Judge, Generator, Executioner: Utilizing an LLM for KBC

Alex Clay[1,*], Ernesto Jiménez-Ruiz[1] and Pranava Madhyastha[1,2]

[1]City St George's, University of London, Northampton Square, London, EC1V 0HB, United Kingdom
[2]The Alan Turing Institute, British Library, 96 Euston Rd., London NW1 2DB, United Kingdom

Abstract

In this work, we introduce our submission to the 2025 LM-KBC Challenge. In keeping with the spirit of the challenge, we avoided using additional tools or information; to explore the degree to which the LLM was able to holistically handle the task of triple completion. We found that candidate filtering is crucial to refining the quality of the generated triples, as all filters we introduced improved on our baseline. This is further improved upon with the addition of more initial tail candidates per triple, which highlights the LLM's potential to act as both the generator of candidates and judge of quality in triple completion tasks.

1. Introduction

Knowledge Base (KB) completion can be costly and difficult to conduct manually [1]. As such, this has led to investigations into effective automated methods and more recently the involvement of Large Language Models (LLMs) [2]. While Automated Knowledge Base Completion (AKBC) typically involves techniques such as Retrieval Augmented Generation (RAG) by incorporating external knowledge sources like Wikidata [3], LLMs are uniquely able to utilize their parametric knowledge and to some degree complete the task in a contained manner. However, LLMs introduce a number of unique considerations, namely the quality of the generation and what information the LLMs possess within their parameters [4].

The 2025 LM-KBC Challenge[1] seeks to answer these questions through introducing a triple completion task with the following restrictions: the model must be Qwen3 [5] 8B without finetuning & RAG is prohibited. The task is to complete a given entity/relation pair with the matching tail entity (be it one, many, or none), a type of KBC. An example of this would be predicting the tail `10,000` for the entity relation pair of `Tokyo Disneyland hasCapacity`.

We propose a solution to the challenge which focuses on utilizing the LLM to supply its own supporting information in place of RAG as well as to filter candidate tail entities through an LLM as a judge process[6], which utilizes the LLM to evaluate the quality of its outputs. Through this approach we investigate the holistic use of an LLM for AKBC within the constraints of the challenge. We find that generating a higher number of candidates as well as thorough filtering improve overall efficacy. Our code is made available[2] as well as our submission on CodaLab[3].

2. Background

The LM-KBC challenge has been held 3 times since 2022 [7, 8, 9], with unique constraints and modifications made each year. As last year's competition did not restrict the use of RAG, the majority of submissions [2, 3, 10] utilized external knowledge to provide grounding in their predictions. The winner of last year's competition, Bara [2], focused on the Teller Taxonomy and fewshot prompting, abstractly

Joint proceedings of KBC-LM and LM-KBC @ ISWC 2025
*Corresponding author.
alex.clay@city.ac.uk (A. Clay); ernesto.jimenez-ruiz@city.ac.uk (E. Jiménez-Ruiz); pranava.madhyastha@city.ac.uk (P. Madhyastha)
0009-0004-2237-7412 (A. Clay); 0000-0002-9083-4599 (E. Jiménez-Ruiz); 0000-0002-4438-8161 (P. Madhyastha)

[1]https://lm-kbc.github.io/challenge2025/
[2]https://github.com/alexclay42/lmkbc
[3]Under the username aclay

experimenting with the quantity of information and details in prompts to yield the highest quality prediction.

In previous years, there has been a clear theme of both prompt engineering and finetuning. Li et al. [11] and Biester et al. [12] focused on the prompt as the fulcrum for their approaches, with Ghosh [13] in particular determining that with an effective prompt, examples are not strictly necessary.

As this year's challenge differed from past years due to additional restrictions, it was necessary to use the LLM in place of RAG, allowing for grounding without the use of external information. Similar to de Oliveira Costa Machado et al. [3] and Das et al. [14] we use relevant information accompanying each query as generated by the LLM, and instead of cosine similarity like [10], we investigated the use of the LLM to provide judgment on the quality of the subsequent triples.

3. Submission Description

3.1. Architecture

In developing the architecture for the submission, we aimed to use the LLM wherever possible to provide a full exploration of its capabilities. Therefore, the LLM provides the generation as well as the judgment of the generated candidates.

Algorithm 1 Triple Completion

1: $entity\_relation\_pairs \leftarrow dataset$
2: $candidate\_triples \leftarrow$ InitialCandidateGeneration$(entity_relation_pairs)$
3: $improved\_triples \leftarrow$ ReRun$(candidate_triples)$
4: $quantity\_checked\_triples \leftarrow$ Consensus$(improved_triples)$
5: $accepted\_triples \leftarrow$ QualityJudge$(quantity_checked_triples)$
6: **return** $accepted\_triples$

In Algorithm 1 we present the process for our approach. Firstly, the target dataset is read in and formatted into entity/relation pairs for which a tail will be predicted. These pairs are then used to generate an initial set of candidate triples in `InitialCandidateGeneration`. Next, these triples are sent to the `ReRun` function, which filters out good quality triples and re-generates the tail entity for all others. The (ideally) improved triples are then consolidated and filtered by quantity in the `Consensus` step, before finally sending any remaining triples to the `QualityJudge` which removes candidates below a certain quality.

After each LLM call throughout the entire process, we employed a post-processing function was applied to sanitize the output. This was necessary to address the model's tendency to deviate from the requested return format and include extraneous conversational text. For instance, responses were often prefaced with phrases such as, "Okay so the user is asking...", which, in earlier implementations, were erroneously accepted as valid candidate tails. This was also intended to reduce error propagation which can happen in sequential generation [15].

We explain in detail each of these steps in the following portions. Additionally, we make all prompts discussed in this portion available in Appendix A in full, which were largely adapted from HuggingFace Documentation[4].

3.1.1. InitialCandidateGeneration

For the initial generation of candidate tail entities, we adapted the prompt format presented in Veseli et al. [16], which is similar in format to that of Cohen et al. [15]. Inspired by the concept of using variations on the relation to generate differing tail entities in Cohen et al. [15], we use a temperature of .15 and increase the number of generated candidates. For those relation types where there might be multiple answers we generate 18 candidates and 15 for all others. These numbers were initially one

[4]https://huggingface.co/learn/cookbook/llm_judge

per entity/relation pair, but increased to accommodate the necessity of 2 appearances of an entity to avoid removal during consensus. The specific numbers selected were due to computational constraints encountered for batches over 3 in this function.

To create the examples provided in the prompt, we separately prompted the LLM to select the 3 most similar triples to the target entity/relation from the validation dataset. We additionally elicited the LLM to provide facts about the target entity which offers supporting information, with the idea that this could provide a replacement for grounding information typically supplied by RAG.

Following the batched generation, each LLM response was cleaned of any extraneous tokens to include only the candidate tail entity, which was formatted into a triple with the entity/relation pair for the subsequent functions.

3.1.2. ReRun

The completed triples produced by the initial run were considered to be a 'first guess' by the LLM, and therefore went through a refinement during the re-run function. The aim in introducing the re-run was to try and improve the number of candidates that would later be accepted by the judge, as in earlier versions of the implementation the judge accepted as little as 1/5th of the candidates. Additionally, by asking the LLM to essentially 'try again' it might return an answer that matched with others produced, yielding more triples appearing above the minimum required by the consensus.

In order to try and determine which candidates were 'good' and which ones needed to be re-run, we employed the technique of using the LLM as a judge[6]. To accomplish this, we prompted the LLM to give each candidate triple a score between 0 and 100, with 0 indicating it was terrible and 100 it was perfect, and any triples receiving a score of above 50 being deemed 'good'. Without access to a ground truth or external information, we relied on the LLM's parametric knowledge to inform quality.

On each turn of the re-run function, all 'good' candidates were removed and stored to continue to the next function, whereas those receiving scores of under 50 were re-run. This process continued 5 times in order to allow as many triples as possible to be approved as the number of initially accepted triples was typically quite low. We decided to run the process 5 times as we found that after the first run, the number of accepted triples reduced significantly each turn and was quite time consuming.

The re-running of the low quality triples included both LLM generated facts about the entity, such as in the initial generation, and previous attempts for that particular triple in place of examples. For example if the triple `(Honorary Knight of the Favonius, awardWonBy, Luke Skywalker)` scored under 50, then it would be re-run with information about `Honorary Knight of the Favonius` generated by the LLM and the previous attempted triple. Each time a triple was re-run any previous attempts would be included in the prompt for the generation of the new iteration. All the triples that were accepted after the 5th turn of the re-run portion continued to the consensus function.

3.1.3. Consensus

The consensus function was designed to reduce unlikely candidates, or triples that only appeared once and therefore had a lower confidence. Take for instance the following list of candidate tails for the entity/relation pair of `Liyue countryLandBordersCountry`: [Mondstadt, Mondstadt, Inazuma, Sumeru, Mondstadt, Sumeru]. The triples returned following the consensus would ideally be `(Liyue, countryLandBordersCountry, Mondstadt)` and `(Liyue, countryLandBordersCountry, Sumeru)`.

At this point additional filtration was conducted in order to reduce nonsense answers like "Okay" or repetitions of certain parts of the prompt caused by the LLM ignoring the requested format. In order to consistently consolidate the candidates to take an accurate count, we utilised the in-built python counter function, and for relation types with only one answer, we selected highest frequency rather than any that appeared twice or more.

3.1.4. QualityJudge

The QualityJudge was the last filter applied to the candidate triples. Upon completion of this portion, the triples were deemed as 'accepted' and processed into the required format for the submission file.

Using the same logic as the judge for the re-run decision, the quality judge called each triple to be evaluated three times, each with a temperature starting at .2 and increasing by .2 for each of the 3 judges. The scores from each judge were then averaged and, if over 50, the triple was accepted as with the re-run function earlier.

While averaging multiple scores improved the reliability of the judgment process, on occasion the numeric tail candidates would be returned instead of the actual score, yielding such a high number the triple would be immediately accepted. Therefore, even with additional guardrails the LLM responses could still break the process.

3.1.5. Generation and Technical Details

The Qwen3 8 billion parameter was used at all points where a call was made to an LLM, without any modifications such as fine tuning or retrieval augmented generation. The generation was handled in batches where possible, with the same generation configuration maintained across single and batched prompts.

The initial experimental runs were mostly done on the laptop of one of the authors using MPS, though the final full test dataset run was completed using free credits to access a hosted h100 GPU instance due to the high runtime of the program.

3.2. Generation Variations and Judge

The settings used at generation time can have an influence on the values returned by an LLM, as we observed in our experiments. As such, we investigated what sampling variation might be most effective for this task, with the intention of eliciting as much factual information as possible. As such, a lower temperature was selected of .15 with the intention of reducing lower confidence tokens, which ultimately performed best when followed by the judge filter, as shown in Table 1. It is possible that the reason for this is that the temperature sampling may have yielded more diverse answers than Top K, which performed best without the judge, therefore making the temperature sampling answers more effective following filtering.

Approach	F1	Precision	Recall
Greedy	.020	.210	.021
Temperature Sampling	.023	.213	.027
P Sampling	.020	.211	.021
Top K	.030	.219	.034
Greedy w/ Check	.085	.312	.207
Temperature Sampling w/ Check	.111	.350	.207
P Sampling w/ Check	.105	.319	.210
Top K w/Check	.097	.350	.209

Table 1
Differing generation techniques, using the Transformers library for generation.

Further exploration of these variations after the completion of the entire program could have proved beneficial. We speculate, that following an increase in the quantity of generated candidates, a different sampling method may have yielded higher performance. The full details of the generation configurations are available in Appendix B.

3.3. Variations

While much of the investigation into improving the proposed system focused on refining and removing bugs from the components as they were added, some initial experiments also investigated other means for improving the score on the training data. The baseline indicated in Table 2 is reflective of the program at the time the experiments were conducted, prior to a number of optimizations that were implemented later. All those detailed also contained the confidence judge following the re-run. Consensus was not implemented at this point.

Approach	F1	Precision	Recall	Empty /100
Base	.172	.610	.198	58
Re-run judge can provide updated candidate	.167	.500	.212	23
More initial candidates for multi-answer relations	.180	.465	.208	44

Table 2
Additional experiments to improve the program. Each entity/relation pair had only one candidate generated.

Throughout the implementation, it had been observed that by providing an acceptance or rejection of the candidate triple, the LLM would often return a different answer to explain why the triple was wrong. Therefore it was checked whether this could work in place of regenerating the triple on rejection to reduce LLM calls. Due to the reduction in precision, this modification was not retained.

The initial version of the program elicited a single candidate tail for each entity/triple pair, which neglected the possibility that some relations like `awardWonBy` might have multiple correct answers. Increasing the number of initial candidates also provided an opportunity to give multiple tries, some of which would be filtered out later due to poor quality. In later versions of the program this approach was adopted for single answer relations as well, with the aim of reducing the number of empty triples overall.

We discuss further experiments and ablations in our paper Clay et al. [17].

4. Evaluation

Our approach yielded a consistently high precision, with no relation dropping below .5 and peaking at .98 as shown in Table 3. However, it is worth noting that the recall was typically quite low, reaching 0 for `awardWonBy` and `hasCapacity`.

The numeric relations of `hasArea` and`hasCapacity` in particular experience very low recall. Throughout the implementation process the numeric relations typically displayed low numbers of accepted values reaching the predictions file. Despite adding specific handling to ensure that numbers were effectively extracted from LLM responses, this persisted. We determined that the number of these triples decreased significantly following the consensus function, highlighting a potential problem in the consolidation of the candidates in this step.

The non-numeric single (or few like `companyTradesAtStockExchange`) answer questions consistently had the highest recall, showing that the LLM managed to return at least one consistent answer with decent frequency. As `awardWonBy` only had 10 triples in the dataset, we take `countryLandBordersCountry` to be more representative of triples with many answers. From reviewing outputs when implementing our solution, we often saw a few correct countries appearing with high frequency as well as others which were also correct appearing very sparsely. We speculate that perhaps the LLM is more confident about certain countries sharing borders than others within its parametric knowledge.

While no particular emphasis was placed on empty values, our approach did accommodate for situations where the LLM approved entity relation "none" triples, which were later formatted to the expected empty list. However, when investigating the submitted prediction file, only one triple actually

had zero answers, indicating that the poor formatting of some of the answers (lists inside strings) may have led them to be evaluated as empty.

Table 3
Competition Submission

Relation	Macro Precision	Macro Recall	Macro F1
awardWonBy	0.7000	0.0000	0.0000
companyTradesAtStockExchange	0.8300	0.4300	0.3900
countryLandBordersCountry	0.5821	0.1493	0.1343
hasArea	0.9800	0.0300	0.0300
hasCapacity	0.8200	0.0000	0.0000
personHasCityOfDeath	0.9100	0.5800	0.5200
All Relations	**0.8386**	**0.2390**	**0.2159**
Zero-object cases	*0.2500*	*0.9000*	*0.3913*

At the point where the initial generation finishes, each triple had 15-18 possible tails. We found that very few of these triples were accepted by the 5th turn of the re-run function. This highlights a potential inability of the LLM to error correct based on previous incorrect attempts, as continuous reinforcement of wrong attempts did not lead to the acceptance of the triples. However, even more likely is that the judgment of the triples at each step was too harsh, despite setting a threshold of 50 as acceptance.

5. Issues With The Evaluation Script

Throughout the experimentation, it became evident that the evaluation script was yielding unexpected outputs. Firstly, the precision defaults to 1 for an entity/triple pair if it is not present in the prediction file, which heavily encourages the reduction of triples as a non-answer is better than an incorrect one. While this does prioritize higher quality information by encouraging answers only if they have good confidence, it can yield a surprisingly high evaluation for an empty file. To highlight this, we submitted an empty prediction file to the validation leaderboard and received the following scores: Macro precision - 1.0 Macro recall - .203 Macro F1 - .203. The recall score is presumably due to situations where the value should be empty. These scores are worth noting, particularly as they lead to a higher score than that of the lm-kbc submission of .200.

6. Discussion

We proposed an architecture for knowledge base completion utilizing the LLM as much as possible in an effort to understand its capabilities across the entire process of KB completion. While our solution ultimately showed slight improvement over the challenge baseline, we determined that using the LLM as both generator and judge may be effective if the requirements for success are lowered to permit the acceptance of more triples. Additionally, we determined that the handling of LLM responses is crucial to acquiring interpretable answers which can be correctly evaluated. In the future, utilizing specialized libraries for the data post processing, such as those handling named entity recognition, might yield more favorable outcomes and reduce the presence of nonsense answers.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] J. Chai, Y. He, H. Hashemi, B. Li, D. Parveen, R. Kondapally, W. Xu, Automatic construction of enterprise knowledge base, in: H. Adel, S. Shi (Eds.), Proceedings of the 2021 Conference on

Empirical Methods in Natural Language Processing: System Demonstrations, Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 2021, pp. 11–19. URL: https://aclanthology.org/2021.emnlp-demo.2/. doi:10.18653/v1/2021.emnlp-demo.2.

[2] D. Bara, Prompt engineering for tail prediction in domain-specific knowledge graph completion tasks, in: S. Razniewski, J. Kalo, S. Singhania, J. Z. Pan, T. Nguyen, B. Zhang (Eds.), Joint proceedings of the 2nd workshop on Knowledge Base Construction from Pre-Trained Language Models (KBC-LM 2024) and the 3rd challenge on Language Models for Knowledge Base Construction (LM-KBC 2024) co-located with the 23nd International Semantic Web Conference (ISWC 2024), Baltimore, USA, November 12, 2024, volume 3853 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024. URL: https://ceur-ws.org/Vol-3853/paper10.pdf.

[3] M. de Oliveira Costa Machado, J. M. B. Rodrigues, G. Lima, V. T. da Silva, LLM store: A KIF plugin for wikidata-based knowledge base completion via llms, in: S. Razniewski, J. Kalo, S. Singhania, J. Z. Pan, T. Nguyen, B. Zhang (Eds.), Joint proceedings of the 2nd workshop on Knowledge Base Construction from Pre-Trained Language Models (KBC-LM 2024) and the 3rd challenge on Language Models for Knowledge Base Construction (LM-KBC 2024) co-located with the 23nd International Semantic Web Conference (ISWC 2024), Baltimore, USA, November 12, 2024, volume 3853 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024. URL: https://ceur-ws.org/Vol-3853/paper11.pdf.

[4] Y. Hu, T.-P. Nguyen, S. Ghosh, S. Razniewski, Enabling LLM knowledge analysis via extensive materialization, in: W. Che, J. Nabende, E. Shutova, M. T. Pilehvar (Eds.), Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Vienna, Austria, 2025, pp. 16189–16202. URL: https://aclanthology.org/2025.acl-long.789/.

[5] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, C. Zheng, D. Liu, F. Zhou, F. Huang, F. Hu, H. Ge, H. Wei, H. Lin, J. Tang, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Zhou, J. Lin, K. Dang, K. Bao, K. Yang, L. Yu, L. Deng, M. Li, M. Xue, M. Li, P. Zhang, P. Wang, Q. Zhu, R. Men, R. Gao, S. Liu, S. Luo, T. Li, T. Tang, W. Yin, X. Ren, X. Wang, X. Zhang, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Zhang, Y. Wan, Y. Liu, Z. Wang, Z. Cui, Z. Zhang, Z. Zhou, Z. Qiu, Qwen3 technical report, 2025. URL: https://arxiv.org/abs/2505.09388. arXiv:2505.09388.

[6] R. Hu, Y. Cheng, L. Meng, J. Xia, Y. Zong, X. Shi, W. Lin, Training an llm-as-a-judge model: Pipeline, insights, and practical lessons, in: Companion Proceedings of the ACM on Web Conference 2025, WWW '25, ACM, 2025, p. 228–237. URL: http://dx.doi.org/10.1145/3701716.3715265. doi:10.1145/3701716.3715265.

[7] S. Singhania, T. Nguyen, S. Razniewski (Eds.), Proceedings of the Semantic Web Challenge on Knowledge Base Construction from Pre-trained Language Models 2022 co-located with the 21st International Semantic Web Conference (ISWC2022), Virtual Event, Hanghzou, China, October 2022, volume 3274 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2022. URL: https://ceur-ws.org/Vol-3274.

[8] S. Razniewski, J. Kalo, S. Singhania, J. Z. Pan (Eds.), Joint proceedings of the 1st workshop on Knowledge Base Construction from Pre-Trained Language Models (KBC-LM) and the 2nd challenge on Language Models for Knowledge Base Construction (LM-KBC) co-located with the 22nd International Semantic Web Conference (ISWC 2023), Athens, Greece, November 6, 2023, volume 3577 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023. URL: https://ceur-ws.org/Vol-3577.

[9] S. Razniewski, J. Kalo, S. Singhania, J. Z. Pan, T. Nguyen, B. Zhang (Eds.), Joint proceedings of the 2nd workshop on Knowledge Base Construction from Pre-Trained Language Models (KBC-LM 2024) and the 3rd challenge on Language Models for Knowledge Base Construction (LM-KBC 2024) co-located with the 23nd International Semantic Web Conference (ISWC 2024), Baltimore, USA, November 12, 2024, volume 3853 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024. URL: https://ceur-ws.org/Vol-3853.

[10] T. Prabhong, N. Kertkeidkachorn, A. Trongratsameethong, KGC-RAG: knowledge graph construction from large language model using retrieval-augmented generation, in: S. Razniewski, J. Kalo, S. Singhania, J. Z. Pan, T. Nguyen, B. Zhang (Eds.), Joint proceedings of the 2nd work-

shop on Knowledge Base Construction from Pre-Trained Language Models (KBC-LM 2024) and the 3rd challenge on Language Models for Knowledge Base Construction (LM-KBC 2024) co-located with the 23nd International Semantic Web Conference (ISWC 2024), Baltimore, USA, November 12, 2024, volume 3853 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024. URL: https://ceur-ws.org/Vol-3853/paper13.pdf.

[11] X. Li, A. J. Hughes, M. Llugiqi, F. Polat, P. Groth, F. J. Ekaputra, Knowledge-centric prompt composition for knowledge base construction from pre-trained language models, in: S. Razniewski, J. Kalo, S. Singhania, J. Z. Pan (Eds.), Joint proceedings of the 1st workshop on Knowledge Base Construction from Pre-Trained Language Models (KBC-LM) and the 2nd challenge on Language Models for Knowledge Base Construction (LM-KBC) co-located with the 22nd International Semantic Web Conference (ISWC 2023), Athens, Greece, November 6, 2023, volume 3577 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023. URL: https://ceur-ws.org/Vol-3577/paper3.pdf.

[12] F. Biester, D. D. Gaudio, M. Abdelaal, Enhancing knowledge base construction from pre-trained language models using prompt ensembles, in: S. Razniewski, J. Kalo, S. Singhania, J. Z. Pan (Eds.), Joint proceedings of the 1st workshop on Knowledge Base Construction from Pre-Trained Language Models (KBC-LM) and the 2nd challenge on Language Models for Knowledge Base Construction (LM-KBC) co-located with the 22nd International Semantic Web Conference (ISWC 2023), Athens, Greece, November 6, 2023, volume 3577 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023. URL: https://ceur-ws.org/Vol-3577/paper4.pdf.

[13] S. Ghosh, Limits of zero-shot probing on object prediction, in: S. Razniewski, J. Kalo, S. Singhania, J. Z. Pan (Eds.), Joint proceedings of the 1st workshop on Knowledge Base Construction from Pre-Trained Language Models (KBC-LM) and the 2nd challenge on Language Models for Knowledge Base Construction (LM-KBC) co-located with the 22nd International Semantic Web Conference (ISWC 2023), Athens, Greece, November 6, 2023, volume 3577 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023. URL: https://ceur-ws.org/Vol-3577/paper0.pdf.

[14] A. Das, N. Fathallah, N. Obretincheva, Navigating nulls, numbers and numerous entities: Robust knowledge base construction from large language models, in: S. Razniewski, J. Kalo, S. Singhania, J. Z. Pan, T. Nguyen, B. Zhang (Eds.), Joint proceedings of the 2nd workshop on Knowledge Base Construction from Pre-Trained Language Models (KBC-LM 2024) and the 3rd challenge on Language Models for Knowledge Base Construction (LM-KBC 2024) co-located with the 23nd International Semantic Web Conference (ISWC 2024), Baltimore, USA, November 12, 2024, volume 3853 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024. URL: https://ceur-ws.org/Vol-3853/paper12.pdf.

[15] R. Cohen, M. Geva, J. Berant, A. Globerson, Crawling the internal knowledge-base of language models, in: A. Vlachos, I. Augenstein (Eds.), Findings of the Association for Computational Linguistics: EACL 2023, Association for Computational Linguistics, Dubrovnik, Croatia, 2023, pp. 1856–1869. URL: https://aclanthology.org/2023.findings-eacl.139/. doi:10.18653/v1/2023.findings-eacl.139.

[16] B. Veseli, S. Razniewski, J.-C. Kalo, G. Weikum, Evaluating the knowledge base completion potential of GPT, in: H. Bouamor, J. Pino, K. Bali (Eds.), Findings of the Association for Computational Linguistics: EMNLP 2023, Association for Computational Linguistics, Singapore, 2023, pp. 6432–6443. URL: https://aclanthology.org/2023.findings-emnlp.426/. doi:10.18653/v1/2023.findings-emnlp.426.

[17] A. Clay, E. Jiménez-Ruiz, P. Madhyastha, Noise or nuance: An investigation into useful information and filtering for llm driven akbc, 2025. URL: https://arxiv.org/abs/2509.08903. arXiv:2509.08903.

A. Prompts

A.1. Background Info

```
"State facts about {entity}."
```

A.2. Custom Examples

""" You will be given a list of knowledge graph triples and a target triple.
Your task is to provide the 3 triples from the given list that are most similar to the target.
Give your answer as the 3 numbers representing the most similar triples to the target triple.
Provide your answer as follows:
Answer:::
3 Similar Triples: (your answer, the numbers associated with the most similar 3 triples to the candidate)
You MUST provide values for '3 Similar Triples:' in your answer.
Now here is the list of triples: list
And here is the target triple: entity rel
Provide your answer. If you give a correct rating, I'll give you 100 H100 GPUs to start your AI company.
Answer:::
3 Similar Triples: """
The line offering H100 GPUs was retained from the documentation, as it was noted to have shown an improvement in performance.

A.3. Initial Generation

```
"""{info}\n""" + examples + """\nQ: {entity} # {rel}\nA:"""
```

The examples were formatted as above with Q: entity relationship A: answer 1 answer 2 ...

A.4. Re-run Judge

""" You will be given a knowledge graph triple.
Your task is to provide a 'total rating' scoring how confident you are that the triple is accurate.
Give your answer on a scale of 0 to 100, where 0 means that you believe the triple is inaccurate, and 100 means that the triple is absolutely accurate.
Here is the scale you should use to build your answer:
0: The triple is terrible: the tail is irrelevant or incorrect given the relation
25: The triple is probably not correct: the tail might make sense given the relation but the information is wrong
50: The triple is probably correct: the tail makes sense given the relation and the information might be right
100: The triple is excellent: the tail makes sense given the relation and the information is correct
Provide your feedback as follows:
Feedback:::
Total rating: (your rating, as a number between 0 and 100)
You MUST provide values for 'Total rating:' in your answer.
Now here is the triple. entity rel tail
Provide your feedback. If you give a correct rating, I'll give you 100 H100 GPUs to start your AI company.
Feedback:::
Total rating: """

A.5. Re-run

```
"""{info}
{prev}
CORRECT: {entity} {rel} # """
```

prev indicates the previous versions of that triple which were not accepted, each preceded by INCORRECT as a delineation. info is the background information for the entity, generated the same as in the initial generation.

A.6. Judge

This prompt was the same as that used in the Re-run Judge, the variation was that it was run 3x with different temperatures and the score then averaged.

A.7. LLM Cleanup

""" You will be given a text.
Your task is to extract any type from the text. Give your answer in a python formatted list.
Here are examples of how to answer:
Input: "The Netherlands is a country in Northern Europe, situated between the North. Denmark is a"
Output: ["Netherlands", "Denmark"]
Input: ", which is located in the city of Toronto. The stadium has a seating capacity of 16,000 people. It's a relatively new facility, opened in 2009. The arena is known for its excellent acoust"
Output: ["16,000", "2009"]
Provide answer as follows:
Answer:::
Output: (your response, as a list of values)
You MUST provide values for 'Output:' in your answer.
Now here is the text.
Input: input
Answer:::
Output: """

Based on the relation type, the 'type' variable would change to indicate the expected output. The input examples are actual outputs produced by the LLM during the implementation. Additionally, the LLM cleanup acts as a fallback should the answer not be extracted by a regex either for numbers or proper nouns.

B. Generation Details

Table 4
Generation technique variations

Technique	do_sample	top_k	top_p	temperature
greedy	False	-	-	-
top_k	True	10	-	-
temp_sampling	True	0	-	0.15
p_sampling	True	0	0.7	-

Table 5
Details for the different generation configurations.

The token count was held at 15 for all calls to the LLM.