extracted back by the LLM.

Table 2: Result for the 2 external databases

Method	Average Precision (%)	Average Recall (%)
Base Model	9.16	7.74
Few Shot	72.13	56.72
Fine Tuned	37.64	31.50
Fine Tuned + Few Shot	68.85	55.06

When evaluating the performances on the remaining two hundred observations the results change

substantially.

Our fine-tuned LLM still demonstrates to be an excellent zero-shot AKGC tool, and significantly improves over the performances of the base Mistral 7B Instruct. The results indicate that the fine-tuning process didn't merely teach the model information specific to the 4 Knowledge Graphs used to construct the training data. Indeed, our LLM proves to better understand the schema provided in the prompt and extract more accurate information from the text. Moreover, it does so by maintaining more consistently the correct formatting. This suggests that the model, during the fine-tuning process, acquired patterns that are beneficial for generalizing on data from unseen KG structures.

However, it's also clear that both metrics notably underperform compared to the results of the base Mistral 7B Instruct with Few Shot prompting. Additionally, while augmenting the prompt of the fine-tuned LLM with the same five examples used in the base model for the Few Shot method does significantly enhance the performance of the fine-tuned model, it still doesn't surpass the base model. In fact, the two models achieve comparable results, with our fine-tuned LLM slightly lagging behind.

The findings of this section highlighted important lessons about AKGC from unstructured text. While our fine-tuned LLM demonstrated to be an excellent Zero-shot learner, results showed that the real difference is achieved by providing examples via few-shots prompts.

These insights reflect the nature and the structure of Knowledge Graphs. Graph Databases have to maintain a consistent formatting of their entities and relationships along all the observations, to assure consistency. Deciding a schema a priori is often not enough: we have to decide how we want to name each entity in the graph. While this may sound trivial for some use cases, it is not for most of the others.

An excellent example of this is the database "wwc2019" that we used for testing. This database contains information about the Woman World Cup of 2019, their players, the match they played and the past tournaments they participated in. Let's take this triple as an example: `(Li Jie)-[PLAYED_IN]->(USA 2003)` from which Gemini correctly generated the following sentence: "Did you know that Li Jie played in the 2003 tournament held in the USA?". In this example, both the base Mistral 7B and our fine tuned model extracted the triple: `(Li Jie)-[PLAYED_IN->(FIFA Women's World Cup 2003)`. The triple itself is correct, since it is both true and follows the schema specified in the prompt: `(Person)-[PLAYED_IN]->(Torunament)`. However, since we did not specify how we wanted the tournament to be formatted, the LLM rightfully opted for a different formatting than the one from which the text was generated.

We want to underline that this is not merely an error in evaluation, but an error in the way the prompt is formulated. If we do not specify exactly how we want each name of the entities to be formatted, we make consistency between multiple extraction impossible: in the next extractions, the LLM could identify the same entity as `(World Cup 2003)`, `(USA FIFA World Cup 2003)` or any other combination. This would result in multiple nodes to be created for the same entity.

In other words, our fine-tuned model proved to be a better zero-shot learner, as it could extract more intricate relationships while maintaining accurate formatting. We do believe, though, that a zero-shot approach is only suitable for AKGC when the goal is to extract triples from a single chunk of text. If the AKGC tool were to process multiple sources, a zero-shot approach would encounter significant consistency issues. These issues can only be addressed by giving the tool specific instructions on how to format information, for instance, like in our case, using a few-shot approach.

With these new precious insights, we decided to revisit our experiment, focusing on building a fine-tuned LLM designed to receive in input not only the schema of the KG and the text from which to extract the triples but also some examples that would guide the formatting of the information it extracts. What we hope will happen is for the model to pick up during fine-tuning the same superior extraction capabilities that the zero-shot model demonstrated to have, while also learning to extract information with the correct formatting as outlined in the examples. In this way, the model will hopefully learn to better take advantage of the examples provided in the prompt in order to extract the triples from the new context.

8 Fine Tuning for Few Shot Prompting

With these new precious insights we repeated the same operations.

We started by inserting into each of the training observations three examples drawn randomly from the training data and related to the same KG structure as the text from which the LLM will have to extract the triples. In this way, each training observation followed the following schema:

```
<s>[INST]Imagine being a Knowledge Graph constructor from unstructured text.
Following the schema provided, extract all the triples you can find in the text.

Schema:
$KB_structure

Here are some examples:
Context:
$example1_text
Extracted Triples:
$examples1_triples
------------
Context:
$example3_text
Extracted Triples:
$example3_triples
------------
Context:
$example3_text
Extracted Triples:
$example3_triples
------------
Context:
$Text[/INST]
Extracted Triples:
$Triples</s>
```

We used three examples rather than five to keep the observations shorter. In fact, when trying to fine-tune the LLM with prompts containing five examples, we stumbled upon OOM errors.

We then proceeded to fine-tune the model using the same parameters as before. The only change we performed was on the maximum length parameter of our tokenizer that was increased to 3072 since the observations were now much longer. We also trained the model for less steps for multiple reasons:

1. Since the observations were lengthier, each step required more time, and we lacked the resources to execute all one thousand steps. In fact, while our initial fine-tuning lasted 3hours 51minutes for all the one thousand steps, this second fine-tuning took 5hours 38minutes just for the first 400 steps.
2. The model started to overfit around the 300th step. This is probably due to the fact that each observation now contains 4 text-triple pairs, which are then repeated randomly.
3. The first fine-tuned model showed serious problem with generalization, as it reached remarkable results when tested on new observations coming from the original four databases, but its performances dropped when presented with observations coming from new databases. For this reason, we opted to evaluate the performance of this new model at earlier subsequent stages of its training: 50, 150, and 300, since we believe that, at later steps, the models starts to learn patterns specific to the databases, rather than generic to triple extraction.

Lastly, we performed inference over the same 200 prompts we used for our models with few shot prompting, and conducted evaluation in the same way. In the following table, we compare the result of the new fine-tuned

model at steps 50, 150 and 300 with the standard Mistral 7B with Few-Shot prompting and the previously fine-tuned model with the same prompts:

Table 3: Result for the 2 external databases

Method	Average Precision (%)	Average Recall (%)
Base Model	72.13	56.71
Fine Tuned (before)	68.85	55.06
Fine Tuned (50 steps)	71.82	66.41
Fine Tuned (150 steps)	67.99	61.96
Fine Tuned (300 steps)	51.35	45.34

The results confirm what we though, as the 50 step fine-tuned model achieves better performances than later stages of fine-tuning. This shows that at later steps, the LLM starts learning patterns relative to the structure of the 4 databases from which the training observations come from, rather than gaining new information that is generalizable to the task of AKGC.

At 50 steps, our newly fine-tuned model demonstrates superior performances as an AKGC tool. Based on the same exact prompts, in fact, it is able of extracting 66.41% of the original triples from which the texts were generated, compared to the Base Model's 56.71% and the previous model's 55.06%. All of this while maintaining roughly the same precision of the Base Model, meaning that both the them correctly extract triples 72% of the time. This conclusively shows that fine-tuning did enhance the LLM capabilities as an AKGC tool, enabling it to extract deeper relations and gain more insights from the examples provided in the context.

9 Conclusions

Our project aimed to evaluate the possibility, along with the eventual challenges, of implementing Large Language Models for the task of Automatic Knowledge Graph Construction from unstructured text.

We firstly introduced the problem of AKGC and why we believe that finding a more compact and resource-requiring pipeline would benefit the entire community. We then defined a closed environment in which operating our LLM. This included stating explicit rules, such as providing the desired structure of the KG in the prompt, keeping the length of documents with which it would have to deal contained, and defining the exact information that would have to constitute its output. We developed a synthetic dataset derived from six public Neo4j graph databases. Four of these databases were used to generate the training and validation sets, with a portion of their data also set aside for the test set. The remaining two were instead reserved for testing, allowing us to asses the tool's performances not only on observations that were new, but that also derived from databases completely external from the training. We then used the training and validation sets to fine-tune a Mistral 7B Instruct LLM (with 4-bit quantization) using QLoRA on a single GPU.

Table 1 presents a comparison between the results of the base Mistral 7B Instruct model (both with a standard prompt and a prompt enhanced with five examples) and our fine-tuned model on new data generated from the same four databases used for training. In this setting, the supremacy of the fine-tuned model is striking. This proves that, when dealing with large streams of data coming over time from which we have to extract the same set of triples, fine-tuning is the optimal approach. This is the case, for the example, of the previously cited RET-LLM paper[12], in which the authors Modarressi et. al. build a conversation agent that is able to interact with an external triple-based memory. In this setting, fine-tuning provides the best results, since the LLM has to extract triples following the same KG structure it was trained on.

Table 2 compares the result of the same models on new observations generated from the two databases that were not used for training. We also reviewed the performances of the fine-tuned model with the same prompts enriched with examples. The results on this table confirm the superiority of our fine-tuned LLM as a Zero-Shot AKGC tool, which is provided with a better comprehension of the schema provided in the

prompt and ability to extract more precise information from the text, all while maintaining more consistently the correct formatting.
However, results also indicate that the real difference in performance is achieved by providing examples through few shots prompts. We attribute this to the structure of Knowledge Graphs, which requires a consistent formatting of their entities and relationships across all of the observations to ensure consistency. In simpler terms, if the AKGC tool were to process multiple sources, a zero-shot approach would encounter serious consistency issues. These problems can only be addressed by providing the tool with specific instructions on how to format information, for instance, in our case, using a few-shot approach.

Armed with these valuable insights, we decided to revisit our experiment with a focus on developing a fine-tuned Large Language Model (LLM) that would not only receive the schema of the Knowledge Graph (KG) and the text from which to extract the triples, but also some examples to guide the formatting of the information it extracts. We accomplished this by incorporating three examples from random observations into the prompts of the training set.
The outcomes of this revised approach are presented in Table 3, alongside the results previously obtained by the few shot approaches in Table 2. We evaluated the performance of the new model at steps 50, 150, and 300. The results indicate that the LLM begins to learn patterns related to the structure of the four databases from which the training observations were derived in the later steps, rather than acquiring new, generalizable information for the task of AKGC. However, at step 50, the newly fine-tuned model demonstrates its superiority as an AKGC tool, matching the average precision of the base model with the same prompts, but significantly surpassing it in terms of recall.

While the model did meet our expectations, we believe that the most significant contribution of this project lies in the valuable insights it provided. Firstly, we were able to delineate the specific and distinct use cases in which zero-shot and few-shot approaches excel. Secondly, we were able to identify specific scenarios where fine-tuning a Large Language Model for zero-shot Knowledge Graph construction would be advantageous. Lastly, the model's lack of generalization capabilities emerged more than once as a significant limitation to its performance. One way to address this in future work could be by creating a more diverse dataset. We personally chose to start with graph databases with real use and coming from a well known company to ensure quality and consistency.
While future work could attempt to expand the set of databases from which to extract the subgraphs, we are uncertain if there will be enough public databases to maintain this level of quality. However, we do believe that with a pipeline of cleverly designed prompts and a state-of-the-art LLM, it could be possible to generate a good-quality, highly diverse, entirely synthetic dataset. The pipeline could follow these steps: create multiple knowledge graph structures, generate multiple consistent triples from each structure, randomly divide the triples, generate text from each set of triples. Another way to address generalization issues could involve creating a better validation set. Due to the limited number of databases we could access, we had to use the same four databases from the training set in the validation set. With the creation of a more diverse dataset, it would be possible to create a validation set composed of observations from entirely different knowledge structures than those used for training. This would make it easier for researchers to pinpoint the steps at which the LLM begins to overfit.

Finally, it's worth noting that due to resource limitations, we were unable to experiment with varying QLoRA parameters, and thus used $\alpha = 16$ and $r = 8$, which are fairly standard values. For a more thorough investigation, future studies could explore different combinations of these two parameters, with a particular emphasis on higher ranks, as they could potentially enable the model to learn deeper concepts.

References

[1] Amit Singhal et al. "Introducing the knowledge graph: things, not strings". In: *Official google blog* 5.16 (2012), p. 3.

[2] Aidan Hogan et al. "Knowledge graphs". In: *ACM Computing Surveys (Csur)* 54.4 (2021), pp. 1–37.

[3] Sören Auer et al. "Dbpedia: A nucleus for a web of open data". In: *international semantic web conference*. Springer. 2007, pp. 722–735.

[4] Kurt Bollacker, Robert Cook, and Patrick Tufts. "Freebase: A shared database of structured general human knowledge". In: *AAAI*. Vol. 7. 2007, pp. 1962–1963.

[5] Denny Vrandečić and Markus Krötzsch. "Wikidata: a free collaborative knowledgebase". In: *Communications of the ACM* 57.10 (2014), pp. 78–85.

[6] George A Miller. "WordNet: a lexical database for English". In: *Communications of the ACM* 38.11 (1995), pp. 39–41.

[7] Lingfeng Zhong et al. "A comprehensive survey on automatic knowledge graph construction". In: *ACM Computing Surveys* 56.4 (2023), pp. 1–62.

[8] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[9] Tom Brown et al. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[10] Hongbin Ye et al. "Generative knowledge graph construction: A review". In: *arXiv preprint arXiv:2210.12714* (2022).

[11] Liang Yao, Chengsheng Mao, and Yuan Luo. "KG-BERT: BERT for knowledge graph completion". In: *arXiv preprint arXiv:1909.03193* (2019).

[12] Ali Modarressi et al. "Ret-llm: Towards a general read-write memory for large language models". In: *arXiv preprint arXiv:2305.14322* (2023).

[13] Rohan Taori et al. *Stanford Alpaca: An Instruction-following LLaMA model*. https://github.com/tatsu-lab/stanford_alpaca. 2023.

[14] Natthawut Kertkeidkachorn and Ryutaro Ichise. "T2kg: An end-to-end system for creating knowledge graph from unstructured text". In: *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*. 2017.

[15] Michael Stewart and Wei Liu. "Seq2kg: an end-to-end neural model for domain agnostic knowledge graph (not text graph) construction from text". In: *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*. Vol. 17. 1. 2020, pp. 748–757.

[16] Muhammad Salman et al. "Tiny But Mighty: A Crowdsourced Benchmark Dataset for Triple Extraction from Unstructured Text". In: *Proceedings of the 20th Joint ACL-ISO Workshop on Interoperable Semantic Annotation@ LREC-COLING 2024*. 2024, pp. 71–81.

[17] Peter Exner and Pierre Nugues. "Entity extraction: From unstructured text to DBpedia RDF triples". In: *The Web of Linked Entities Workshop (WoLE 2012)*. CEUR-WS. 2012, pp. 58–69.

[18] Gemini Team et al. "Gemini: a family of highly capable multimodal models". In: *arXiv preprint arXiv:2312.11805* (2023).

[19] Albert Q Jiang et al. "Mistral 7B". In: *arXiv preprint arXiv:2310.06825* (2023).

[20] Hugo Touvron et al. "Llama: Open and efficient foundation language models". In: *arXiv preprint arXiv:2302.13971* (2023).

[21] Tim Dettmers et al. "Qlora: Efficient finetuning of quantized llms". In: *Advances in Neural Information Processing Systems* 36 (2024).

[22] Edward J Hu et al. "Lora: Low-rank adaptation of large language models". In: *arXiv preprint arXiv:2106.09685* (2021).

[23] Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. “Intrinsic dimensionality explains the effectiveness of language model fine-tuning”. In: *arXiv preprint arXiv:2012.13255* (2020).