# LogosKG: Hardware-Optimized Scalable and Interpretable Knowledge Graph Retrieval

Anonymous ACL submission


## Abstract

Knowledge graphs (KGs) are increasingly integrated with large language models (LLMs) to provide structured, verifiable reasoning. A core operation in this integration is multi-hop retrieval, yet existing systems struggle to balance efficiency, scalability, and interpretability. We introduce LogosKG, a novel, hardware-aligned framework that reformulates $k$-hop retrieval as sparse-matrix operations by decomposing graphs into subject, object, and relation matrices. To scale to billion-edge graphs, LogosKG integrates degree-aware partitioning, cross-graph routing, and on-demand caching. Experiments show substantial efficiency gains over CPU and GPU baselines without loss of retrieval fidelity. Beyond system performance, a case study demonstrates how LogosKG enables large-scale, evidence-grounded analysis of how KG topology, such as hop distribution and connectivity, governs the alignment between structured biomedical knowledge and LLM diagnostic reasoning, thereby opening the door for high-performance KG retrieval and next-generation KG-LLM integration.


## 1 Introduction

For decades, knowledge graphs (KGs) have served as a foundation for structured knowledge representation, linking concepts through relations across domains such as social networks (Cai et al., 2023), biomedicine (Lu et al., 2025), and recommendation systems (Wang et al., 2025a). With the rise of large language models (LLMs), KGs have gained renewed importance as an external symbolic knowledge source that complements LLMs' statistical reasoning in tasks like retrieval-augmented generation (RAG) (Liu et al., 2025; Sharma et al., 2024), knowledge verification (Pham et al., 2025a; Dammu et al., 2024), and reasoning (Wang et al., 2025b; Wu et al., 2025). In high-stakes domains such as medical diagnosis, where LLM reliability directly affects patient safety, KGs are increasingly used to ground model predictions in verified biomedical knowledge (Jia et al., 2024; Zuo et al., 2025; Rezaei et al., 2025).

| Method (year) | Matrix-based | Scala-bility | Path Reconstruction | Device |
|---|---|---|---|---|
| Neo4j (Neo4j, Inc., 2025) | ✗ | ✓ | ✓ | CPU |
| TigerGraph (Deutsch et al., 2019) | ✗ | ✓ | ✓ | CPU |
| GraphBLAS (Kepner et al., 2016) | ✓ | ✗ | ✗ | CPU |
| igraph (Csardi and Nepusz, 2006) | ✗ | ✗ | ✓ | CPU |
| NetworkX (Hagberg et al., 2008) | ✗ | ✗ | ✓ | CPU |
| graph-tool (Peixoto, 2014) | ✗ | ✗ | ✓ | CPU |
| SNAP (Leskovec and Sosič, 2016) | ✗ | ✓ | ✓ | CPU |
| cuGraph (RAPIDS AI, 2025) | ✓ | ✗ | ✓ | GPU |
| DGL (Wang et al., 2019) | ✗ | ✓ | ✗ | GPU |
| PyG (Fey and Lenssen, 2019) | ✗ | ✓ | ✗ | GPU |
| **LogosKG (ours, 2025)** | ✓ | ✓ | ✓ | CPU/GPU |

Table 1: Comparison of retrieval systems and libraries. Matrix-based indicates the use of linear-algebra primitives for retrieval. *Note:* Entries reflect default design; unavailable features can be achieved with additional code or preprocessing, at the cost of extra time and memory.

A fundamental operation of KGs is *multi-hop retrieval*, which connects distant concepts through intermediate entities and relations. Traditional graph traversal algorithms such as depth-first search (DFS) and breadth-first search (BFS) can handle small graphs but quickly become infeasible as graph size grows, with cost scaling as $O(|V|+|E|)$ and reachable entities growing exponentially with hop depth. Memory is another major bottleneck: for instance, biomedical KGs used in our experiments (**UMLS** (Bodenreider, 2004), 3.4M triplets; **PubMedKG** (Xu et al., 2020, 2025), 500M triplets) occupy 1.5 GB and 23.5 GB of memory even before traversal. A two-hop expansion from a high-degree concept in UMLS (on average ≈33k 1-hop neighbors) can involve over $10^9$ reachable edges, consuming tens of gigabytes of memory to materialize adjacency information. Such exponential growth in traversal and storage defines the *core systems challenge* in scaling multi-hop retrieval, forcing prior work to operate on limited subgraphs (Gao et al., 2025; Chang et al., 2020).

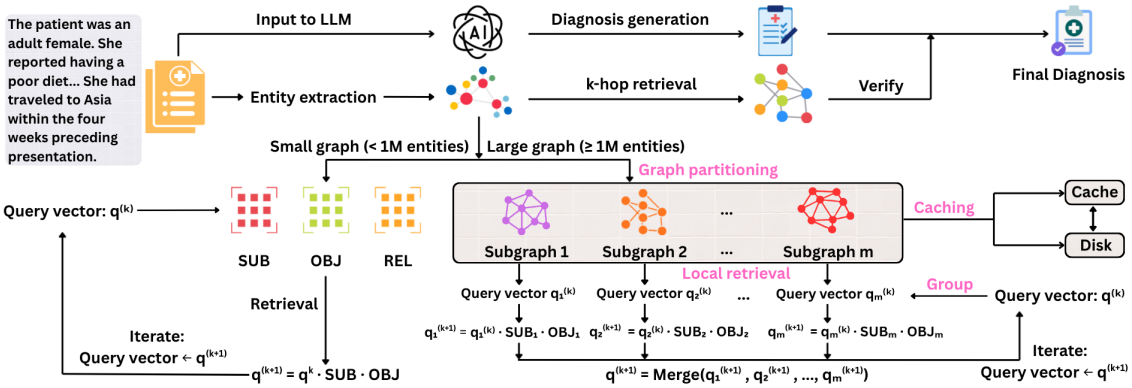


Figure 1: Overview of the LogosKG retrieval framework. LogosKG adopts a linear algebra–based retrieval method that supports both small and large knowledge graphs. For large graphs, LogosKG incorporates graph partitioning, cross-graph routing, and on-demand caching to enable efficient multi-hop retrieval across distributed subgraphs. The retrieved evidence can provide knowledge-grounded support for medical diagnosis generation.

Numerous systems have been developed to improve retrieval efficiency (Table 1) emphasizing three dimensions: (1) *matrix-based* graph representation replacing pointer-based data structures with matrix or tensor operations for vectorized computation that maps naturally onto parallel hardware such as multi-core CPUs and GPUs (e.g., GraphBLAS-based systems); (2) *scalability*, achieved through partitioning and caching to handle billion-edge graphs (e.g., cuGraph, DGL, PyG); and (3) *path reconstruction* enabling interpretable reasoning by recovering intermediate entities and relations (e.g., Neo4j, TigerGraph). However, most systems optimize only one or two of these aspects and often rely on distributed infrastructures rather than single-device efficiency.

**Our Approach.** To achieve scalable and interpretable KG retrieval, we present **LogosKG**, a *novel* hardware-aligned framework that rethinks graph traversal at system level. As shown in Figure 1, LogosKG reformulates retrieval as sparse-matrix operations, building on the algorithmic foundation of (Cohen et al., 2020) and extending it to a fully hardware-optimized design. This shift eliminates irregular memory access, allowing contiguous data layouts and kernel-level parallelism on CPUs and GPUs. For large graphs, LogosKG implements degree-aware partitioning and cross-graph routing guided by memory-locality principles, while on-demand caching further reduces I/O overhead, achieving $\mathcal{O}(|\mathcal{E}| \log |\mathcal{E}| + |\mathcal{T}|)$ complexity with degree-aware partitioning and amortized batch execution. Finally, LogosKG stores intermediate entities and relations that enables path reconstruction. Together, these innovations transform a theoretical formulation into a practical and scalable framework for large-scale KG retrieval.

We evaluate LogosKG against existing libraries and systems (Table 1) across efficiency, scalability, and retrieval correctness. Experiments focus on large, semantically rich biomedical KGs, which present realistic challenges for dense connectivity, heterogeneous relations, and interpretable reasoning. Although the experiments use biomedical data, LogosKG is domain-agnostic and applicable to any large structured graph.

We further conduct a case study on a rapidly growing research direction: integrating knowledge graphs with LLMs (KG+LLMs) for evidence-grounded reasoning and knowledge verification (Dammu et al., 2024; Pham et al., 2025b). Because LogosKG performs symbolic, sparse-matrix retrieval without task-specific learning, it exposes the raw relational structure of KGs, enabling large-scale analysis of how existing biomedical graphs align with LLM diagnostic reasoning. This study demonstrates the feasibility of evidence-grounded reasoning at scale and reveals how KG topology, such as hop distribution and connectivity, shapes the alignment between structured knowledge and model reasoning.

Our work makes the three contributions:

- **Framework:** We present LogosKG, a hardware-aligned, general-purpose system for scalable multi-hop retrieval (§3).
- **Evaluation:** We benchmark efficiency, scalability, and retrieval fidelity across CPU and GPU baselines (§5).
- **Case Study:** We use LogosKG to examine how biomedical KGs align with LLM diagnostic reasoning (§6).

LogosKG establishes a foundation for scal-

able retrieval and highlights a broader open challenge: harmonizing the topology of symbolic KGs with neural reasoning in the next generation of KG+LLM systems.

## 2 Related Work

### 2.1 Systems and Libraries for KG Retrieval

Existing systems and libraries can be classified by their architecture, ranging from database-backed engines and computation-focused libraries to graph analysis tools and GPU-based frameworks. **Database-backed engines** such as Neo4j and TigerGraph (Deutsch et al., 2019) provide expressive query languages and optimized storage for large graphs, but incur high setup and system overheads. **Computation-focused libraries** such as GraphBLAS (Kepner et al., 2016) use sparse matrix operations to make multi-hop retrieval efficient, yet lack native support for full path reconstruction needed in tasks such as biomedical reasoning. **Graph analysis tools** like igraph (Csardi and Nepusz, 2006), NetworkX (Hagberg et al., 2008), graph-tool (Peixoto, 2014), and SNAP (Leskovec and Sosič, 2016) offer flexible APIs but rely on traditional traversal algorithms that do not scale. Finally, **GPU-based frameworks** such as cuGraph, DGL (Wang et al., 2019), and PyG (PyTorch Geometric) (Fey and Lenssen, 2019) enable fast single-machine computation and integrate well with deep learning pipelines, though they primarily target graph analytics rather than retrieval.

### 2.2 KGs as Verifiers for LLMs

LLMs are prone to hallucinations and unverified claims, motivating research on KG-based verification and fact-checking. Recent methods leverage KGs to ground textual claims in structured evidence and construct multi-hop reasoning chains that link claims to supporting facts. Examples include FactKG (Kim et al., 2023) and GraphCheck (Chen et al., 2025) for long-context fact-checking, ClaimVer (Dammu et al., 2024) and Verify-in-the-Graph (Pham et al., 2025b) for explainable evidence attribution and entity disambiguation, and GraphFC (Huang et al., 2025) and FactCheck (Shami et al., 2025) for integrating graph reasoning with LLM-based verification. In contrast, our work exposes the raw topology of KGs and reveal a structural gap between symbolic knowledge organization and neural reasoning.

## 3 LOGOSKG

### 3.1 Problem Statement

**Knowledge Graph.** A *KG* is defined as a directed multi-relational graph

$$\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{T}),$$

where $\mathcal{E}$ is the set of entities, $\mathcal{R}$ is the set of relation types, and $\mathcal{T} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ is the set of observed triples. Each triple $(e_s, r, e_o) \in \mathcal{T}$ denotes that subject entity $e_s \in \mathcal{E}$ is linked to object entity $e_o \in \mathcal{E}$ through relation $r \in \mathcal{R}$.

**$k$-hop Retrieval.** Given a query $\mathbf{q} \in \mathcal{E}$ consisting of a set of entities, the goal of $k$-hop retrieval is to identify all entities reachable from any $e_q \in \mathbf{q}$ within $k$ relational steps. In practice, querying large KGs can be time-consuming. The task is to retrieve $k$-hop entities in large KGs while keeping query latency as low as possible.

### 3.2 KGs Decomposition

To support efficient $k$-hop retrieval, a KG could be represented using three sparse incidence matrices encoding subjects, objects, and relations (Cohen et al., 2020). This decomposition transforms heavy graph traversal into lightweight sparse matrix operations, as shown in Figure 1.

**Subject matrix.** The *subject matrix* $\mathbf{SUB} \in \{0,1\}^{|\mathcal{E}| \times |\mathcal{T}|}$ is defined as:

$$[\mathbf{SUB}]_{i,t} = \begin{cases} 1, & \text{if } e_i \text{ is the subject of } t, \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

where $e_i$ denotes an entity and $t$ a triplet, with rows as entities and columns as triplets.

**Object matrix.** The *object matrix* $\mathbf{OBJ} \in \{0,1\}^{|\mathcal{T}| \times |\mathcal{E}|}$ is defined as:

$$[\mathbf{OBJ}]_{t,j} = \begin{cases} 1, & \text{if } e_j \text{ is the object of } t, \\ 0, & \text{otherwise.} \end{cases} \quad (2)$$

where $e_j$ is an entity and $t$ a triplet, with rows as triplets and columns as entities.

**Relation matrix.** The *relation matrix* $\mathbf{REL} \in \{0,1\}^{|\mathcal{T}| \times |\mathcal{R}|}$ encodes the relation type of each triplet, and is defined as:

$$[\mathbf{REL}]_{t,r} = \begin{cases} 1, & \text{if } t \text{ uses relation } r, \\ 0, & \text{otherwise.} \end{cases} \quad (3)$$

where $t$ denotes a triplet and $r$ a relation type. Each row corresponds to a triplet and each column to a relation, enabling recovery of relation paths in multi-hop retrieval.

### 3.3 Efficient Retrieval

**One-hop retrieval.** Define query vector as $\mathbf{q}^{(0)} \in \{0,1\}^{1\times|\mathcal{E}|}$, where $\mathbf{q}^{(0)}[i] = 1$ if entity $e_i$ is included in the query. Multiplying by $\mathbf{SUB}$ activates the triplets starting from these entities, and multiplying further by $\mathbf{OBJ}$ yields the reachable entities:

$$\begin{aligned} \mathbf{q}^{(0)} \cdot \mathbf{SUB} &\mapsto \mathbf{t}^{(1)}(\text{active triplets}), \\ \mathbf{t}^{(1)} \cdot \mathbf{OBJ} &\mapsto \mathbf{q}^{(1)}(\text{active entities}). \end{aligned}$$

Formally,

$$\mathbf{q}^{(1)} = \mathbf{q}^{(0)} \cdot \mathbf{SUB} \cdot \mathbf{OBJ}. \tag{4}$$

**$k$-hop retrieval.** Repeating this operation $k$ times gives the $k$-hop neighborhood:

$$\mathbf{q}^{(k)} = \mathbf{q}^{(0)} \cdot \big(\mathbf{SUB} \cdot \mathbf{OBJ}\big)^k, \tag{5}$$

where $\mathbf{q}^{(k)} \in \{0,1\}^{1\times|\mathcal{E}|}$ and nonzero entries indicate the entities reachable exactly at $k$ steps.

**Path reconstruction.** At any hop $k$, we retrieve the activated triplets by

$$\mathbf{t}^{(h)} = \mathbf{q}^{(h-1)} \cdot \mathbf{SUB}. \tag{6}$$

The set of nonzero indices is

$$T_h = \{\, t \mid \mathbf{t}^{(h)}[t] = 1 \,\}. \tag{7}$$

For each $t \in T_h$, we retrieve the subject entity, object entity, and relation of the triple directly from the incidence matrices:

$$\begin{aligned} s_h &= \{\, s \mid [\mathbf{SUB}]_{i,t} = 1 \,\}, \\ r_h &= \{\, r \mid [\mathbf{REL}]_{t,r} = 1 \,\}, \\ e_h &= \{\, o \mid [\mathbf{OBJ}]_{t,j} = 1 \,\}. \end{aligned} \tag{8}$$

We build the path $s_h \xrightarrow{r_h} e_h$, and if $e_h$ also appears as a subject, we extend it at hop $h{+}1$. Repeating this yields complete paths of length $k$.

### 3.4 Scalable Retrieval

Large-scale KGs can contain hundreds of millions or even billions of triplets, making it infeasible to load the entire graph into memory. To address this, we design a hierarchical system with degree-aware partitioning, cross-graph retrieval, and on-demand caching. Globally, queries are distributed across subgraphs; locally, retrieval within each subgraph uses sparse matrix operations.

Specficially, a KG is divided into balanced subgraphs processed independently, with a mapping table tracking each entity's subgraph assignment. During retrieval, queries are routed to their respective subgraphs for local multi-hop retrieval (§3.2), and results are merged. On-demand caching loads only the required subgraphs into memory, keeping the rest on disk. This step is the key to efficient retrieval over billion-scale KG on limited hardware.

**Algorithm 1** Cross-graph $k$-Hop Retrieval with On-Demand Caching

**Require:** Initial query vector $\mathbf{q}^{(0)}$, hop count $k$, subgraphs $\{\mathcal{G}_i\}_{i=1}^{M}$ with $(\mathbf{SUB}_i, \mathbf{OBJ}_i, \mathbf{REL}_i)$, metadata $P$, cache $C$
**Ensure:** Query vector $\mathbf{q}^{(k)}$

```
for ℓ = 0 to k − 1 do
    Route nonzero entries of q^(ℓ) to subgraphs via P
    Initialize q^(ℓ+1) ← 0
    for all subgraphs G_i with active entities do
        Load G_i into cache C if absent (evict LRU if full)
        Restrict q^(ℓ) to local subvector q_i^(ℓ)
        Compute local update q_i^(ℓ+1) = q_i^(ℓ) · SUB_i · OBJ_i
        Map q_i^(ℓ+1) back to global indices
        q^(ℓ+1) ← q^(ℓ+1) ∨ q_i^(ℓ+1)
    end for
end for
return q^(k)
```

**Degree-aware graph partitioning.** A KG can be partitioned into disjoint subgraphs $\{\mathcal{G}_i = (\mathcal{E}_i, \mathcal{R}_i, \mathcal{T}_i)\}_{i=1}^{m}$, where each $\mathcal{G}_i$ contains a subset of entities, relations, and triplets. Subject entities are assigned in a degree-aware manner so that high-degree subject entities are distributed evenly, avoiding bottlenecks caused by imbalance. To preserve KG structures, all triplets sharing the same subject entity are assigned to the same subgraph. The procedure runs in $O(|\mathcal{E}|\log|\mathcal{E}| + |\mathcal{T}|)$ time, including finding relations and sorting entities by degrees in $O(|\mathcal{E}|\log|\mathcal{E}| + |\mathcal{T}|)$ time and assigning triplets to subgraphs in $O(|\mathcal{T}|)$ time. We also maintain a metadata map $P : \mathcal{E} \to \{1,\ldots,m\}$ that records the partition assignment of each subject entity for efficient cross-partition retrieval. For each subgraph $\mathcal{G}_i$, we build local incidence matrices $\mathbf{SUB}_i$, $\mathbf{OBJ}_i$, and $\mathbf{REL}_i$ to support efficient retrieval within the partition.

**Cross-subgraph retrieval.** Given a query entity vector $\mathbf{q}^{(0)}$, we first group entities by their assigned subgraphs using the metadata map $P$. For each subgraph $\mathcal{G}_i$, we build a local query vector $\mathbf{q}_i^{(0)}$ representing the subset of entities assigned to $\mathcal{G}_i$. Retrieval is then carried out independently within each subgraph using its local incidence matrices $\mathbf{SUB}_i$, $\mathbf{OBJ}_i$, and $\mathbf{REL}_i$. For any hop $k$, the update step follows the same principle as in the single-graph case:

$$\mathbf{t}_i^{(k)} = \mathbf{q}_i^{(k)} \cdot \mathbf{SUB}_i, \tag{9}$$

$$\mathbf{q}_i^{(k+1)} = (\mathbf{q}_i^{(k)} \cdot \mathbf{SUB}_i) \cdot \mathbf{OBJ}_i. \quad (10)$$

Entities $\mathbf{q}_i^{(k+1)}$ from all subgraphs are merged into a global query vector $\mathbf{q}^{(k+1)}$, then redistributed for the next hop. Repeating this $k$ times reconstructs the $k$-hop neighborhood while limiting computation to accessed subgraphs.

**On-demand caching.** This mechanism makes sure subgraphs are loaded only when required. It avoids memory blow-up but introduces additional I/O overhead, since not all subgraphs needed for a query are guaranteed to be in memory.

An in-memory cache of fixed capacity $n$ is implemented and capable of storing at most $n$ subgraphs, managed under a least-recently-used (LRU) policy. When a subgraph is requested, it is reused if present in the cache, or otherwise loaded from disk and inserted by evicting the least recently accessed subgraph. Each subgraph is stored on disk in a sparse format containing the incidence matrices. Cache hits require only sparse matrix multiplications, while cache misses incur the additional cost of disk I/O.

Let $h$ denote the cache hit rate, $\tau_{\text{mm}}$ the time for in-memory sparse matrix multiplication, and $\tau_{\text{io}}$ the average disk load time. The expected retrieval cost per subgraph is

$$\mathbb{E}[\tau_{\text{retrieval}}] = h \cdot \tau_{\text{mm}} + (1-h) \cdot (\tau_{\text{mm}} + \tau_{\text{io}}). \quad (11)$$

We optimize batch processing for better cache efficiency by grouping queries with similar subgraph requirements. Each query's subgraphs are identified via $P$, queries are reordered for joint processing, and results are restored afterward. This improves temporal locality, allowing subgraph reuse and reducing cache misses and I/O overhead.

**Overall pipeline.** The retrieval process begins with an initial query vector $\mathbf{q}^{(0)}$. At each hop $k$, we (i) map the active entities in $\mathbf{q}^{(h)}$ to their subgraphs using $P$, (ii) perform local retrieval within each subgraph using incidence matrices, (iii) merge the local results into the next global query vector $\mathbf{q}^{(h+1)}$, and (iv) repeat for $k$ hops. The procedure is summarized in Algorithm 1.

### 3.5 Implementation Details

We implement LogosKG with three computational backends: Numba, SciPy, and Torch, where the Torch backend supports both CPU and GPU execution. Experiments were conducted on a HIPPA-compliant Linux server with Ubuntu 22.04 system, dual AMD EPYC 9454 48-Core CPUs (192 threads), 256 GB RAM, and two NVIDIA H100 NVL GPUs (94 GB VRAM each). Source code is available[1].

## 4 Dataset and Setup

Our results are organized into two parts: (1) LogosKG system evaluation, focusing on retrieval efficiency, scalability, and correctness; and (2) a KG+LLM case study, evaluating how knowledge graphs support diagnostic reasoning by LLMs.

Three biomedical knowledge graphs are examined here: UMLS, as introduced earlier, a large-scale biomedical ontology with 133 semantic types and 54 relations over 5M concepts; PubMed Knowledge Graph (PKG) (Xu et al., 2020, 2025), connecting over ≈485M entities such as authors, publications, and institutions; and PrimeKG (Chandak et al., 2023) integrates 20 high-quality biomedical resources to describe 17,080 diseases with ≈4M relationships.

We include two clinical datasets that connect the KGs under study to real-world diagnostic applications. ProbSum (Gao et al., 2023) contains de-identified clinical notes with findings and diagnoses, supplying entity mentions that serve as realistic query inputs for retrieval evaluation. DDXPlus (Fansi Tchango et al., 2022) offers large-scale symptom–diagnosis pairs and is used in the KG+LLM case study to assess how KG structure aligns with LLM-predicted diagnoses. Retrieval experiments use UMLS and PKG with Probsum, while the KG+LLM case study uses UMLS and PrimeKG with Probsum and DDXPlus.

For the second part of the paper where we explore KG+LLMs, we include the following instruction-tuned, widely used LLMs: Qwen-2.5-7B-Instruct (Qwen et al., 2025), Llama-3.1-8B-Instruct (Dubey et al., 2024), GPT-o3-mini (OpenAI, 2025). While main results are reported on these LLMs, we additionally include Mistral-7B-Instruct-v0.3 (Jiang et al., 2023) and Gemma-7b-it (Team et al., 2024) into appendix due to space limit. All open-sourced LLMs are running on the GPU server. We use GPT-o3-mini via HIPPA-compliant Microsoft Azure OpenAI endpoint, thereby complying with the ProbSum Data Use Agreement for MIMIC-III derived content (Johnson et al., 2016).

[1] https://anonymous.4open.science/r/LogosKG-9ADA

| Method | Hop 1 (2000 ms) | | Hop 2 (4000 ms) | | Hop 3 (6000 ms) | | Hop 4 (8000 ms) | | Hop 5 (10000 ms) | |
|---|---|---|---|---|---|---|---|---|---|---|
| | QT (ms) | TR (%) | QT (ms) | TR (%) | QT (ms) | TR (%) | QT (ms) | TR (%) | QT (ms) | TR (%) |
| *Baselines* | | | | | | | | | | |
| NetworkX | **0.21** | **0.00** | **5.47** | **0.00** | 93.92 | 0.00 | 621.95 | 0.00 | 1511.28 | 0.00 |
| igraph | 1.15 | 0.00 | 26.13 | 0.00 | 309.90 | 0.00 | 837.12 | 0.00 | 580.91 | 0.00 |
| graph-tool | > 1458.79 | 45.33 | > 1900.01 | 10.00 | > 2141.41 | 2.00 | > 2396.95 | 0.67 | > 2306.34 | 0.67 |
| SNAP | 1.80 | 0.00 | 10.02 | 0.00 | 115.00 | 0.00 | 378.81 | 0.00 | 446.15 | 0.00 |
| GraphBLAS | 3.03 | 0.00 | 43.64 | 0.00 | 291.89 | 0.00 | 528.07 | 0.00 | 415.43 | 0.00 |
| Neo4j | > 923.86 | 11.33 | > 1946.43 | 20.00 | > 5739.02 | 95.33 | > 8000.00 | 100 | > 10000.00 | 100 |
| cuGraph | > 722.66 | 3.33 | > 919.30 | 0.67 | 1204.00 | 0.00 | 1504.82 | 0.00 | 1616.55 | 0.00 |
| DGL | > 966.65 | 10.00 | > 989.91 | 0.67 | 1042.25 | 0.00 | 1121.70 | 0.00 | 1099.26 | 0.00 |
| PyG | 249.66 | 0.00 | 271.46 | 0.00 | 365.90 | 0.00 | 646.96 | 0.00 | 735.34 | 0.00 |
| *LogosKG family* | | | | | | | | | | |
| LogosKG (Numba) | 12.28 | 0.00 | 28.72 | 0.00 | 77.65 | 0.00 | 140.07 | 0.00 | 204.25 | 0.00 |
| LogosKG (SciPy) | 13.81 | 0.00 | 34.97 | 0.00 | 104.21 | 0.00 | 289.61 | 0.00 | 677.23 | 0.00 |
| LogosKG (Torch-CPU) | 526.55 | 0.00 | 884.89 | 0.00 | 1321.05 | 0.00 | 1803.18 | 0.00 | 2207.25 | 0.00 |
| LogosKG (Torch-GPU) | 6.00 | 0.00 | 14.40 | 0.00 | **43.07** | **0.00** | **77.73** | **0.00** | **101.05** | **0.00** |
| LogosKG (Large-Numba) | 4.76 | 0.00 | 25.52 | 0.00 | 226.83 | 0.00 | 1411.72 | 0.00 | > 4085.72 | 4.00 |
| LogosKG (Large-SciPy) | 7.64 | 0.00 | 63.93 | 0.00 | 324.95 | 0.00 | > 1660.91 | 0.67 | > 4532.75 | 5.33 |
| LogosKG (Large-Torch-CPU) | 126.61 | 0.00 | > 1205.50 | 0.67 | > 2551.68 | 5.33 | > 4836.16 | 22.00 | > 7467.56 | 48.67 |
| LogosKG (Large-Torch-GPU) | 13.75 | 0.00 | 103.50 | 0.00 | 412.38 | 0.00 | 1634.93 | 0.00 | > 4482.14 | 6.00 |

Table 2: Retrieval efficiency comparison across hops. Timeout limits are fixed at 2000, 4000, 6000, 8000, and 10000 ms for 1–5 hops, respectively. All methods are run under the same CPU workload for fair comparison. For each hop, the best method is shown in bold and the second-best is underlined.

| Factor | Value | QT (ms) | Loads | Evicts |
|---|---|---|---|---|
| *Exp. 1: hops* (Numba, cache size $n = 16$, batch size=50) | | | | |
| hops | 1 | 3410.90 | 16 | 0 |
| hops | 2 | 1610.78 | 16 | 0 |
| hops | 3 | 6114.69 | 16 | 0 |
| hops | 4 | 19592.08 | 16 | 0 |
| hops | 5 | 62726.25 | 16 | 0 |
| *Exp. 2: batch size* (Numba, cache size $n = 16$, hops $k = 2$) | | | | |
| batch size | 1 | 100912.09 | 12 | 0 |
| batch size | 10 | 5489.45 | 16 | 0 |
| batch size | 25 | 1365.09 | 16 | 0 |
| batch size | 50 | 1499.39 | 16 | 0 |
| batch size | 100 | 1444.68 | 16 | 0 |
| batch size | 150 | 1483.49 | 16 | 0 |
| *Exp. 3: cache size* (Numba, hops $k = 2$, batch size=50) | | | | |
| cache size | 1 | 441870.19 | 3010 | 3009 |
| cache size | 2 | 419436.59 | 2918 | 2916 |
| cache size | 4 | 384086.42 | 2659 | 2655 |
| cache size | 8 | 304066.00 | 1967 | 1959 |
| cache size | 16 | 4037.69 | 16 | 0 |
| *Exp. 4: backend* (cache size $n = 16$, hops $k = 2$, batch size=50) | | | | |
| backend | Numba | 4143.16 | 16 | 0 |
| backend | Scipy | 3889.86 | 16 | 0 |
| backend | Torch-CPU | 245311.21 | 16 | 0 |
| backend | Torch-GPU | 6409.32 | 16 | 0 |

Table 3: Scalability of LogosKG-Large on the PKG across hops, batch sizes, cache sizes, and backends.

## 5 LogosKG System Evaluation

We evaluate LogosKG and baselines along three dimensions: **Accuracy** measures whether retrieved entities match ground truth across hops; **Efficiency** is assessed by query time and timeout rates; **Scalability** examines how performance changes of LogosKG under different settings, evaluating the ability to handle large KGs.

**Baselines** We compare LogosKG against a diverse set of widely used graph retrieval systems, including database engines (Neo4j), graph analysis toolkits (igraph, NetworkX, graph-tool, SNAP), matrix-based libraries (GraphBLAS and its Python implementation), and GPU-accelerated frameworks (cuGraph, DGL, PyG). Together, these baselines cover the major design paradigms for knowledge graph retrieval, allowing comprehensive evaluation of efficiency, scalability, and retrieval fidelity.

**Accuracy** On a set of queries, we compare the retrieved results from LogosKG family with CPU- and GPU-based baselines across 1–5 hops. This result is measured by Jaccard similarity, with 1.0 denoting exact agreement. All comparisons yield a perfect Jaccard score of 1.0, confirming that retrieval is deterministic given the KG structure. We neglect the detailed results here and present them in Table 6. This validates LogosKG as a faithful and reliable replacement for existing engines.

**Efficiency** For each method, we report the average **query time (QT)** across all test samples and the **timeout rate (TR)** per hop, defined as the proportion of queries exceeding the hop-specific limit. QT reflects the latency of a method, while TR captures its responsiveness under practical constraints. These metrics provide a clear view of efficiency between LogosKG and the baselines.

Table 2 reports QT and TR for 1–5 hops. Classical CPU libraries (NetworkX, igraph, SNAP,

| Model | Method | UMLS KG | | | | | | PrimeKG KG | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | ProbSum | | | DDXPlus | | | ProbSum | | | DDXPlus | | |
| | | P | R | F1 | P | R | F1 | P | R | F1 | P | R | F1 |
| Qwen-2.5-7B-Instruct | Baseline | 27.82 | 19.90 | 20.30 | 80.38 | 84.07 | **80.05** | 23.84 | 25.07 | **20.70** | 81.82 | 84.57 | **80.98** |
| | +LogosKG ($k = 1$) | 40.58↑ | 16.77 | 21.39↑ | 49.22 | 11.93 | 17.19 | 14.61 | 5.78 | 7.39 | 12.96 | 2.09 | 3.18 |
| | +LogosKG ($k = 2$) | 31.76↑ | 18.55 | 20.83↑ | 70.14 | 35.45 | 44.26 | 20.66 | 12.44 | 13.32 | 35.22 | 10.40 | 14.22 |
| | +LogosKG ($k = 3$) | 30.45↑ | 18.62 | 20.48↑ | 81.08↑ | 68.65 | 72.02 | 24.09↑ | 20.09 | 18.44 | 79.61 | 44.24 | 53.01 |
| | +LogosKG ($k = 4$) | 30.42↑ | 18.76 | 20.58↑ | 80.91↑ | 69.68 | 72.55 | 23.31 | 23.05 | 19.49 | 81.61 | 70.27 | 72.66 |
| | +LogosKG ($k = 5$) | 30.55↑ | 18.97 | **20.75**↑ | 80.90↑ | 69.84 | 72.62 | 24.04↑ | 24.28 | 20.40 | 81.99↑ | 79.19 | 78.31 |
| Llama-3.1-8B-Instruct | Baseline | 28.43 | 22.52 | 22.14 | 83.13 | 82.40 | **80.24** | 25.54 | 27.95 | **22.95** | 85.38 | 82.22 | **81.15** |
| | +LogosKG ($k = 1$) | 38.96↑ | 18.77 | 22.73↑ | 51.49 | 11.70 | 17.11 | 17.55 | 5.51 | 7.63 | 13.19 | 2.11 | 3.19 |
| | +LogosKG ($k = 2$) | 31.86↑ | 20.93 | 22.49↑ | 72.81 | 34.58 | 43.82 | 23.08 | 14.26 | 14.96 | 35.78 | 10.29 | 14.14 |
| | +LogosKG ($k = 3$) | 31.52↑ | 21.23 | 22.42↑ | 83.63↑ | 67.20 | 71.60 | 25.76↑ | 23.13 | 20.56 | 83.13 | 43.23 | 52.83 |
| | +LogosKG ($k = 4$) | 31.48↑ | 21.37 | 22.53↑ | 83.53↑ | 68.21 | 72.23 | 25.51 | 26.16 | 22.00 | 85.01 | 68.05 | 72.36 |
| | +LogosKG ($k = 5$) | 31.31↑ | 21.51 | **22.55**↑ | 83.51↑ | 68.37 | 72.32 | 26.13↑ | 27.25 | 22.72 | 85.39↑ | 76.81 | 78.13 |
| GPT-o3-mini | Baseline | 9.30 | 27.08 | 12.91 | 9.46 | 17.24 | 10.02 | 10.28 | 38.47 | 14.83 | 14.62 | 25.36 | **15.88** |
| | +LogosKG ($k = 1$) | 20.37↑ | 20.19 | **18.01**↑ | 8.83 | 4.23 | 3.99 | 14.38↑ | 7.83 | 8.77 | 5.97 | 1.87 | 1.95 |
| | +LogosKG ($k = 2$) | 12.98↑ | 23.71 | 15.46↑ | 10.27↑ | 10.00 | 8.17 | 12.68↑ | 17.41 | 12.48 | 6.93 | 4.51 | 3.72 |
| | +LogosKG ($k = 3$) | 12.38↑ | 24.87 | 15.27↑ | 11.02↑ | 15.75 | **10.63**↑ | 11.38↑ | 31.35 | 14.91↑ | 14.43 | 13.30 | 10.82 |
| | +LogosKG ($k = 4$) | 12.23↑ | 25.07 | 15.21↑ | 10.97↑ | 15.94 | **10.63**↑ | 10.76↑ | 35.15 | 15.01↑ | 14.66↑ | 20.45 | 14.40 |
| | +LogosKG ($k = 5$) | 11.93↑ | 25.18 | 14.97↑ | 10.92↑ | 15.94 | 10.61↑ | 10.95↑ | 37.14 | **15.40**↑ | 15.26↑ | 23.43 | **15.88** |

Table 4: LLM performance comparison of LogosKG with varying hop distances ($h = 1$ to 5) across UMLS and PrimeKG knowledge graphs on ProbSum and DDXPlus datasets. Bold F1 scores indicate the best performance across different hop settings and baselines. ↑ denotes improvement over baseline. Mistral-7B-Instruct-v0.3 and Gemma-7b-it show the same trend, shown in Table 7.

GraphBLAS) perform well at shallow hops but degrade rapidly with depth. Neo4j remains slow with consistently high TR. GPU-based methods (graph-tool, cuGraph, DGL, PyG) scale roughly linearly: graph-tool, cuGraph, and DGL show higher latency, while PyG starts lower. In contrast, the LogosKG family scales robustly: Numba and Torch-GPU remain under 200 ms with zero TR, and partitioned LogosKG-Large variants maintain moderate TR even at deeper hops. Torch-CPU performs weaker due to the lack of dedicated optimization. Overall, LogosKG achieves efficient and reliable multi-hop retrieval.

**Scalability** We evaluate LogosKG-Large on the PKG using the degree-aware partitioning strategy. Synthetic queries contain 1–20 randomly selected entities and are executed at depths of 1–5 hops. The cache size $N$ determines how many subgraphs remain in memory, while loads and evictions record swaps between disk and memory. We measure QT under varying hops, batch sizes, cache sizes, and backends. Cache activities are tracked through the number of loads and evictions.

As shown in Table 3, QT increases with hop count as deeper expansions involve more entities, while larger batches reduce latency due to batch optimization. Cache size has the largest impact: small caches trigger frequent swaps, increasing latency, whereas larger caches cut these operations and improve QT. Among backends, Numba, SciPy, and Torch-GPU are fastest, while Torch-CPU remains slower due to limited parallelization.

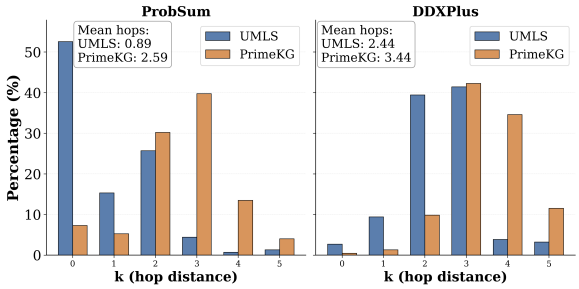


Figure 2: Hop-distance distribution of pair entities for the ProbSum and DDXPlus dataset on UMLS and PrimeKG.

**Overall performance** These experiments confirm that LogosKG delivers deterministic accuracy, strong efficiency across CPU and GPU settings, and scalable performance on billion-edge graphs, establishing it as a reliable and hardware-optimized solution for large-scale KG retrieval.

## 6 High-Hop KG with LLMs for Diagnosis

In this section we use LogosKG to assess the alignment between biomedical KGs and LLM diagnostic predictions, rather than to boost model accuracy. LLMs can produce unsupported conclusions, whereas KGs offer explicit multi-hop evidence. Recall that existing libraries struggle to perform high-hop retrieval efficiently (e.g., $k$=5). With LogosKG's scalable retrieval, we evaluate this alignment over entire graphs and quantify when KG structure supports or diverges from model reasoning.

We test LLMs on ProbSum and DDXPlus. All models are supervised fine-tuned (SFT) except *GPT-o3-mini*. Baseline scores in Table 4 correspond to SFT LLM outputs without LogosKG. Following prior work on KG+LLM for diagnostic reasoning (Gao et al., 2025), we report entity-level precision, recall, and F1 by mapping predictions and gold labels to the same biomedical entities via QuickUMLS (Soldaini and Goharian, 2016).

We use LogosKG to retrieve KG evidence and filter LLM-generated entities, retaining only those with KG support. The filtered set is then used to recompute precision, recall, and F1 (Section A.2). Beyond entity-level correctness, we apply PDSQI-9, a published, clinically validated nine-criterion rubric for LLM clinical evaluation (Croxford et al., 2025). Speficially, PDSQI-9 provides LLM-As-Judge that uses Azure GPT-o4-mini, with proven high correlation with physician judgments.

**Hop-distance characteristics** To better understand the structural properties of both KGs, we analyze the hop-distance distribution between entity pairs (from entities in input to the gold diagnosis entity) extracted from the ProbSum and DDXPlus datasets in Figure 2. The distributions reveal distinct topological patterns: on average, PrimeKG has longer paths (2.59–3.44 vs. 0.89–2.44 in UMLS), indicating richer multi-hop relations but higher reasoning complexity. These patterns suggest UMLS favors precise, localized retrieval, whereas PrimeKG supports broader, high-hop reasoning useful for LLM-based exploration.

**LLM performance with high-hop KG** Table 4 presents a sensitivity analysis of LLM performance with LogosKG under varying hop distances ($k$ = 1–5). Whether KG integration improves performance closely aligns with the connectivity patterns in Figure 2. On ProbSum, UMLS covers most gold concepts within shallow hops, leading to clear improvement for LLM+KG at $k = 1$ to 2. In contrast, DDXPlus contains many gold entities that located 2-3 hops away, reducing the benefit of early retrieval. For PrimeKG, where the majority of relevant concepts lie at longer distances (4–5 hops), performance gains only appear once LogosKG reaches higher hop depths. These results confirm that KG structure, particularly its hop-distance distribution, strongly governs the effectiveness of LLM–KG integration.

**Clinical rubric results** Figure 3 and Table 5 summarize evaluation using the clinically validated PDSQI-9 rubric with GPT-o4-mini as the LLM-as-Judge. Overall, incorporating LogosKG leads to moderate but consistent gains in dimensions related to comprehensibility, succinctness, and organization, suggesting that KG grounding improves coherence and focus in diagnostic reasoning. Improvements are most evident on DDXPlus with PrimeKG, where multi-hop retrieval provides richer contextual evidence, while shallow or already well-connected graphs such as UMLS show smaller changes. These findings indicate that LogosKG enhances reasoning clarity rather than raw factual accuracy, particularly when the underlying KG contributes deeper relational context.

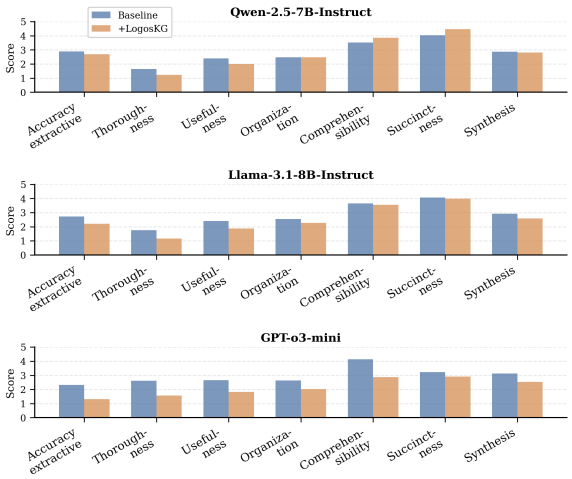


Figure 3: PDSQI-9 metric comparison for three models on ProbSum dataset with PrimeKG knowledge graph ($k$=5). Each subplot displays seven evaluation dimensions comparing with and without LogosKG.

| Dataset | KG | Qwen-2.5-7B | Llama-3.1-8B | GPT-o3-mini |
|---|---|---|---|---|
| **ProbSum** | UMLS | 2.85 / 2.70 | 2.91 / 2.46 | 2.96 / 1.80 |
| | PrimeKG | 2.84 / 2.77 | 2.85 / 2.48 | 2.89 / 2.07 |
| **DDXPlus** | UMLS | 2.67 / 2.64 | 2.74 / 2.67 | 2.32 / **2.91** ↑ |
| | PrimeKG | 2.64 / **2.83** ↑ | 2.73 / **2.79** ↑ | 2.34 / **2.73** ↑ |

Table 5: Average PDSQI-9 scores for baseline / +LogosKG ($k$=5) across three models. ↑ indicates improvement with LogosKG. Full per-metric results in Appendix Table 8.

## 7 Conclusion

We presented LogosKG, a hardware-aligned framework for scalable and interpretable multi-hop retrieval on large KGs. LogosKG achieves efficient and deterministic retrieval across billion-scale graphs. Beyond system performance, our case study showed that KG topology strongly influences how well structured knowledge supports LLM reasoning. These results position LogosKG as both a high-performance retrieval system and a foundation for future research on KG–LLM integration.

## Limitations

While LogosKG focuses on efficient multi-hop retrieval, the current implementation can be further developed to support other types of graph analysis based on similar sparse-matrix operations. Future extensions may explore broader algorithmic capabilities while maintaining efficiency and scalability.

In addition, retrieval from large KGs often produces a substantial number of candidate entities at higher hop distances. Further work is needed to refine these outputs, for example by integrating ranking or filtering strategies to highlight the most relevant results.

## Ethical Statement

This study does not involve any human subjects. All datasets and models used, including UMLS, PKG, PrimeKG, ProbSum, and DDXPlus, are publicly available and used strictly for research purposes under their respective data use agreements. All experiments were conducted on HIPAA-compliant servers, adhering to institutional and data governance policies.

## References


Olivier Bodenreider. 2004. The unified medical language system (umls): integrating biomedical terminology. *Nucleic acids research*, 32(suppl_1):D267–D270.

Xingjuan Cai, Wanwan Guo, Mengkai Zhao, Zhihua Cui, and Jinjun Chen. 2023. A knowledge graph-based many-objective model for explainable social recommendation. *IEEE Transactions on Computational Social Systems*, 10(6):3021–3030.

Payal Chandak, Kexin Huang, and Marinka Zitnik. 2023. Building a knowledge graph to enable precision medicine. *Nature Scientific Data*.

David Chang, Ivana Balažević, Carl Allen, Daniel Chawla, Cynthia Brandt, and Richard Andrew Taylor. 2020. Benchmark and best practices for biomedical knowledge graph embeddings. In *Proceedings of the conference. Association for Computational Linguistics. Meeting*, volume 2020, page 167.

Yingjian Chen, Haoran Liu, Yinhong Liu, Jinxiang Xie, Rui Yang, Han Yuan, Yanran Fu, Peng Yuan Zhou, Qingyu Chen, James Caverlee, and 1 others. 2025. Graphcheck: Breaking long-term text barriers with extracted knowledge graph-powered fact-checking. *arXiv preprint arXiv:2502.16514*.

William W. Cohen, Haitian Sun, R. Alex Hofer, and Matthew Siegler. 2020. Scalable neural methods for reasoning with a symbolic knowledge base. In *International Conference on Learning Representations*.

Emma Croxford, Yanjun Gao, Nicholas Pellegrino, Karen Wong, Graham Wills, Elliot First, Miranda Schnier, Kyle Burton, Cris Ebby, Jillian Gorski, and 1 others. 2025. Development and validation of the provider documentation summarization quality instrument for large language models. *Journal of the American Medical Informatics Association*, 32(6):1050–1060.

Gabor Csardi and Tamas Nepusz. 2006. The igraph software. *Complex syst*, 1695:1–9.

Preetam Prabhu Srikar Dammu, Himanshu Naidu, Mouly Dewan, YoungMin Kim, Tanya Roosta, Aman Chadha, and Chirag Shah. 2024. Claimver: Explainable claim-level verification and evidence attribution of text through knowledge graphs. *arXiv preprint arXiv:2403.09724*.

Alin Deutsch, Yu Xu, Mingxi Wu, and Victor Lee. 2019. Tigergraph: A native mpp graph database. *arXiv preprint arXiv:1901.08248*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407.

Arsene Fansi Tchango, Rishab Goel, Zhi Wen, Julien Martel, and Joumana Ghosn. 2022. Ddxplus: A new dataset for automatic medical diagnosis. *Advances in neural information processing systems*, 35:31306–31318.

Matthias Fey and Jan Eric Lenssen. 2019. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*.

Yanjun Gao, Dmitriy Dligach, Timothy Miller, and Majid Afshar. 2023. Overview of the problem list summarization (ProbSum) 2023 shared task on summarizing patients' active diagnoses and problems from electronic health record progress notes. In *Proceedings of the 22nd Workshop on Biomedical Natural Language Processing and BioNLP Shared Tasks*, pages 461–467, Toronto, Canada. Association for Computational Linguistics.

Yanjun Gao, Ruizhe Li, Emma Croxford, John Caskey, Brian W Patterson, Matthew Churpek, Timothy Miller, Dmitriy Dligach, and Majid Afshar. 2025. Leveraging medical knowledge graphs into large language models for diagnosis prediction: Design and application study. *Jmir Ai*, 4:e58670.

Aric Hagberg, Pieter J Swart, and Daniel A Schult. 2008. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Laboratory (LANL), Los Alamos, NM (United States).

Yani Huang, Richong Zhang, Zhijie Nie, Junfan Chen, and Xuefeng Zhang. 2025. A graph-based verification framework for fact-checking. *arXiv preprint arXiv:2503.07282*.

Mingyi Jia, Junwen Duan, Yan Song, and Jianxin Wang. 2024. medikal: Integrating knowledge graphs as assistants of llms for enhanced clinical diagnosis on emrs. *arXiv preprint arXiv:2406.14326*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

Alistair EW Johnson, Tom J Pollard, Lu Shen, Li-wei H Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. 2016. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3(1):1–9.

Jeremy Kepner, Petri Aaltonen, David A. Bader, Aydın Buluç, Franz Franchetti, John R. Gilbert, Dylan Hutchison, Manoj Kumar, Andrew Lumsdaine, Henning Meyerhenke, Scott McMillan, José E. Moreira, John D. Owens, Carl Yang, Marcin Zalewski, and Timothy G. Mattson. 2016. Mathematical foundations of the graphblas. In *IEEE High Performance Extreme Computing Conference (HPEC)*.

Jiho Kim, Sungjin Park, Yeonsu Kwon, Yohan Jo, James Thorne, and Edward Choi. 2023. Factkg: Fact verification via reasoning on knowledge graphs. *arXiv preprint arXiv:2305.06590*.

Jure Leskovec and Rok Sosič. 2016. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1):1–20.

Hao Liu, Zhengren Wang, Xi Chen, Zhiyu Li, Feiyu Xiong, Qinhan Yu, and Wentao Zhang. 2025. HopRAG: Multi-hop reasoning for logic-aware retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 1897–1913, Vienna, Austria. Association for Computational Linguistics.

Yuxing Lu, Sin Yee Goi, Xukai Zhao, and Jinzhuo Wang. 2025. Biomedical knowledge graph: A survey of domains, tasks, and real-world applications. *arXiv preprint arXiv:2501.11632*.

Neo4j, Inc. 2025. Neo4j Graph Data Science. https://neo4j.com/product/graph-data-science/. Accessed: 2025-10-06.

OpenAI. 2025. Openai o3-mini system card. Technical Report, OpenAI.

Tiago P. Peixoto. 2014. The graph-tool python library. *figshare*.

Hoang Pham, Thanh-Do Nguyen, and Khac-Hoai Nam Bui. 2025a. Verify-in-the-graph: Entity disambiguation enhancement for complex claim verification with interactive graph representation. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 5181–5197, Albuquerque, New Mexico. Association for Computational Linguistics.

Hoang Pham, Thanh-Do Nguyen, and Khac-Hoai Nam Bui. 2025b. Verify-in-the-graph: Entity disambiguation enhancement for complex claim verification with interactive graph representation. *arXiv preprint arXiv:2505.22993*.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

RAPIDS AI. 2025. cuGraph: RAPIDS GPU Graph Analytics Library. https://github.com/rapidsai/cugraph. Accessed: 2025-10-06.

Mohammad Reza Rezaei, Reza Saadati Fard, Jayson Parker, Rahul G Krishnan, and Milad Lankarany. 2025. Adaptive knowledge graphs enhance medical question answering: Bridging the gap between llms and evolving medical knowledge. *arXiv e-prints*, pages arXiv–2502.

Farzad Shami, Stefano Marchesin, and Gianmaria Silvello. 2025. Fact verification in knowledge graphs using llms. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 3985–3989.

Kartik Sharma, Peeyush Kumar, and Yunqing Li. 2024. Og-rag: Ontology-grounded retrieval-augmented generation for large language models. *arXiv preprint arXiv:2412.15235*.

Luca Soldaini and Nazli Goharian. 2016. Quickumls: a fast, unsupervised approach for medical concept extraction. In *MedIR workshop, sigir*, pages 1–4.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, and 1 others. 2024. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*.

Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, and 1 others. 2019. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*.

Shijie Wang, Wenqi Fan, Yue Feng, Lin Shanru, Xinyu Ma, Shuaiqiang Wang, and Dawei Yin. 2025a. Knowledge graph retrieval-augmented generation for

LLM-based recommendation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 27152–27168, Vienna, Austria. Association for Computational Linguistics.

Song Wang, Junhong Lin, Xiaojie Guo, Julian Shun, Jundong Li, and Yada Zhu. 2025b. Reasoning of large language models over knowledge graphs with super-relations. In *The Thirteenth International Conference on Learning Representations*.

Rong Wu, Pinlong Cai, Jianbiao Mei, Licheng Wen, Tao Hu, Xuemeng Yang, Daocheng Fu, and Botian Shi. 2025. Kg-traces: Enhancing large language models with knowledge graph-constrained trajectory reasoning and attribution supervision. *arXiv preprint arXiv:2506.00783*.

Jian Xu, Sunkyu Kim, Min Song, Minbyul Jeong, Donghyeon Kim, Jaewoo Kang, Justin F Rousseau, Xin Li, Weijia Xu, Vetle I Torvik, and 1 others. 2020. Building a pubmed knowledge graph. *Scientific data*, 7(1):205.

Jian Xu, Chao Yu, Jiawei Xu, Vetle I Torvik, Jaewoo Kang, Mujeen Sung, Min Song, Yi Bu, and Ying Ding. 2025. Pubmed knowledge graph 2.0: Connecting papers, patents, and clinical trials in biomedical science. *Scientific Data*, 12(1):1018.

Kaiwen Zuo, Yirui Jiang, Fan Mo, and Pietro Lio. 2025. Kg4diagnosis: A hierarchical multi-agent llm framework with knowledge graph enhancement for medical diagnosis. In *AAAI Bridge Program on AI for Medicine and Healthcare*, pages 195–204. PMLR.

# A Appendix

## A.1 Jaccard similarity for LogosKG retrieval fidelity

To evaluate retrieval accuracy, we use the Jaccard similarity:

$$\text{Jaccard}(A, B) = \frac{|R_A \cap R_B|}{|R_A \cup R_B|},$$

where $R_A$ and $R_B$ are the retrieved entity sets from the method $A$ and method $B$ on the same query. Higher values indicate stronger agreement (1.00 = identical results).

Table 6 shows that all retrieved results are perfectly overlap, proving the fidelity of LOGOSKG.

## A.2 Entity-level Evaluation Metric

We compute precision and recall based on the overlap between predicted and gold-standard entities, as defined below:

$$\begin{aligned} \text{Precision} &= \frac{|\,pred \cap gold\,|}{|\,pred\,|}, \\ \text{Recall} &= \frac{|\,pred \cap gold\,|}{|\,gold\,|}. \end{aligned} \tag{12}$$

| LogosKG family | Hop 1 | Hop 2 | Hop 3 | Hop 4 | Hop 5 |
|---|---|---|---|---|---|
| Neo4j | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| GraphBLAS | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| igraph | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| NetworkX | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| graph-tool | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| SNAP | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| cuGraph | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| DGL | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| PyG | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |

Table 6: Jaccard similarity of the **LogosKG family** vs. CPU and GPU baselines across hops 1–5.

We further compute *filtered precision* and *filtered recall* to assess the effect of KG verification. Let $pred_f$ denote the subset of predictions supported by LogosKG within $k$ hops:

$$\begin{aligned} \text{Filtered Precision} &= \frac{|\,pred_f \cap gold\,|}{|\,pred_f\,|}, \\ \text{Filtered Recall} &= \frac{|\,pred_f \cap gold\,|}{|\,gold\,|}. \end{aligned} \tag{13}$$

## A.3 Supplementary Sensitivity Analysis

| Model | Method | ProbSum | | | DDXPlus | | |
|---|---|---|---|---|---|---|---|
| | | P | R | F1 | P | R | F1 |
| Mistral-7B-Instruct-v0.3 | Baseline | 20.55 | 19.30 | 18.02 | 81.44 | 82.31 | **79.25** |
| | +LogosKG ($k = 1$) | 32.44↑ | 15.55 | 18.92↑ | 49.90 | 11.57 | 16.84 |
| | +LogosKG ($k = 2$) | 24.01↑ | 17.55 | 18.32↑ | 72.15 | 34.40 | 43.50 |
| | +LogosKG ($k = 3$) | 23.21↑ | 17.99 | 18.27↑ | 82.17↑ | 66.94 | 70.78 |
| | +LogosKG ($k = 4$) | 23.15↑ | 18.06 | 18.31↑ | 81.96↑ | 67.97 | 71.38 |
| | +LogosKG ($k = 5$) | 22.92↑ | 18.28 | **18.39**↑ | 81.85↑ | 68.11 | 71.43 |
| Gemma-7b-it | Baseline | 10.23 | 14.77 | 9.01 | 59.33 | 67.20 | **54.36** |
| | +LogosKG ($k = 1$) | 20.24↑ | 9.23 | **10.46**↑ | 38.74 | 9.88 | 13.51 |
| | +LogosKG ($k = 2$) | 13.56↑ | 11.46 | 9.71↑ | 55.85 | 28.30 | 32.84 |
| | +LogosKG ($k = 3$) | 12.65↑ | 11.50 | 9.23↑ | 66.27↑ | 53.53 | 52.34 |
| | +LogosKG ($k = 4$) | 12.07↑ | 11.62 | 9.13↑ | 65.80↑ | 54.44 | 52.49 |
| | +LogosKG ($k = 5$) | 11.99↑ | 11.88 | 9.15↑ | 65.33↑ | 54.55 | 52.25 |

(a) UMLS KG

| Model | Method | ProbSum | | | DDXPlus | | |
|---|---|---|---|---|---|---|---|
| | | P | R | F1 | P | R | F1 |
| Mistral-7B | Baseline | 19.28 | 22.90 | **17.83** | 83.68 | 82.63 | **80.43** |
| | +LogosKG ($k = 1$) | 11.78 | 3.91 | 5.04 | 13.50 | 2.12 | 3.23 |
| | +LogosKG ($k = 2$) | 19.05 | 12.11 | 12.48 | 35.75 | 10.11 | 13.83 |
| | +LogosKG ($k = 3$) | 20.74↑ | 20.26 | 17.14 | 82.53 | 43.13 | 52.72 |
| | +LogosKG ($k = 4$) | 19.81↑ | 21.85 | 17.63 | 83.65 | 68.37 | 71.99 |
| | +LogosKG ($k = 5$) | 19.58↑ | 22.37 | 17.82 | 84.01↑ | 77.02 | 77.57 |
| Gemma-7B | Baseline | 10.10 | 15.02 | 10.16 | 69.87 | 67.54 | **63.56** |
| | +LogosKG ($k = 1$) | 9.46 | 3.09 | 4.19 | 12.56 | 2.14 | 3.17 |
| | +LogosKG ($k = 2$) | 11.69↑ | 8.03 | 7.25 | 29.70 | 8.85 | 11.73 |
| | +LogosKG ($k = 3$) | 12.19↑ | 12.81 | 9.80 | 69.91↑ | 35.30 | 42.13 |
| | +LogosKG ($k = 4$) | 11.12↑ | 14.25 | 10.18↑ | 70.75↑ | 55.32 | 57.24 |
| | +LogosKG ($k = 5$) | 11.20↑ | 14.48 | **10.28**↑ | 71.16↑ | 62.64 | 62.18 |

(b) PrimeKG KG

Table 7: Performance comparison of LogosKG with varying hop distances ($h = 1$ to 5) across UMLS and PrimeKG knowledge graphs on ProbSum and DDXPlus datasets. Bold F1 scores indicate the best performance across different hop settings and baselines. ↑ denotes improvement over baseline.

We further analyze how LLM performance changes with different retrieval depths ($k = 1$–5) using LOGOSKG. The analysis includes two models, *Mistral-7B-Instruct-v0.3* and *Gemma-7b-it*, evaluated on UMLS and PrimeKG. As shown in Table 7, we can observe similar results as in Table 4. On ProbSum, most gold entities in UMLS

| Dataset | KG | Metric | Qwen-2.5-7B-Instruct | | Llama-3.1-8B-Instruct | | Mistral-7B-Instruct-v0.3 | | Gemma-7b-it | | GPT-o3-mini | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Baseline | +LogosKG | Baseline | +LogosKG | Baseline | +LogosKG | Baseline | +LogosKG | Baseline | +LogosKG |
| ProbSum | UMLS | Accuracy extractive | 2.85 | 2.43 | 2.76 | 2.03 | 2.25 | 1.89 | 1.29 | 1.52 ↑ | 2.40 | 1.26 |
| | | Thoroughness | 1.60 | 1.29 | 1.81 | 1.33 | 1.67 | 1.23 | 1.34 | 1.27 | 2.59 | 1.45 |
| | | Usefulness | 2.36 | 2.13 | 2.50 | 1.97 | 2.28 | 1.88 | 1.29 | 1.60 ↑ | 2.81 | 1.65 |
| | | Organization | 2.52 | 2.48 | 2.61 | 2.24 | 2.40 | 2.14 | 1.14 | 1.87 ↑ | 2.82 | 1.69 |
| | | Comprehensibility | 3.60 | 3.75 ↑ | 3.69 | 3.42 | 3.35 | 3.34 | 1.19 | 2.96 ↑ | 4.25 | 2.45 |
| | | Succinctness | 4.05 | 4.31 ↑ | 4.14 | 3.84 | 3.97 | 3.92 | 1.91 | 3.07 ↑ | 3.27 | 2.28 |
| | | Synthesis | 2.94 | 2.71 | 2.98 | 2.68 | 2.96 | 2.71 | 1.52 | 2.36 ↑ | 3.05 | 2.31 |
| | | **Average** | 2.85 | 2.70 | 2.91 | 2.46 | 2.66 | 2.38 | 1.38 | 2.04 ↑ | 2.96 | 1.80 |
| | PrimeKG | Accuracy extractive | 2.88 | 2.69 | 2.72 | 2.20 | 2.18 | 2.00 | 1.31 | 1.66 ↑ | 2.31 | 1.32 |
| | | Thoroughness | 1.65 | 1.23 | 1.76 | 1.16 | 1.67 | 1.14 | 1.29 | 1.17 | 2.62 | 1.56 |
| | | Usefulness | 2.40 | 2.01 | 2.40 | 1.88 | 2.30 | 1.85 | 1.34 | 1.64 ↑ | 2.66 | 1.83 |
| | | Organization | 2.47 | 2.48 ↑ | 2.55 | 2.27 | 2.39 | 2.19 | 1.15 | 2.07 ↑ | 2.64 | 2.03 |
| | | Comprehensibility | 3.53 | 3.85 ↑ | 3.65 | 3.55 | 3.31 | 3.64 ↑ | 1.19 | 3.50 ↑ | 4.13 | 2.88 |
| | | Succinctness | 4.03 | 4.46 ↑ | 4.06 | 3.99 | 3.96 | 4.25 ↑ | 1.76 | 3.93 ↑ | 3.23 | 2.92 |
| | | Synthesis | 2.87 | 2.82 | 2.93 | 2.58 | 2.97 | 2.76 | 1.58 | 2.78 ↑ | 3.12 | 2.53 |
| | | **Average** | 2.84 | 2.77 | 2.85 | 2.48 | 2.64 | 2.48 | 1.37 | 2.32 ↑ | 2.89 | 2.07 |
| DDXPlus | UMLS | Accuracy extractive | 1.62 | 1.69 ↑ | 1.58 | 1.58 | 1.55 | 1.54 | 1.55 | 2.30 ↑ | 1.35 | 2.95 ↑ |
| | | Thoroughness | 3.36 | 1.99 | 3.51 | 2.15 | 3.33 | 1.94 | 2.82 | 1.82 | 2.33 | 1.37 |
| | | Usefulness | 2.20 | 2.12 | 2.18 | 2.04 | 2.25 | 2.03 | 1.84 | 2.38 ↑ | 1.97 | 2.26 ↑ |
| | | Organization | 2.44 | 2.25 | 2.53 | 2.45 | 2.38 | 2.26 | 1.63 | 2.65 ↑ | 2.12 | 2.88 ↑ |
| | | Comprehensibility | 3.68 | 4.18 ↑ | 3.75 | 4.19 ↑ | 3.61 | 4.26 ↑ | 1.82 | 4.24 ↑ | 4.09 | 4.19 ↑ |
| | | Succinctness | 3.60 | 4.47 ↑ | 3.86 | 4.40 ↑ | 3.97 | 4.56 ↑ | 2.45 | 4.58 ↑ | 2.15 | 4.64 ↑ |
| | | Synthesis | 2.86 | 2.68 | 3.00 | 3.00 | 3.06 | 3.06 | 2.24 | 3.00 ↑ | 3.01 | 2.38 |
| | | **Average** | 2.67 | 2.64 | 2.74 | 2.67 | 2.70 | 2.63 | 1.98 | 2.88 ↑ | 2.32 | 2.91 ↑ |
| | PrimeKG | Accuracy extractive | 1.60 | 2.09 ↑ | 1.61 | 2.11 ↑ | 1.51 | 2.02 ↑ | 1.48 | 2.22 ↑ | 1.36 | 2.46 ↑ |
| | | Thoroughness | 3.43 | 1.85 | 3.55 | 1.79 | 3.48 | 1.80 | 2.83 | 1.65 | 2.29 | 1.51 |
| | | Usefulness | 2.20 | 2.36 ↑ | 2.19 | 2.32 ↑ | 2.12 | 2.13 ↑ | 1.91 | 2.19 ↑ | 1.97 | 2.16 ↑ |
| | | Organization | 2.30 | 2.57 ↑ | 2.41 | 2.51 ↑ | 2.34 | 2.38 ↑ | 1.67 | 2.64 ↑ | 2.13 | 2.54 ↑ |
| | | Comprehensibility | 3.63 | 4.28 ↑ | 3.74 | 4.29 ↑ | 3.60 | 4.26 ↑ | 1.87 | 4.21 ↑ | 4.08 | 3.95 |
| | | Succinctness | 3.44 | 4.58 ↑ | 3.73 | 4.57 ↑ | 3.82 | 4.65 ↑ | 2.55 | 4.59 ↑ | 2.27 | 4.47 ↑ |
| | | Synthesis | 2.97 | 3.00 ↑ | 3.00 | 2.87 | 3.00 | 3.00 | 2.04 | 2.92 ↑ | 3.00 | 2.47 |
| | | **Average** | 2.64 | 2.83 ↑ | 2.73 | 2.79 ↑ | 2.66 | 2.74 ↑ | 1.98 | 2.82 ↑ | 2.34 | 2.73 ↑ |

Table 8: Comparison of diagnosis quality with and without LogosKG ($k$=5) using PDSQI-9 metrics with GPT-o4-mini as the judge. Evaluated across five models on ProbSum and DDXPlus datasets with UMLS and PrimeKG knowledge graphs. ↑ indicates improvement with LogosKG compared to baselines.

are reachable within one or two hops, leading to strong performance gains at smaller $k$. In DDXPlus, many gold entities are located two to three hops away from the start entities, so the benefit is smaller for low hops. For PrimeKG, where related concepts tend to be farther apart, improvements appear only at higher hop depths. Overall, these results show that the structure of the KG plays a major role in how much LLMs can benefit from KG integration.

### A.4 Diagnosis Quality Evaluation using PDSQI-9 Criteria

Table 8 shows that LOGOSKG generally improves diagnosis quality across most models and datasets. The gains are apparent for Comprehensibility, Succinctness, and Organization, suggesting that high-hop retrieval helps models produce clearer and more concise outputs. Improvements are modest on ProbSum, where most concepts are already covered by the input notes (as shown in Figure 2), but much stronger on DDXPlus because it requires evidence at high hops. In particular, models such as Gemma-7b-it and GPT-o3-mini benefit the most, with improvements across more metrics, showing that LOGOSKG effectively filters out unsupported or redundant diagnoses and thus improves overall diagnosis quality.