ULTRAG: A UNIVERSAL SIMPLE SCALABLE RECIPE FOR KNOWLEDGE GRAPH RAG

Dobrik Georgiev, Kheeran K. Naidu, Alberto Cattaneo, Federico Monti, Carlo Luschi, Daniel Justus
Graphcore Research
{dobrikg,kheerann,albertoc,federicom,carlo,danielj}@graphcore.ai

ABSTRACT

Large language models (LLMs) frequently generate confident yet factually incorrect content when used for language generation (a phenomenon often known as *hallucination*). Retrieval augmented generation (RAG) tries to reduce factual errors by identifying information in a knowledge corpus and putting it in the context window of the model. While this approach is well-established for document-structured data, it is non-trivial to adapt it for Knowledge Graphs (KGs), especially for queries that require multi-node/multi-hop reasoning on graphs. We introduce ULTRAG, a general framework for retrieving information from Knowledge Graphs that shifts away from classical RAG. By endowing LLMs with off-the-shelf *neural* query executing modules, we highlight how readily available language models can achieve state-of-the-art results on Knowledge Graph Question Answering (KGQA) tasks *without any retraining of the LLM or executor involved*. In our experiments, ULTRAG achieves better performance when compared to state-of-the-art KG-RAG solutions, and it enables language models to interface with Wikidata-scale graphs (116M entities, 1.6B relations) at comparable or lower costs.

1 INTRODUCTION

Large language models (LLMs) have rapidly evolved to become a key source of information and a valuable tool for natural language tasks. However, despite impressive linguistic capabilities that emerged by scaling the size of models and training datasets, LLMs remain notoriously unreliable when it comes to their factual correctness. In particular, a frequently observed failure mode is the overconfident generation of plausible, yet fabricated content that is unsupported or even contradicted by facts (Huang et al., 2025a).

Mitigating these so-called *hallucinations* is a central challenge for LLM research, potentially unlocking a wide range of applications that strongly rely on trustworthy language models. Retrieval-augmented generation (RAG) has emerged as a central strategy for grounding LLMs in external knowledge sources by identifying and retrieving information relevant to a query from an unstructured corpus of documents (Lewis et al., 2020; Borgeaud et al., 2022; Izacard et al., 2023). However, not all real-world information is organized as documents. Knowledge Graphs (KGs) such as Wikidata (Vrandečić & Krötzsch, 2014) or domain-specific KGs, often containing proprietary enterprise data, represent a huge source of reliable, interpretable, and easily updatable information, stored in a highly efficient way as (subject, predicate, object) triples. Unfortunately, translating the key concepts of RAG to graph-structure data is not an obvious task, as relevant facts are often distributed across multiple entities and relations within the graph, which need to be considered together to provide a complete picture of the available knowledge. While some approaches for using KGs to augment LLMs have been developed Mavromatis & Karypis (2025); Li et al. (2025), they often fail for more complex queries that require deeper understanding of logic (Cattaneo et al., 2025), and at the same time they do not scale well to large KGs with hundreds of millions / billions of entities and relations.

Main Contributions We present ULTRAG, a general framework for information retrieval from KGs that can be applied efficiently and effectively to arbitrary *web-scale graphs* and can be implemented with off-the-shelf components *with no retraining* of the modules involved. The key idea at the base of ULTRAG is that a successful LLM-based Knowledge Graph Question Answering (KGQA) system

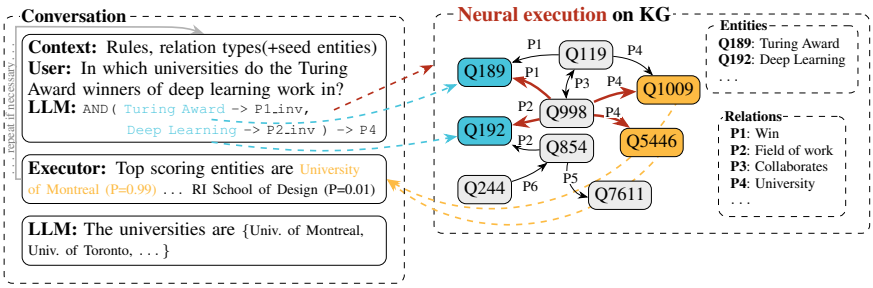

Figure 1: ULTRAG pipeline. The LLM is provided with the syntactic rules for queries and the relation types. Ground-truth seed entities (Turing Award, Deep Learning, ...) may be given, if not, an entity linking step takes place. The generated query is then neurally executed against the knowledge graph. The most likely query answers are fed back to the LLM, which produces a final answer set.

needs to be robust to *both* LLM and KG imperfections (Section 3). Motivated by this intuition, we argue for the need of using *neural* query-execution modules for interfacing with real-world KGs. In our experiments, we show that such executors achieve $\sim$16% improvement on average over symbolic ones when used as a tool by LLMs ceteris paribus. In addition, our version of ULTRAG implemented with foundational query execution modules (ULTRAG-OTS) achieves zero-shot state-of-the art results on various inductive KGQA benchmarks (even when compared to transductive approaches finetuned on the specific datasets), and it is able to effectively scale to KGs the size of Wikidata (116 million entities and 1.6 billions triples). To the best of our knowledge, ULTRAG is the first framework that is able to successfully combine LLMs with query execution modules for KGQA systems, and in doing so it highlights a research direction that appears underexplored in the literature.

Related work For graph-structured knowledge sources, prior RAG approaches can be generally grouped into four main categories: KG agent-based approaches, path-based approaches, graph neural network (GNN)-based approaches, and query-based approaches. KG agents (Sun et al., 2024; Chen et al., 2024) are LLMs that are trained to reason over KGs, starting from a given seed entity and exploring the KG step by step until a likely answer is found. Path-based approaches (Luo et al., 2024; 2025a) first retrieve a set of relevant paths (a chain of relations) between the *seed entities* (entities mentioned in the question) and other entities, and then use an LLM to reason over these paths to select the most likely answer. GNN-based approaches have been relatively sporadic in the literature so far, with Mavromatis & Karypis (2025); Mavromatis et al. (2025) being among the most prominent ones. These methods compute node/relation embeddings and use them to compute a score for each entity to be the answer. Similarly, query-based approaches, which translate natural questions into structured executable queries, have received little attention in the literature to date, largely due to relatively poor performance (Das et al., 2021; Yu et al., 2022; Mavromatis & Karypis, 2025). While not falling squarely in any of the previous categories, we should also highlight that there are approaches, like SubgraphRAG (Li et al., 2025), that fuse together ideas from more than one of the above groups.

Analogously to Li et al. (2025), our approach can be classified as a hybrid solution for KGQA systems, since (as we shall see) it combines the use of LLMs (for generating queries and reasoning over retrived information) and GNNs (for running queries over KGs) to produce the desired answers. Yet, it differs from prior art by explicitly requiring the GNN to be aligned with the behaviour of a symbolic query executor. The idea of aligning (graph) neural networks with algorithms (a.k.a. neural algorithmic reasoning; NAR) has first been proposed by Veličković et al. (2020), with further theoretical foundations developed by Xu et al. (2021; 2020). While there has been success in applications of this class of models (Deac et al., 2021; Yu et al., 2023; Numeroso et al., 2024), their use as a LLM tool (this work) has never been explored to the best of our knowledge. Notably, if viewed through the lens of NAR, this work corresponds with the largest-scale application of NAR to date, with the second being Numeroso et al. (2024) (graphs up to 6M edges).

2 BACKGROUND

Knowledge Graphs A knowledge graph $\mathcal{G} = (\mathcal{V}, \mathcal{R}, \mathcal{E})$ is a database represented as entities $\mathcal{V}$, a finite number of relation types $\mathcal{R}$, and relations between entities $\mathcal{E} \subset \mathcal{V} \times \mathcal{R} \times \mathcal{V}$. Relation triplets $(h, r, t) \in \mathcal{E}$ consist of a head h, a tail t and relation type r. Usually, $|\mathcal{R}| \lll |\mathcal{E}|$. Knowledge graphs

are often *incomplete* (Paulheim, 2016) and *contain redundant information* (Akrami et al., 2020), mainly as a result of automated construction (Zhong et al., 2023).

From Link Prediction to Relation Projection Due to the incompleteness of knowledge graphs (i.e. missing relations between entities), there has been growing interest in learnable solutions for solving link prediction problems (Bordes et al., 2013; Yang et al., 2015; Trouillon et al., 2016; Dettmers et al., 2018). Graph Neural Networks (GNNs) appeared in particular as a prominent class of approaches for *inductive* KG-completion tasks, thanks to their ability to generalize over unseen entities and connectivity patterns (Schlichtkrull et al., 2018; Zhang et al., 2020; Zhu et al., 2021; 2023). In the domain of graph-based machine learning, the standard recipe for link prediction uses the embeddings of the edge (relation) $\boldsymbol{h}_r$, and its two endpoints $\boldsymbol{h}_u$ and $\boldsymbol{h}_v$ (Bronstein et al., 2021), to predict the existence of a link. However, in the presence of symmetries in the connectivity of knowledge graphs, this recipe can underperform without the use of labelling tricks. As shown by Zhang et al. (2021), when predicting whether (u, r, v) exists, labelling u and v differently from the remaining part of $\mathcal{V}$ is theoretically optimal. Unfortunately, this also comes at high computational cost, as node embeddings need to be recomputed for each possible link one might want to predict. To address this limitation, ID-GNN (You et al., 2021) reformulates the task, conditioning node embeddings *only* on source node u (rather than on candidate link (u, r, v)), and then predicting the probability for *all* $v \in \mathcal{V}$ to be the tail of an r relation. Thanks to its reduced complexity, this formulation gained popularity in recent years, and it has been adopted by models such as NBFNet Zhu et al. (2021), which often serves as a baseline for inductive link prediction tasks.

Foundation models for knowledge graphs While in the previous paragraph we briefly discussed methodologies capable of generalizing over unseen nodes and connectivity patterns, an ideal link predictor should also be able to process KGs built with previously unseen relation types. In this direction, Galkin et al. (2024b) recently introduced ULTRA, a foundation model for knowledge graph completion, which generalizes across KGs with different relation vocabularies by making relation type embeddings a function of their relative interactions. To achieve this, ULTRA builds an auxiliary graph $\mathcal{G}_{\mathcal{R}} = (\mathcal{R}, \mathcal{R}_{fund}, \mathcal{E}_{\mathcal{R}})$, with relation types $\mathcal{R}$ as nodes, relation interactions as edges $\mathcal{E}_{\mathcal{R}}$ and four fundamental relation interaction types $\mathcal{R}_{fund} = \{h2h, t2t, h2t, t2h\}$. The relation interaction types describe the four possible ways in which two relation types can interact with each other (i.e. two relation types can share same head, can share same tail, the head of one can be the tail of the other, and the tail of one can be the head of the other). Whenever a relation projection $(u, r, ?)$ (i.e. a link prediction task) is performed on $\mathcal{G}$, relation type embeddings are computed by applying a GNN on $\mathcal{G}_{\mathcal{R}}$. Tail probabilities for projection $(u, r, ?)$ are then inferred by initializing relations representations (and the embedding of source node u) on $\mathcal{G}$ with the output of said GNN, and applying an inductive link predictor such as the ones described in the previous paragraph. For further details, we refer the reader to Galkin et al. (2024b).

Foundation models for logical query reasoning Beyond knowledge graph completion, another relevant task on KGs is complex logical query answering (CLQA). In short, provided a compositional query built by several projections and logical operators (i.e. written in first-order logic), the goal is to retrieve the set of entities that verify the query based on the information contained in the knowledge base. In the absence of noise, a symbolic executor would always be able to infer the correct entities that verify the query simply by "walking" over the edges of the provided graph. However, in real-world scenarios, things are not as easy. Due to the incompleteness of the knowledge base, some relevant connections might indeed be missing, and as a result one might need to resort to link prediction modules to infer the likelihood of a node being part of the solution set. Along these lines and building on top of ULTRA, Galkin et al. introduced ULTRAQUERY (Galkin et al., 2024a), a foundational logical query answering system operating with fuzzy sets[1]. Starting from the leaves (e.g. `Turing award` in Figure 1), ULTRAQUERY builds the answer upwards: it either projects an intermediate result with a relation (e.g. $(y, \texttt{University}, ?)$) using ULTRA's link prediction capabilities, or uses fuzzy logic to combine previous projection results (at each intermediate step, the likelihood of a node satisfying a portion of the query can thus be seen as the membership function of a fuzzy set). While the architecture used for ULTRA remains the same, Galkin et al. retrain its weights to make the model able to deal with non-leaf relation projections, where the input is a generic membership function over the node set, rather than a Kronecker's delta centered over a source node.

[1]A fuzzy set $\mathcal{S}$ is a generalization of a set where each element u has "membership" $\mu_{\mathcal{S}}(u) \in [0; 1]$ to $\mathcal{S}$.

Algorithm 1 ULTRAG Recipe

Require: Question $q \in \mathcal{T}$, Knowledge Graph $\mathcal{G} = (\mathcal{V}, \mathcal{R}, \mathcal{E})$, Neural Query Executor $\mathcal{X} : 2^{\mathcal{F}} \times \Phi \times \mathcal{G} \to \mathcal{F}$, Entity Linker $\mathcal{L} : \Phi \times 2^{\mathcal{F}} \to 2^{\mathcal{F}}$, Decider $\mathcal{D} : \mathcal{F} \times \mathcal{T} \to \{\text{True}, \text{False}\}$, Arbitrator $\mathcal{A} : \mathcal{F} \times \Phi \times \mathcal{T} \to 2^{\mathcal{V}}$, (optional) seed entities $\mathcal{S} \subseteq \mathcal{V}$

Ensure: Answer set $\mathcal{A} \subseteq \mathcal{V}$

$\mathcal{P} \leftarrow \begin{cases} \{\boldsymbol{x}_s : s \in \mathcal{S}\} & \text{if } \mathcal{S} \text{ provided} \\ \emptyset & \text{otherwise} \end{cases}$
sufficient $\leftarrow$ False
while $\neg$*sufficient* **do**
  $\varphi \leftarrow \text{LLM}(q, \mathcal{R}, \mathcal{P})$
  $\mathcal{I} \leftarrow \mathcal{L}(\varphi, \mathcal{P})$
  $\boldsymbol{x} \leftarrow \mathcal{X}(\mathcal{I}, \varphi, \mathcal{G})$
  sufficient $\leftarrow \mathcal{D}(\boldsymbol{x}, q)$
  $\mathcal{P} \leftarrow \mathcal{P} \cup \{\boldsymbol{x}\}$
end while
return $\mathcal{A}(\boldsymbol{x}, \varphi, q)$

3 ULTRAG

Our approach builds on the following two key insights:

Key insight #1: A successful query executor has to be robust to "LLM+KG noise".

The first insight comes from our observations that LLMs cannot be easily[2] and reliably constrained to build queries that use *only* existing triplets (LLM noise). A method like GCR (Luo et al., 2025a) that restricts the usable LLM output tokens for query generation, requires a modification to the LLM pipeline which is often infeasible and even impossible for closed LLMs. Moreover, KGs are incomplete (KG noise) and have missing relations, so such an approach may be robust to LLM noise, but susceptible to KG noise.

Prior research has made similar observations (Das et al., 2021; Yu et al., 2022; Mavromatis & Karypis, 2025) which is why the field has mostly moved away from LLM-written graph queries run with symbolic executors, and instead delegates the query execution/reasoning to the LLM. This brings us to our second insight:

Key insight #2: LLMs are **not** good neural executors.

LLMs underperform (Markeeva et al., 2024; Taylor et al., 2024) on graph algorithm simulation and in particular the Bellman-Ford algorithm (Cormen et al., 2022), when tested on larger instances. This is not an expressivity issue of the transformer architecture (De Luca & Fountoulakis, 2024), but to our best knowledge, no LLM has reported a robust performance on graph algorithms. This observation is important in the context of graph-RAG, because SOTA link prediction approaches such as ULTRA Galkin et al. (2024b) build on NBFNet (Zhu et al., 2021), which is a variation of the Bellman-Ford algorithm. Even if LLMs would improve on their graph-reasoning capabilities in the upcoming years, LLM inference would remain significantly less efficient: a back-of-the-envelope calculation, assuming near-perfect LLM conditions (linear attention, one output token, etc.) would still take 10^6 times more FLOPs than lightweight models, such as graph neural networks – see Appendix A.

In light of these observations, we hypothesise that *efficient and effective* query execution on KGs can be better achieved through the use of specialized neural query executors, rather than with pure LLM-based approaches.

3.1 ULTRAG

Our framework is visualised in Figure 1 and described in detail in Algorithm 1, with corresponding parts colour coded for clarity. Apart from the LLM and the neural query executor $\mathcal{X}$, the other key

[2] E.g. giving relation types to the LLM or prompt engineering.

components in our recipe are the entity linker $\mathcal{L}$, the sufficiency decider $\mathcal{D}$, and the arbitrator $\mathcal{A}$. The algorithm operates over fuzzy sets with membership functions in $\mathcal{F} = [0,1]^{|\mathcal{V}|}$. It proceeds iteratively constructing queries and refining a set of partial answers $\mathcal{P}$, until the membership function yielded by the neural query executor is deemed sufficient to answer the question. The partial answer set $\mathcal{P}$ consits of fuzzy sets, and it is instantiated at the beginning of the loop with seed information $\mathcal{P} = \{\boldsymbol{x}_s = \mathbf{1}_{v=s}\}$ if available; otherwise $\mathcal{P}$ is set to be the empty set. At each iteration, *after the query φ is constructed*, the entity linker populates the leaves of the query with a set of membership functions $\mathcal{I} = \{\boldsymbol{x}_{l_1}, \ldots, \boldsymbol{x}_{l_m}\}$, which connect the mentions associated to the leaves with the entities of the KG (m is the number of leaves in the query). The result $\boldsymbol{x} \in \mathcal{F}$ of the query execution is then checked by the decider $\mathcal{D}$ and the loop is terminated if $\boldsymbol{x}$ is enough to answer the query. The arbitrator $\mathcal{A}$ finally converts $\boldsymbol{x}$ into the desired answer.

We strongly emphasise that our recipe is universal: we do not fix $\mathcal{L}$, $\mathcal{D}$, $\mathcal{A}$ or $\mathcal{X}$, nor we fix the type of logic used by the LLM. In fact, in Section 3.2 and Section 4 we show that our framework can be instantiated with only off-the-shelf tools. However, experiments suggest that parametrising $\mathcal{X}$ as a neural network capable of simulating symbolic reasoning on knowledge graph is key to achieving good performance. To our best knowledge, this is the first time that a *neural* query executor is used in a KG RAG system.

3.2 UltRAG-OTS: An off-the-shelf UltRAG instance

3.2.1 Query construction and entity linking

A simpler Domain Specific Language Our initial attempt at constructing queries used a BetaE-style dataset format inspired from Ren & Leskovec (2020) and used in the original UltraQuery implementation. Unfortunately, this tuple based Domain Specific Language (DSL) often resulted in heavy bracketing (see Appendix C, Figure 3), which turned out to be hard to handle even for flagship LLMs like GPT-5 (OpenAI, 2025b). In our initial experiments, depending on the dataset, 15-30% of all queries were indeed found to be invalid tuples, which made those queries unexecutable. It is not our aim to deeply investigate this phenomenon, but our conjecture is that it closely relates to oversquashing (Barbero et al., 2024) and attention sinks (Gu et al., 2025; Arroyo et al., 2025) in LLMs. We therefore developed a custom DSL (see Appendix D, Figure 4), where for projection the LLM only has to append '−>' plus the relation identifier, and where logical operators can be n-ary. This seemingly minor edit reduced the number of invalid queries to less than 1%.

Entity linking A notable example of the generality of our framework, via the fuzzy set parametrization, is that it allows for the non-existence of uniquely identified seed entities (i.e. single entities that can be used as starting point for the reasoning process). Provided the target question, in the absence of seed entity information (i.e. $\mathcal{P} = \emptyset$), the LLM can generate the structured query one needs to execute, together with mentions $\{l_1, \ldots, l_m\}$ that will be matched against the knowledge base. Depending on similarity, each entity $v_j \in \mathcal{V}$ receives a probability of being the seed entity. Specifically, in our experiments, the probabilities $p(v_j \text{ is seed entity of } l_i)$ are computed as:

$$d_{ij} = \|\text{enc}(l_i) - \text{enc}(v_j)\|_2, \quad p(v_j \text{ is seed entity of } l_i) = \frac{\exp\left(-\frac{(d_{ij})^2}{2\sigma^2}\right)}{\sum_k \exp\left(-\frac{(d_{ik})^2}{2\sigma^2}\right)} \tag{1}$$

where $\sigma = 0.1$. The encoding function enc can be implemented with any pre-trained text embedding model (in our experiments we used E5_{large} (Wang et al., 2022)), and embeddings for $v_i \in \mathcal{V}$ can be inferred once and cached for multiple re-use. Given that knowledge bases can involve hundreds of millions of entities, efficient similarity search frameworks can additionally be used to enable UltRAG to deal with data of this scale. In our experiments, we used FAISS (Johnson et al., 2019; Douze et al., 2024), specifically its inverted file with product quantization (IVFPQ; Jégou et al., 2011) approximate nearest neighbor search, to efficiently implement Equation (1), and retrieve the k most similar entities to a given mention.

3.2.2 Query execution

For what concerns the choice of the neural query executor, we opted for UltraQuery (Galkin et al., 2024a) in our off the shelf implementation due to its good zero-shot performance, and robustness to

different choices of projection operators. Using relative relation type embeddings, in particular, allows the LLM to swap out a relation type with its semantic equivalent (e.g. `Child` for `Parent_inv`), while achieving the same result from the query execution. We expect that any improvements to knowledge graph foundational models (e.g. concurrent works such as Huang et al., 2025b) to naturally translate as improvements to ULTRAG, and any relation projection method with similar foundational properties (Arun et al., 2025) to have similar performance. However, we will not embark on a quantitative evaluation on the best choice of the neural query executor here – our aim is to show that there exists an instantiation of ULTRAG that can *efficiently* couple LLMs with query execution, while achieving *state-of-the-art performance*.

SEPPR-tor ULTRAQUERY scales linearly with the graph size both in time and in memory and can easily process all of `ogbl-wikikg2` (2.5M entities, 17M relations; Hu et al., 2020) on a single GPU (GH200 96GB). While multi-GPU setups can in principle be used for even larger KGs such as Wikidata, this would introduce architectural complexity and additional costs (both in terms of hardware and power consumption). Based on the observation that if answers exist for a generated query, they are typically located in a small neighborhood of the seed entities, we introduced a graph sampling step in our pipeline to reduce the amount of information that is fed in input to the query executor. Inspired by previous works in the graph machine learning literature (Klicpera et al., 2019; Gasteiger et al., 2019; Frasca et al., 2020), we resorted in particular to personalised page rank (Page et al., 1998) to extract a relevant subgraph localized around seed entities. Due to space limitations, we provide details of this algorithm in Appendix B.

Privacy and utility We conclude by noting that our approach can add an additional layer of privacy. ULTRAG is efficient enough to be deployed locally, even for large databases. Further, unlike prior methods (Sun et al., 2024; Chen et al., 2024; Li et al., 2025), the graph connectivity itself is not directly exposed to the LLM, nor do we need access to the LLM weights.

3.2.3 SUFFICIENCY AND QUERY ARBITRATION

While in principle multiple iterations of our approach could be useful to maximize performance (e.g. by breaking complex questions in multiple sub-queries that are iteratively executed), in our experiments we observed that a single query execution is generally sufficient to achieve good performance with modern powerful LLMs (GPT-5). Therefore, $\mathcal{D}$ in ULTRAG-OTS is set to always return *True*, thus terminating the while loop of Algorithm 1 after one iteration.

Finally, we implemented the arbitrator $\mathcal{A}$ as an LLM (GPT-5) that takes as input the top ranked entities in $\boldsymbol{x}$, together with their probabilities, and then is prompted to return the final set of answers. Using an LLM at this stage can be particularly beneficial as it makes the overall approach able to deal with questions that one cannot directly answer with first-order logic (e.g. counting or temporal queries), while also giving the LLM the possibility to use its knowledge to correct mistakes done by the query executor.

4 EXPERIMENTS

We present a set of experimental results that address multiple **R**esearch **Q**uestions (Section 4.2), along with details on the datasets and baselines used, and additional information required to reproduce our implementation (Section 4.1).

4.1 EXPERIMENTAL SETUP

KGQA Datasets and Knowledge Graphs For evaluating ULTRAG , we focus on datasets where the ground-truth source of information is already in the form of a knowledge graph. This allows to remove any evaluation bias originating from a knowledge-graph construction step (e.g. from web or textual data). In line with this, datasets such as SimpleQA (Wei et al., 2024) or FACTS (Cheng et al., 2025) were excluded in our experiments. In a recent work, Zhang et al. (2025) additionally observed that the average correctness rate in some prominent KGQA datasets is too low for reliable benchmarking. Therefore, we decided to not focus on widely used but error-prone datasets such as WebQSP (Yih et al., 2016, 52% correct) or CWQ (Talmor & Berant, 2018, 49% correct), and to

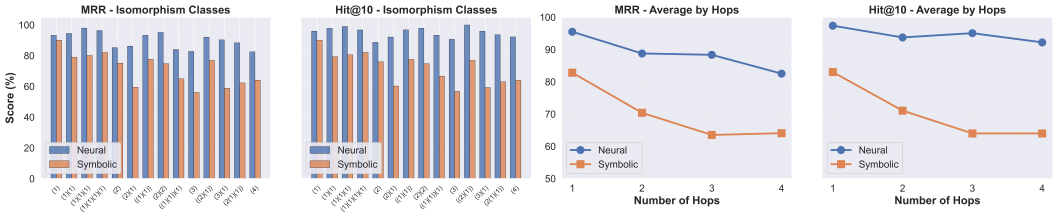

Figure 2: Comparison of ULTRAQUERY vs symbolic query execution on GTSQA. Both receive identical queries generated by the LLM. Number inside brackets denotes projections, concatenation of expression denotes intersection. Best viewed on screen.

prioritise instead newer, programmatically verified, datasets: KGQAGen-10K (Zhang et al., 2025) and GTSQA (Cattaneo et al., 2025). For the knowledge graph we use the Wikidata graph from `ogbl-wikikg2` for GTSQA, and the `20251202` dump of Wikidata, processed as in Chen et al. (2024)[3], for KGQAGen-10K.

Baselines We exclude methods constructing a knowledge graph from another data format or retrieving documents as context, based on some knowledge graph connectivity (Luo et al., 2025b). Here we compare ULTRAG-OTS against *KG Agents* (ToG Sun et al., 2024), *path-based* approaches (RoG; GCR Luo et al., 2024; 2025a), *GNN-based* approaches (GNN-RAG Mavromatis & Karypis, 2025) and *hybrid* approaches (SubgraphRAG Li et al., 2025). We follow established literature in our choice of metrics. For ranking of entities, we report ranking-based scores – Mean Reciprocal Rank (MRR) and Hit@1,3,10. For final, discrete set answers, we report exact match hits, recall and F1 of precision and recall. Unlike recall, Hits measures the percentage of questions with at least one correct answer getting retrieved.

Training and Hyperparameters We do not perform any training on $\mathcal{X}, \mathcal{A}, \mathcal{D}$. When ground truth entities are available, no training of $\mathcal{L}$ is required. When they are not, a training phase is required with FAISS for the construction of the IVFPQ index (which takes less than 1 hr on a 64-core ARM Neoverse-V2). This makes ULTRAG-OTS an inductive approach, as opposed to baselines, which require retraining for each dataset. For ULTRAQUERY (i.e. $\mathcal{X}$), we used the official checkpoint[4] that is trained on FB15k-237, WN18RR, CoDEx-Medium.

For ULTRAG we did not perform hyperparameter searches – our goal is to show a working instantiation not find the most optimal one. With FAISS, we used $16,384$ cluster centroids for IVF, and 128 8-bit subquantizers per vector embedding for PQ. For Algorithm 2, we picked $k = 30,000$ for both GTSQA and KGQAGen-10K. We passed the top 50 candidates to $\mathcal{A}$, as ranked by $\mathcal{X}$ to be in the answer set.

We should point out that, in our experiments, all our queries are constructed using only AND operators. While we could have added other logical operators (OR, NOT, etc.), or allow for cyclic queries (Cucumides et al., 2024), we found it unnecessary for obtaining good performance. We also allow the LLM to use inverse projections by adding an `_inv` suffix. This can be done regardless of whether inverses exist in the knowledge graph or not (e.g. `Child_inv` is semantically equivalent to `Parent`). Lastly, we require the LLM to refer to relations by their identifiers (`P<digits>` in Wikidata / WikiKG2), instead of their labels or descriptions.

4.2 RESULTS

RQ1: How much does neural query execution improve a LLM's ability to interface with KGs? We compare the performance of ULTRAQUERY against the one achieved by a symbolic executor, when receiving as input structured queries generated by a LLM (GPT-5). For both methods, we run the queries on the WikiKG2 subgraphs provided in GTSQA and rank the entities by their probability to be in the answer set (for symbolic execution each entity receives a score equal to 1 or 0 based on the result of the executor). In Figure 2 we compare the MRR, and the hit@10 produced for

[3] https://github.com/liyichen-cly/PoG
[4] https://github.com/DeepGraphLearning/ULTRA/tree/ultraquery

Table 1: Comparison on PPR subgraphs extracted from the ground-truth seed entities and seed entities obtained via Entity Linking. PPR subgraphs contain up to 30,000 nodes. All models use GPT-5 as reasoning LLM. We color-code settings that require **transductive** and **inductive** reasoning for the considered baselines; for GTSQA (Wikidata), we used baseline models originally trained on GTSQA (WikiKG2). We leave a '-' where a baseline cannot be applied.

	Ground-truth seed nodes									Entity Linking								
	GTSQA (WikiKG2)			GTSQA (Wikidata)			KGQAGen-10k			GTSQA (WikiKG2)			GTSQA (Wikidata)			KGQAGen-10k		
Model	**Hits**	**Recall**	**F1**	**Hits**	**Recall**	**F1**	**Hits**	**Recall**	**F1**	**Hits**	**Recall**	**F1**	**Hits**	**Recall**	**F1**	**Hits**	**Recall**	**F1**
RoG	72.63	70.73	69.81	62.55	59.81	58.93	86.25	82.60	82.68	72.00	69.97	68.88	60.64	57.66	56.85	82.42	78.89	79.22
GNN-RAG	64.98	62.89	62.70	-	-	-	81.04	76.66	77.56	51.91	49.36	49.59	-	-	-	72.89	69.04	69.53
SubgraphRAG (200)	73.98	71.14	70.91	63.29	59.60	59.56	80.47	75.88	76.54	71.82	69.37	69.01	58.61	55.43	55.04	76.68	72.63	73.17
ULTRAG-OTS	**90.81**	**89.39**	**87.18**	**86.74**	**84.47**	**82.08**	**90.62**	**89.41**	**87.63**	**85.70**	**83.71**	**81.08**	**72.93**	**70.48**	**66.60**	**83.98**	**82.59**	**80.58**

WikiKG2. As we can see, ULTRAQUERY consistently outperforms the query executor in retrieving the answer nodes across all the considered classes of answers in both scenarios, achieving a class average improvement of 18.58% (MRR) and 24.09% (hit@10). Due to space constraints, additional metrics and results with Wikidata subsampled to 30,000 nodes using Algorithm 2 are presented in Appendix E. Taken together, the results highlight the benefit of using a *neural* query executor to make our framework resilient to both LLM and KG noise. Additional experiments comparing the performance obtained with ULTRAQUERY using noiseless queries generated by an oracle, to the ones produced by the LLM can be found in the Appendix in Table 6. These highlight that, while GPT-5 is generally able to produce queries that correctly retrieve the answer nodes, this is by no means perfect, and further improvements in the LLM's query generation capabilities can only be expected to improve the performance discussed in the next paragraphs.

RQ2: How does ULTRAG-OTS compare in performance against other KG-RAG approaches? To answer RQ2, we break down evaluation in two different parts. First, we compare ULTRAG-OTS and our baselines on the WikiKG2 subgraphs provided in GTSQA Cattaneo et al. (2025) and on subgraphs of similar size constructed from Wikidata via PPR for KGQAGen-10K Zhang et al. (2025) (Table 2). For both datasets, proof edges and answer nodes are always contained in the considered subgraphs, adding them back if they were lost during subsampling. This setting represents a simplified controlled version of a real-world scenario, where all the information a model needs to answer a given question is guaranteed to be available in the considered subgraph. As a result, this provides a suitable testbed for evaluating our LLM plus neural query executor framework, which is the core novel component of ULTRAG. On GTSQA, ULTRAG achieves 92.66% exact match hits, 91.05% recall, and 89.29% F1. This represents an improvement of $+8.32\%$ in hits, $+9.39\%$ in recall, and $+8.03\%$ in F1 over the second best performing baseline (SubgraphRAG with 200 retrieved triples Li et al. (2025)). On KGQAGen-10k, ULTRAG achieves 92.04% hits, 90.77% recall, and 88.82% F1, outperforming the best baseline (again SubgraphRAG (200)) by respectively $+2.28\%$, $+5.25\%$, and $+2.54\%$. In the second part of our evaluation, we compare performance of the full pipeline described for ULTRAG-OTS against similar end-to-end pipelines implemented for RoG, GNN-RAG and SubgraphRAG (200) (the three best performing baselines in Table 2). For these baselines, we use a LLM (GPT-5) to retrieve mentions that the LLM believes to be relevant for answering the provided question. We then link said mentions with entities in the provided KGs and extract relevant subgraphs in the same way we do for ULTRAG-OTS. Table 1 provides a comparison of this evaluation for both transductive and inductive reasoning settings, together with results for an intermediate step, where we assume seed entities to be given. As we can see, ULTRAG-OTS outperforms all methods in our evaluation, in most cases achieving a double digit improvement in F1 on the considered datasets.

Table 2: Comparison on question-specific graphs, for GTSQA (WikiKG2) and KGQAGen-10k (Wikidata).

	GTSQA			KGQAGen-10k		
Model	**Hits**	**Recall**	**F1**	**Hits**	**Recall**	**F1**
GPT-5-mini	33.42	31.92	31.82	62.18	59.17	59.58
GPT-4.1	33.72	32.63	32.36	56.21	53.38	53.93
GPT-5	46.36	44.40	44.16	67.96	65.08	65.27
ToG	64.73	61.99	62.06	80.97	75.03	76.10
GCR	60.83	58.41	58.15	86.11	79.89	80.22
RoG	76.51	74.61	73.99	88.43	84.69	84.92
GNN-RAG	76.76	74.95	74.90	82.56	77.98	78.92
SubgraphRAG (200)	84.34	81.66	81.62	89.76	85.52	86.28
ULTRAG-OTS	**92.66**	**91.05**	**89.29**	**92.04**	**90.77**	**88.82**

RQ3: How does ULTRAG-OTS perform with different LLMs (both of different families and sizes)? For this particular research question we report in Table 7 (in Appendix G due to space constraints) an ablation study showing the performance achieved by ULTRAG-OTS with LLMs of the GPT and DeepSeek Liu et al. (2025) families. For these experiments, we used PPR subgraphs extracted from Wikidata starting from ground truth seed entities. While using GPT-5 for both query generation and arbitration yields best performance (likely due to its reasoning capabilities), we note that all the tested combinations achieve good results in our analysis. Interestingly, the combination of GPT-5, for query generation, and GPT-5-mini, for arbitration, achieves close performance to our best model, suffering a reduction of only 4% in Hits and F1 ($\sim$7% for recall). These findings suggest that lighter LLMs could possibly be used in ULTRAG whenever cost considerations are of relevance (especially for the arbitration stage, which likely requires less reasoning), while maintaining good performance overall.

RQ4: How does ULTRAG-OTS compare with previous KG-RAG approaches in terms of efficiency? To assess the efficiency of ULTRAG-OTS, we measured the average API cost per query with our LLM of choice (GPT-5), the average (non-API) runtime, and the average number of input/output tokens processed/generated by the LLM for the top-3 baselines of Table 2 and ULTRAG-OTS. All experiments start from the PPR subgraphs extracted from either WikiKG2 or Wikidata. We note that this comparison excludes entity linking and the subgraph extraction phases, as these are not defined for any of the considered baselines. Results for GTSQA are reported in Table 3 and commented here, while results and analysis for KGQAGen are available in Appendix H. On WikiKG2 subgraphs, ULTRAG-OTS is 19x/99x/167x faster than RoG/GNN-RAG/SubgraphRAG in terms of non-API time per query (similar results also hold for Wikidata subgraphs). Due to the presence of relation types in the prompt and two API calls, ULTRAG-OTS processes between 25x and 62x more input tokens than the baselines. However, $94 - 96\%$ of these tokens are cached by GPT-5 (due to our prompts being structurally identical), limiting both cost and computational complexity. The average number of output tokens is comparable across all methods. In terms of API cost, ULTRAG-OTS is 23% to 27% more expensive than the baselines (due to the larger size of the input), however, we emphasize that no additional cost would be required for extracting relevant mentions from input queries with ULTRAG-OTS, while all baselines would incur extra costs for this step in an end-to-end pipeline[5]. Moreover, as shown in Table 7, ULTRAG-OTS achieves a better performance than the baselines even when using smaller and cheaper LLMs, in which case ULTRAG-OTS compares favourably both in terms of results and efficiency.

Table 3: Average efficiency per query on PPR subgraphs extracted for GTSQA from WikiKG2 (top) and Wikidata (bottom) with ground-truth seed nodes. All models use GPT-5 as reasoning LLM. Non-API running times are measured on GH200 chips, excluding API calls and PPR computation.

KG	Model	API cost	Non-API time (s)	Input tokens	Output tokens	Cache hit %
WikiKG2	RoG	0.011$	1.9 ± 0.5	0.9K	2.2K	0
	GNN-RAG	0.012$	9.9 ± 2.3	1.5K	2.1K	0
	SubgraphRAG (200)	0.012$	16.7 ± 3.8	2.7K	2.1K	0
	ULTRAG-OTS	0.014$	0.10 ± 0.0	23K	2.2K	93.77
Wikidata	RoG	0.013$	4.0 ± 1.1	1.1K	2.5K	0
	SubgraphRAG (200)	0.014$	20.9 ± 3.5	3.1K	2.4K	0
	ULTRAG-OTS	0.017$	0.15 ± 0.1	69K	2.3K	95.99

5 CONCLUSIONS

In this work we introduced ULTRAG, a universal modular framework for Knowledge Graph Question Answering systems. Our off-the-shelf implementation, ULTRAG-OTS, achieves state-of-the-art results across a variety of settings and is able to effectively handle Wikidata-scale KGs in an end-to-end QA pipeline. Despite showing strong performance in standard KGQA settings, ULTRAG-OTS has limitations that may constrain its applicability for some real-world use cases. In particular, ULTRAG-OTS does not natively support temporal queries, which would require extending our methodology to Temporal Knowledge Graphs Lin et al. (2023). Similarly, ULTRAG-OTS might underperform on Knowledge HyperGraphs due to its reliance on ULTRA, which has been shown to be suboptimal in this setting Huang et al. (2025c). Addressing these limitations constitutes an interesting research direction that we plan to investigate further in future work.

[5]With GPT-5 for mention extraction, baselines would incur an extra 0.003$ (+490 outp. tok.) per query, making costs comparable.

REFERENCES

Farahnaz Akrami, Mohammed Samiul Saeef, Qingheng Zhang, Wei Hu, and Chengkai Li. Realistic re-evaluation of knowledge graph completion methods: An experimental study. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pp. 1995–2010, 2020.

Álvaro Arroyo, Federico Barbero, Hugh Blayney, Michael Bronstein, Xiaowen Dong, Pietro Liò, Razvan Pascanu, and Pierre Vandergheynst. Bridging graph neural networks and large language models: A survey and unified perspective. 2025.

Arvindh Arun, Sumit Kumar, Mojtaba Nayyeri, Bo Xiong, Ponnurangam Kumaraguru, Antonio Vergari, and Steffen Staab. Semma: A semantic aware knowledge graph foundation model. *arXiv preprint arXiv:2505.20422*, 2025.

Federico Barbero, Andrea Banino, Steven Kapturowski, Dharshan Kumaran, João Madeira Araújo, Oleksandr Vitvitskyi, Razvan Pascanu, and Petar Veličković. Transformers need glasses! information over-squashing in language tasks. *Advances in Neural Information Processing Systems*, 37: 98111–98142, 2024.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26, 2013.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pp. 2206–2240. PMLR, 2022.

Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges. *arXiv preprint arXiv:2104.13478*, 2021.

Alberto Cattaneo, Carlo Luschi, and Daniel Justus. Ground-truth subgraphs for better training and evaluation of knowledge graph augmented llms. *arXiv preprint arXiv:2511.04473*, 2025. URL https://arxiv.org/abs/2511.04473.

Liyi Chen, Panrong Tong, Zhongming Jin, Ying Sun, Jieping Ye, and Hui Xiong. Plan-on-graph: Self-correcting adaptive planning of large language model on knowledge graphs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=CwCUEr6wO5.

Aileen Cheng, Alon Jacovi, Amir Globerson, Ben Golan, Charles Kwong, Chris Alberti, Connie Tao, Eyal Ben-David, Gaurav Singh Tomar, Lukas Haas, Yonatan Bitton, Adam Bloniarz, Aijun Bai, Andrew Wang, Anfal Siddiqui, Arturo Bajuelos Castillo, Aviel Atias, et al. The FACTS leaderboard: A comprehensive benchmark for large language model factuality. *arXiv preprint arXiv:2512.10791*, 2025. URL https://arxiv.org/abs/2512.10791.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 4 edition, 2022.

Tamara Cucumides, Daniel Daza, Pablo Barcelo, Michael Cochez, Floris Geerts, Juan L Reutter, and Miguel Romero Orth. Unravl: A neuro-symbolic framework for answering graph pattern queries in knowledge graphs. In *The Third Learning on Graphs Conference*, 2024. URL https://openreview.net/forum?id=183XrFqaHN.

Rajarshi Das, Manzil Zaheer, Dung Thai, Ameya Godbole, Ethan Perez, Jay-Yoon Lee, Lizhen Tan, Lazaros Polymenakos, and Andrew Mccallum. Case-based reasoning for natural language queries over knowledge bases. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 9594–9611, 2021.

Arturs Back De Luca and Kimon Fountoulakis. Simulation of graph algorithms with looped transformers. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 2319–2363, 2024.

Andreea Deac, Petar Veličković, Ognjen Milinković, Pierre-Luc Bacon, Jian Tang, and Pietro Liò. XLVIN: executed latent value iteration nets. In *Third Workshop on Graphs and more Complex Structures for Learning and Reasoning at AAAI 2021*, 2021.

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. 2024.

Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 2020.

Michael Galkin, Jincheng Zhou, Bruno Ribeiro, Jian Tang, and Zhaocheng Zhu. A foundation model for zero-shot logical query reasoning. *Advances in Neural Information Processing Systems*, 37: 54137–54160, 2024a.

Mikhail Galkin, Xinyu Yuan, Hesham Mostafa, Jian Tang, and Zhaocheng Zhu. Towards foundation models for knowledge graph reasoning. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7–11, 2024*. OpenReview.net, 2024b. URL https://openreview.net/forum?id=jVEoydFOl9.

Johannes Gasteiger, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. *Advances in neural information processing systems*, 32, 2019.

Xiangming Gu, Tianyu Pang, Chao Du, Qian Liu, Fengzhuo Zhang, Cunxiao Du, Ye Wang, and Min Lin. When attention sink emerges in language models: An empirical view. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=78Nn4QJTEN.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43(2):1–55, 2025a.

Xingyue Huang, Pablo Barceló, Michael M. Bronstein, İsmail İlkan Ceylan, Mikhail Galkin, Juan L. Reutter, and Miguel A. Romero Orth. How expressive are knowledge graph foundation models? In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13–19, 2025*. OpenReview.net, 2025b. URL https://openreview.net/forum?id=mXEdUcLtaK.

Xingyue Huang, Mikhail Galkin, Michael M Bronstein, and İsmail İlkan Ceylan. Hyper: A foundation model for inductive link prediction with knowledge hypergraphs. *arXiv preprint arXiv:2506.12362*, 2025c.

Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. Atlas: Few-shot learning with retrieval augmented language models. *Journal of Machine Learning Research*, 24(251):1–43, 2023.

Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *International Conference on Learning Representations (ICLR)*, 2019.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33: 9459–9474, 2020.

Mufei Li, Siqi Miao, and Pan Li. Simple is effective: The roles of graphs and large language models in knowledge-graph-based retrieval-augmented generation. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=JvkuZZ04O7.

Xueyuan Lin, Chengjin Xu, Gengxian Zhou, Haoran Luo, Tianyi Hu, Fenglong Su, Ningyuan Li, Mingzhi Sun, et al. Tflex: Temporal feature-logic embedding framework for complex reasoning over temporal knowledge graph. *Advances in Neural Information Processing Systems*, 36:73039–73081, 2023.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. Reasoning on graphs: Faithful and interpretable large language model reasoning. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=ZGNWW7xZ6Q.

Linhao Luo, Zicheng Zhao, Gholamreza Haffari, Yuan-Fang Li, Chen Gong, and Shirui Pan. Graph-constrained reasoning: Faithful reasoning on knowledge graphs with large language models. In *Forty-second International Conference on Machine Learning*, 2025a.

Linhao Luo, Zicheng Zhao, Gholamreza Haffari, Dinh Phung, Chen Gong, and Shirui Pan. Gfm-rag: Graph foundation model for retrieval augmented generation. *NeurIPS 2025*, 2025b.

Larisa Markeeva, Sean McLeish, Borja Ibarz, Wilfried Bounsi, Olga Kozlova, Alex Vitvitskyi, Charles Blundell, Tom Goldstein, Avi Schwarzschild, and Petar Veličković. The clrs-text algorithmic reasoning language benchmark. In *ICML 2024 Workshop on Data-centric Machine Learning Research (DMLR): Datasets for Foundation Models*, 2024. URL https://openreview.net/forum?id=cG0UutsUYh. Workshop poster (listed on ICML 2024 virtual site).

Costas Mavromatis and George Karypis. Gnn-rag: Graph neural retrieval for efficient large language model reasoning on knowledge graphs. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 16682–16699, 2025.

Costas Mavromatis, Soji Adeshina, Vassilis N Ioannidis, Zhen Han, Qi Zhu, Ian Robinson, Bryan Thompson, Huzefa Rangwala, and George Karypis. Byokg-rag: Multi-strategy graph retrieval for knowledge graph question answering. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 27869–27886, 2025.

Danilo Numeroso, Davide Bacciu, and Petar Veličković. Dual algorithmic reasoning. *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=bnEarBBf7q.

OpenAI. gpt-oss-120b & gpt-oss-20b model card, August 2025a. URL https://arxiv.org/abs/2508.10925.

OpenAI. GPT-5 System Card. https://cdn.openai.com/gpt-5-system-card.pdf, August 2025b. System card detailing GPT-5's architecture, safety evaluations, and mitigations. Accessed December 30, 2025.

Larry Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical Report SIDL-WP-1998-0120, Stanford University, 1998. URL ilpubs.stanford.edu. Often cited as Page et al. (1999) in academic literature.

Heiko Paulheim. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic web*, 8(3):489–508, 2016.

Hongyu Ren and Jure Leskovec. Beta embeddings for multi-hop logical reasoning in knowledge graphs. *Advances in Neural Information Processing Systems*, 33:19716–19726, 2020.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pp. 593–607. Springer, 2018.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel Ni, Heung-Yeung Shum, and Jian Guo. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=nnVO1PvbTv.

Alon Talmor and Jonathan Berant. The web as a knowledge-base for answering complex questions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Long Papers)*, pp. 641–651, New Orleans, Louisiana, 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1059.

Alexander K Taylor, Anthony Cuturrufo, Vishal Yathish, Mingyu Derek Ma, and Wei Wang. Are large-language models graph algorithmic reasoners? *arXiv preprint arXiv:2410.22597*, 2024.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *International conference on machine learning*, pp. 2071–2080. PMLR, 2016.

Petar Veličković, Rex Ying, Matilde Padovano, Raia Hadsell, and Charles Blundell. Neural execution of graph algorithms. *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SkgKO0EtvS.

Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*, 2022.

Jason Wei, Karina Nguyen, Hyung Won Chung, Yunxin Joy Jiao, Spencer Papay, Amelia Glaese, John Schulman, and William Fedus. Measuring short-form factuality in large language models. *arXiv preprint arXiv:2411.04368*, 2024. URL https://arxiv.org/abs/2411.04368.

Keyulu Xu, Jingling Li, Muhan Zhang, Simon S. Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rJxbJeHFPS.

Keyulu Xu, Mozhi Zhang, Jingling Li, Jure Leskovec, Stefanie Jegelka, et al. How neural networks extrapolate: From feedforward to graph neural networks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=UH-cmocLJC.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7–9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6575.

Wen-tau Yih, Matthew Richardson, Christopher Meek, Ming-Wei Chang, and Jina Suh. The value of semantic parse labeling for knowledge base question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 201–206, Berlin, Germany, 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-2033.

Jiaxuan You, Jonathan M Gomes-Selman, Rex Ying, and Jure Leskovec. Identity-aware graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 10737–10745, 2021.

Donghan Yu, Shenmin Zhang, Patrick Ng, Henghui Zhu, A. Li, J. Wang, Yiqun Hu, William Wang, Zhiguo Wang, and Bing Xiang. Decaf: Joint decoding of answers and logical forms for question answering over knowledge bases. *International Conference on Learning Representations*, 2022. doi: 10.48550/arXiv.2210.00063.

He Yu, Gleb Nikitin, Junyoung Chung, Andrew Dudzik, and Petar Veličković. Primal-dual neural algorithmic reasoning. In *Learning on Graphs Conference 2023*, 2023.

Liangliang Zhang, Zhuorui Jiang, Hongliang Chi, Haoyang Chen, Mohammed ElKoumy, Fali Wang, Qiong Wu, Zhengyi Zhou, Shirui Pan, Suhang Wang, and Yao Ma. Diagnosing and addressing pitfalls in KG-RAG datasets: Toward more reliable benchmarking. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025. URL https://openreview.net/forum?id=Vd5JXiX073.

Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. Labeling trick: A theory of using graph neural networks for multi-node representation learning. *Advances in Neural Information Processing Systems*, 34:9061–9073, 2021.

Zhanqiu Zhang, Jianyu Cai, Yongdong Zhang, and Jie Wang. Learning hierarchy-aware knowledge graph embeddings for link prediction. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 3065–3072, 2020.

Lingfeng Zhong, Jia Wu, Qian Li, Hao Peng, and Xindong Wu. A comprehensive survey on automatic knowledge graph construction. *ACM Computing Surveys*, 56(4):1–62, 2023.

Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. Neural bellman-ford networks: A general graph neural network framework for link prediction. *Advances in neural information processing systems*, 34:29476–29490, 2021.

Zhaocheng Zhu, Xinyu Yuan, Michael Galkin, Louis-Pascal Xhonneux, Ming Zhang, Maxime Gazeau, and Jian Tang. A* net: A scalable path-based reasoning approach for knowledge graphs. *Advances in neural information processing systems*, 36:59323–59336, 2023.

A The Cost of Running LLMs – a FLOPs Analysis

This appendix provides a coarse-grained floating point operations (FLOPs) comparison between a message-passing graph neural network (GNN) and a middle-sized LLM, in particular `GPT-OSS-120B` (OpenAI, 2025a), under simplified but explicit assumptions. The goal is to illustrate order-of-magnitude compute differences incurred by using an unnecessarily large architecture rather than precise hardware-level performance. We note that this is a smaller GPT-class LLM than the ones we used, so the LLM compute flops are quite optimistic.

A.1 GNN Assumptions and FLOPs calculation

We consider a $L = 6$ layer message-passing GNN with hidden dimension $d = 64$ (Galkin et al., 2024b, Appendix C). Let N denote the number of nodes and E denote the number of edges in the graph.

Each GNN layer consists of the following operations:

$$
\begin{aligned}
m_{ij} &= h_i + h_j \\
m_i &= \sum_{j \in \mathcal{N}(i)} m_{ij} \\
h_i' &= \mathrm{MLP}(m_i)
\end{aligned}
$$

The update MLP is a 2-layer perceptron with dimensions $64 \to 64 \to 64$, applied independently to each node. As this is strictly a FLOP analysis, we do not account for irregular access patterns, which can influence wall-clock GNN runtime.

Message computation For each edge, an elementwise addition of two d-dimensional vectors is performed:

$$\text{FLOPs}_{\text{msg}} = E \times d$$

Aggregation Aggregation is implemented as a sum over incoming messages, incurring one addition per feature per edge:

$$\text{FLOPs}_{\text{aggr}} = E \times d$$

Update MLP Each linear layer of size $d \times d$ requires approximately $2d^2$ FLOPs per node (counting one multiply and one add). For a two-layer MLP:

$$\text{FLOPs}_{\text{MLP}} \approx 2 \times 2d^2 N = 4d^2 N$$

Combining all terms and substituting for $d = 64$ gives:

$$\text{FLOPs}_{\text{GNN, layer}} \approx 2Ed + 4d^2 N = 128E + 16384N$$

For $L = 6$ layers:

$$\text{FLOPs}_{\text{GNN}} \approx 6(128E + 16384N) = 768E + 98304N$$

A.2 LLM Assumptions and FLOPs calculation

We compare against a `GPT-OSS-120B`, with the following *overly* simplifying assumptions:

- Active parameters per token are $P_{\text{active}} \approx 5.1 \times 10^9$ as per OpenAI (2025a, Table 1).
- FLOPs per token are twice the active parameters – one add and one multiply per active parameter.
$$\text{FLOPs}_{\text{token}} \approx 10.2 \times 10^9$$
- Input sequence length is $T = N + E$ tokens.
- One output token is generated
- Quadratic attention costs are ignored, assuming some perfect KV-caching/compression algorithm. Attention would otherwise further disadvantage the LLM at large scales.

These additional assumptions further lower the bound on total LLM compute.

LLM FLOPs Total FLOPs for prompt prefill and one decoding step:

$$\text{FLOPs}_{\text{LLM}} \approx \text{FLOPs}_{\text{token}} \times (T + 1) \approx 10.2 \times 10^9 \times (N + E)$$

A.3 Numerical Example

Assume we have a graph with $N = 3{,}000$ nodes and $E = 30{,}000$ edges. This is considerably smaller than Wikidata or our top-30,000 PPR graph.

$$\begin{aligned}
\text{FLOPs}_{\text{GNN}} &= 768 \times 30{,}000 + 98{,}304 \times 3{,}000 \\
&\approx 3.18 \times 10^8 \text{ FLOPs} \\
\text{FLOPs}_{\text{LLM}} &= 10.2 \times 10^9 \times 33{,}000 \\
&\approx 3.37 \times 10^{14} \text{ FLOPs}
\end{aligned}$$

The ratio of compute is:

$$\frac{\text{FLOPs}_{\text{LLM}}}{\text{FLOPs}_{\text{GNN}}} \approx 1.06 \times 10^6$$

Thus, for the considered graph size and architectures, a single inference pass of the LLM requires approximately six orders of magnitude more floating point operations than the full 6-layer GNN.

Algorithm 2 Seed Entity Personalized PageRank (SEPPR)

Require: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, seed set $\mathcal{S}$: either (i) crisp set $\mathcal{S} \subseteq \mathcal{V}$ if ground-truth entities known, or (ii) set of fuzzy sets $\{\{p_i(v)\}_{v \in \mathcal{V}}\}_i$ from entity linking (see **??**), damping factor $\alpha = 0.85$, $T = 5$, k

Ensure: Top-k nodes sorted by PPR scores

$\boldsymbol{x}_0 \leftarrow \mathbf{0}$ {Initialize probability vector}
if $\mathcal{S}$ is crisp set **then**
  $\boldsymbol{x}_0[v] \leftarrow 1/|\mathcal{S}|$ for all $v \in \mathcal{S}$ {Case (i): uniform weights}
else
  $\boldsymbol{x}_0[v] \leftarrow \sum_i p_i(v)$ for all $v \in \mathcal{V}$ {Case (ii): Combine probabilities across fuzzy sets}
  $\boldsymbol{x}_0 \leftarrow \boldsymbol{x}_0 / \sum_{u \in \mathcal{V}} \boldsymbol{x}_0[u]$ {Renormalize}
end if
$\boldsymbol{x} \leftarrow \boldsymbol{x}_0$
for $t = 1$ to T **do**
  $\boldsymbol{x}'[v] \leftarrow \sum_{(u,v) \in \mathcal{E}} \frac{\boldsymbol{x}[u]}{\text{degree}[u]}$ for all $v \in \mathcal{V}$ {Diffuse}
  $\boldsymbol{x} \leftarrow \alpha \boldsymbol{x}' + (1 - \alpha) \boldsymbol{x}_0$ {Teleport to starting nodes with probability α}
end for
return top-k nodes by $\boldsymbol{x}$ values

B SEPPR

We present our implementation in Algorithm 2. Differently from past works:

- when we use ground-truth seed entities, we initialise the boundary condition $\boldsymbol{x}_0$ to be a uniform signal placed on the seed entities;
- when entity linking is used instead, $\boldsymbol{x}_0[v]$ is set to be the (normalized) sum of probabilities of v being one of the seed entities.

After initialisation, T probability diffusion steps take place and then the knowledge graph is subsampled to the top-k nodes with highest $\boldsymbol{x}[v]$. Differently from ULTRAQUERY, our PPR algorithm processes only a mono-dimensional signal, and as a result it can fit on a single GPU (GH200 96GB), even for web scale KGs such as Wikidata.

In our experiments, we set $k = 30,000$. However, the number of edges in the subgraphs induced by the PPR nodes cause out-of-memory issues for certain baselines. To remedy that, we prune the number of PPR nodes to ensure that there are no more than 500,000 edges in the subgraphs for the baselines.

C BETAE NESTING EXAMPLE

Full Query:
`(((TA, (P1_inv,)), (DL, (P2_inv,))), (P4,))`

Breakdown:
Projection 1: `(TA, (P1_inv,))`
Projection 2: `(DL, (P2_inv,))`
Intersection: `(Proj 1, Proj 2)`
Final Projection: `(Intersection, (P4,))`

Figure 3: BetaE format representation of the Turing Award query example from Figure 1 (entities have been abbreviated). Projections follow the language defined in Figure 4, left. The LLM is tasked with producing the full query; the breakdown is added only for human readability.

Table 4: Comparison of neural (UltraQuery) vs symbolic query execution on LLM-generated queries from GTSQA using WikiKG2 subgraphs (in %). Class notation as in Cattaneo et al. – number inside brackets denotes hops, concatenation of expression denotes intersection. Both executors receive identical queries generated by the LLM.

Metric	# hops = 1					# hops = 2						# hops = 3					# hops = 4	
	(1)	(1) (1)	(1) (1) (1)	(1) (1) (1) (1)	Avg	(2)	(2) (1)	((1) (1))	(2) (2)	((1) (1)) (1)	Avg	(3)	((2) (1))	(3) (1)	(2 (1) (1))	Avg	(4)	Avg
Neural executor (UltraQuery)																		
MRR	93.25	94.48	97.96	96.36	95.51	85.24	86.19	93.28	95.17	84.01	88.78	82.78	92.00	90.30	88.35	88.36	82.54	82.54
Hit@1	91.33	93.17	96.94	95.93	94.34	82.67	82.48	92.00	93.05	80.00	86.04	77.50	87.69	88.00	85.00	84.55	76.22	76.22
Hit@3	94.67	95.00	98.98	96.75	96.35	87.33	87.99	93.60	97.60	86.17	90.54	87.87	93.85	90.67	92.27	91.16	88.22	88.22
Hit@10	96.00	97.78	98.98	96.75	97.38	88.67	92.13	96.80	97.84	93.33	93.75	90.65	100.00	96.00	93.64	95.07	92.22	92.22
Symbolic executor																		
MRR	90.03	79.05	80.13	82.12	82.83	75.11	59.51	77.62	74.84	64.96	70.41	56.13	76.95	58.78	62.28	63.53	64.08	64.08
Hit@1	90.00	78.67	79.59	82.11	82.59	74.67	59.06	77.60	74.82	63.33	69.90	55.65	76.92	58.00	61.67	63.06	64.00	64.00
Hit@3	90.00	79.33	80.61	82.11	83.01	75.33	59.84	77.60	74.82	66.67	70.85	56.20	76.92	59.33	63.03	63.87	64.00	64.00
Hit@10	90.00	79.33	80.61	82.11	83.01	76.00	60.24	77.60	74.82	66.67	71.07	56.76	76.92	59.33	63.03	64.01	64.00	64.00
Improvement (Neural - Symbolic)																		
MRR	+3.22	+15.43	+17.83	+14.24	+12.68	+10.13	+26.68	+15.66	+20.33	+19.05	+18.37	+26.65	+15.05	+31.52	+26.07	+24.82	+18.46	+18.46
Hit@1	+1.33	+14.50	+17.35	+13.82	+11.75	+8.00	+23.42	+14.40	+18.23	+16.67	+16.14	+21.85	+10.77	+30.00	+23.33	+21.49	+12.22	+12.22
Hit@3	+4.67	+15.67	+18.37	+14.64	+13.34	+12.00	+28.15	+16.00	+22.78	+19.50	+19.69	+31.67	+16.93	+31.34	+29.24	+27.29	+24.22	+24.22
Hit@10	+6.00	+18.45	+18.37	+14.64	+14.37	+12.67	+31.89	+19.20	+23.02	+26.66	+22.69	+33.89	+23.08	+36.67	+30.61	+31.06	+28.22	+28.22

Table 5: Comparison of neural (UltraQuery) vs symbolic query execution on LLM-generated queries from GTSQA on PPR subgraphs extracted from Wikidata (in %). Both executors receive identical queries generated by the LLM.

Metric	# hops = 1					# hops = 2						# hops = 3					# hops = 4	
	(1)	(1) (1)	(1) (1) (1)	(1) (1) (1) (1)	Avg	(2)	(2) (1)	((1) (1))	(2) (2)	((1) (1)) (1)	Avg	(3)	((2) (1))	(3) (1)	(2 (1) (1))	Avg	(4)	Avg
Neural executor (UltraQuery)																		
MRR	91.07	86.74	91.14	90.19	89.78	77.01	78.43	88.19	77.87	74.89	79.28	61.04	75.81	75.44	72.64	71.23	70.69	70.69
Hit@1	90.00	83.65	88.61	87.26	87.38	74.00	74.69	85.60	69.54	69.17	74.60	54.44	66.15	66.67	64.97	63.06	65.22	65.22
Hit@3	91.33	88.38	93.88	93.09	91.67	78.67	81.73	89.60	82.73	78.61	82.27	65.28	86.15	81.33	76.50	77.31	73.44	73.44
Hit@10	93.33	92.38	93.88	94.31	93.47	85.33	84.88	94.40	93.17	83.00	88.16	72.22	89.23	90.00	86.32	84.44	82.44	82.44
Symbolic executor																		
MRR	84.45	74.30	71.01	77.89	76.91	70.05	58.12	76.27	73.94	62.94	68.26	45.96	63.97	57.84	64.29	58.02	65.04	65.04
Hit@1	84.00	73.33	69.73	77.24	76.08	69.33	57.32	76.00	72.66	61.67	67.40	44.17	63.08	56.00	62.68	56.48	64.00	64.00
Hit@3	84.67	74.67	72.45	78.59	77.59	70.67	58.90	76.80	75.54	63.33	69.05	47.50	64.62	59.33	67.10	59.64	64.67	64.67
Hit@10	85.33	76.00	72.45	78.86	78.16	72.00	59.69	76.80	76.26	65.00	69.95	48.61	64.62	60.00	67.10	60.08	67.78	67.78
Improvement (Neural - Symbolic)																		
MRR	+6.62	+12.44	+20.13	+12.30	+12.87	+6.96	+20.31	+11.92	+3.93	+11.95	+11.01	+15.08	+11.84	+17.60	+8.35	+13.22	+5.65	+5.65
Hit@1	+6.00	+10.32	+18.88	+10.02	+11.30	+4.67	+17.37	+9.60	-3.12	+7.50	+7.20	+10.27	+3.07	+10.67	+2.29	+6.57	+1.22	+1.22
Hit@3	+6.66	+13.71	+21.43	+14.50	+14.07	+8.00	+22.83	+12.80	+7.19	+15.28	+13.22	+17.78	+21.53	+22.00	+9.40	+17.68	+8.77	+8.77
Hit@10	+8.00	+16.38	+21.43	+15.45	+15.32	+13.33	+25.19	+17.60	+16.91	+18.00	+18.21	+23.61	+24.61	+30.00	+19.22	+24.36	+14.66	+14.66

D QUERY LANGUAGE COMPARISON

Query ::= *Projection* | *Intersection*

Projection ::= (*Entity*, (*Relation*,)) (leaf)
| (*Query*, (*Relation*,)) (chained)

Intersection ::= (*Query*, *Query*)

Entity ::= "Q<digits>"
Relation ::= "P<digits>" | "P<digits>_inv"

Query ::= *Projection* | *Intersection*

Projection ::= *Entity* -> *Relation* (leaf)
| *Query* -> *Relation* (chained)

Intersection ::= AND(*Query* , *Query* [, *Query* ...])

Entity ::= "Q<digits>"
Relation ::= "P<digits>" | "P<digits>_inv"

```
(((TA, (P1_inv,)), (DL, (P2_inv,))), (P4,))     | AND(TA -> P1_inv, DL -> P2_inv) -> P4
```

Figure 4: Haskell-like grammar definitions for the old BetaE format (left) and our preferred DSL (right). The former uses nested tuples for projections and (binary) intersections. Our DSL uses infix notation with -> for projections and n-ary tuples for intersections, attempting to reduce bracket nesting. Below the horizontal line we show how the example from Figure 1 would transform (entities have been abbreviated). The maximum nesting depth reduces from 4 to 1.

Table 6: ULTRAQUERY per-class performance metrics on WikiKG2 subgraphs contained in GTSQA (in %). To generate ground-truth structured queries (i.e. queries provided by the oracle), given a subgraph with the answer as the root node, we perform a breadth-first traversal, identify leaves, and work upwards to construct the query.

Metric	# hops = 1					# hops = 2						# hops = 3					# hops = 4	
	(1)	(1) (1)	(1) (1) (1)	(1) (1) (1) (1)	Avg	(2)	(2) (1)	((1) (1))	(2) (2)	((1) (1)) (1)	Avg	(3)	((2) (1))	(3) (1)	(2 (1) (1))	Avg	(4)	Avg
With ground-truth queries																		
MRR	99.33	100.00	100.00	100.00	99.83	99.56	100.00	100.00	100.00	100.00	99.91	98.61	100.00	99.83	97.12	98.89	96.07	96.07
Hit@1	98.67	100.00	100.00	100.00	99.67	99.33	100.00	100.00	100.00	100.00	99.87	97.78	100.00	99.67	94.55	98.00	94.00	94.00
Hit@3	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	99.44	100.00	100.00	100.00	99.86	98.00	98.00
Hit@10	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	99.44	100.00	100.00	100.00	99.86	98.00	98.00
With LLM-generated queries																		
MRR	93.25	94.48	97.96	96.36	95.51	85.24	86.19	93.28	95.17	84.01	88.78	82.78	92.00	90.30	88.35	88.36	82.54	82.54
Hit@1	91.33	93.17	96.94	95.93	94.34	82.67	82.48	92.00	93.05	80.00	86.04	77.50	87.69	88.00	85.00	84.55	76.22	76.22
Hit@3	94.67	95.00	98.98	96.75	96.35	87.33	87.99	93.60	97.60	86.17	90.54	87.87	93.85	90.67	92.27	91.16	88.22	88.22
Hit@10	96.00	97.78	98.98	96.75	97.38	88.67	92.13	96.80	97.84	93.33	93.75	90.65	100.00	96.00	93.64	95.07	92.22	92.22
Improvement (LLM generated queries - GT queries)																		
MRR	-6.08	-5.52	-2.04	-3.64	-4.32	-14.32	-13.81	-6.72	-4.83	-15.99	-11.13	-15.83	-8.00	-9.53	-8.77	-10.53	-13.53	-13.53
Hit@1	-7.34	-6.83	-3.06	-4.07	-5.33	-16.66	-17.52	-8.00	-6.95	-20.00	-13.83	-20.28	-12.31	-11.67	-9.55	-13.45	-17.78	-17.78
Hit@3	-5.33	-5.00	-1.02	-3.25	-3.65	-12.67	-12.01	-6.40	-2.40	-13.83	-9.46	-11.57	-6.15	-9.33	-7.73	-8.69	-9.78	-9.78
Hit@10	-4.00	-2.22	-1.02	-3.25	-2.62	-11.33	-7.87	-3.20	-2.16	-6.67	-6.25	-8.79	0.00	-4.00	-6.36	-4.79	-5.78	-5.78

E NEURAL VS SYMBOLIC

As discussed in the main text, we present the full table of results for WikiKG2. The neural executor provides a clear advantage over symbolic execution, with average gains (computed by averaging the per-hop "Avg" columns in the tables) of 18.58% (MRR), 15.40% (Hit@1), 21.14% (Hit@3), and 24.09% (Hit@10).

In addition to full results on WikiKG2 in Table 4, in Table 5 we provide the results of neural vs symbolic query execution performance on PPR subgraphs extracted from Wikidata using seed entities available in Cattaneo et al. (2025). As already highlighted in the main text of the paper, the neural query executor consistently outperforms symbolic execution across the considered metrics and classes of queries (only exception on Wikidata being Hit@1 for class (2)(2), where the symbolic executor has a gain of 3.12%). Compared to WikiKG2 subgraphs, the absolute performance is generally lower on Wikidata, which can be the result of both an increased amount of noise in the queries (as a result of the larger set of entities and relation types available in Wikidata) and possibly noise in the extracted subgraph due to the additional PPR extraction phase. This said, while performance degrade to some extent, they still appear good even in this more challenging scenario, and the relative advantage of neural over symbolic execution remains substantial.

F GROUND-TRUTH QUERIES VS LLM GENERATED QUERIES ON WIKIKG2

In Table 6 we compare performance of our chosen neural query executor (ULTRAQUERY) when processing GT queries vs LLM-generated queries on the WikiKG2 subgraphs of GTSQA Cattaneo et al. (2025). As we can see, the hop-averaged performance gap ("Avg" columns in the table) between ground-truth and LLM-generated queries appears to be generally growing with query complexity, starting at 4.32% of MRR for 1-hop subgraphs, and ending with 13.53% MRR for 4-hops (similar trends also appear for the other considered metrics). This highlights the challenge of query generation for complex multi-hop reasoning.

G ULTRAG WITH DIFFERENT LLMS

In Table 7 we provide a comparison of ULTRAG-OTS performance with different LLMs. For this analysis, we used GTSQA using PPR subgraphs extracted from Wikidata with ground-truth seed entities.

H RUNTIME COSTS ON WIKIKG2 AND WIKIDATA

In Table 8 we show average API costs per query, average non-API times, and average number of input / output tokens processed / generated by the LLM (GPT-5) on KGQAGen subgraphs extracted from Wikidata. On this dataset, ULTRAG-OTS is 15x faster than RoG, 122x faster than GNN-RAG,

Table 7: Comparison of ULTRAG-OTS with different LLMs on GTSQA (Wikidata) using ground-truth seed entities (in %). PPR graphs have up to 30,000 nodes. With $A \rightarrow B$, we describe a setting where we used LLM A for query generation and B for arbitration.

LLM	Hits	Recall	F1	API cost
ULTRAG				
- GPT-5$\times$2	86.74	84.47	82.08	0.017$
- GPT-5$\rightarrow$GPT-5-mini	82.37	77.83	78.41	0.009$
- GPT-5-mini$\rightarrow$GPT-5	80.71	78.19	75.72	0.011$
- DeepSeek-reasoner$\times$2	75.77	72.40	72.05	0.005$
- GPT-5-mini$\times$2	75.71	71.08	71.60	0.004$

Table 8: Average efficiency per query on PPR subgraphs extracted for KGQAGen-10K from Wikidata with ground-truth seed nodes. All models use GPT-5 as reasoning LLM. Non-API running time measured on GH200 chips, excluding API calls and PPR computation.

KGQAGen-10K	API cost	Non-API time (s)	Input tokens	Output tokens	Cache hit %
RoG	0.008$	3.1 ± 0.9	1.4K	1.4K	0
GNN-RAG	0.008$	25.8 ± 4.4	1.4K	1.4K	0
SubgraphRAG (200)	0.009$	27.5 ± 3.9	3.1K	1.5K	0
ULTRAG-OTS	0.019$	0.21 ± 0.1	69K	2.8K	96.06
- GPT-5-mini	0.004$	0.15 ± 0.1	69K	2.4K	95.53

and 130x faster than SubgraphRAG, in terms of non-API time per query. In terms of number of input tokens, ULTRAG-OTS processes between 22x and 50x more tokens compared to the baselines (again due to the presence of relation types in the input prompt and two API calls). However, as it was the case for GTSQA, 96% of these tokens are cached by GPT-5. Interestingly, average number of output tokens and API costs are $\sim$2x larger for ULTRAG-OTS compared to the baselines, which appear more efficient on this dataset. We note however, that state-of-the-art performance (Hits$=86.07\%$, Recall$=84.45\%$, F1$=83.91\%$) can be still achieved on KGQAGen with ULTRAG-OTS, at a fraction of the cost ($0.0036), by using GPT-5-mini for both generation and arbitration in place of GPT-5 (ULTRAG-OTS implemented with GPT-5-mini is actually the cheapest solution in our comparison). This highlights that, while KGQAGen questions requires larger amounts of output tokens for producing valuable results with our methodology, questions there appear to be simpler: ULTRAG-OTS with lighter and less skillful LLMs can still outperform baselines that use GPT-5. As it was the case for GTSQA, we also highlight that in an end-to-end pipeline, no additional cost would be required for mention extraction with ULTRAG-OTS, while all the considered baselines would incur extra costs for this additional step[6].

[6]If GPT-5 was used for this, all baselines would incur an additional cost of 0.004$ per query, while also showing an increase of 780 in the average number of output tokens produced.