Scaling Biomedical Knowledge Graph Retrieval for Interpretable Reasoning: Applications to Clinical Diagnosis Prediction

He Cheng[1], Yifu Wu[1], Saksham Khatwani[1,2], Maya Kruse[1], Dmitriy Dligach[3], Timothy A. Miller[4,5], Majid Afshar[6], Yanjun Gao[1]*

[1]University of Colorado Anschutz Medical Campus, Aurora, CO, USA; [2]University of Colorado Boulder, Boulder, CO, USA; [3]Loyola University Chicago, Chicago, IL, USA; [4]Harvard Medical School, Boston, MA, USA; [5]Boston Children's Hospital, Boston, MA, USA; [6]University of Wisconsin-Madison, Madison, WI, USA

Abstract

Biomedical knowledge graphs (KGs) organize molecular mechanisms, biological pathways, and clinical concepts into structured representations that support diagnostic reasoning. As these graphs grow in scale and connectivity, scalable and interpretable multi-hop retrieval over deep biomedical graph structures has become a major computational bottleneck.

We present LogosKG, a hardware-optimized retrieval system that enables efficient k-hop traversal over very large biomedical KGs using symbolic graph formulations and hardware-efficient execution. By integrating degree-aware partitioning, cross-partition routing, and on-demand caching, LogosKG scales to billion-edge graphs while preserving retrieval fidelity.

Experiments demonstrate substantial efficiency improvements over CPU- and GPU-based baselines. Using diagnosis-oriented retrieval workloads as a downstream case study, we show that scalable access to deep, high-hop biomedical graph structures enables interpretable diagnostic evidence propagation. A clinician-aligned LLM-as-judge evaluation further indicates that high-hop KG retrieval improves reasoning quality in terms of accuracy, comprehensibility, and succinctness, underscoring the value of deep graph retrieval for diagnostic reasoning.

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), 407K nodes, 3.4M edges; and the $100\times$ larger **PubMedKG** (Xu et al., 2020, 2025), 54.4M nodes, 86.5M edges) 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, con-

* Corresponding author

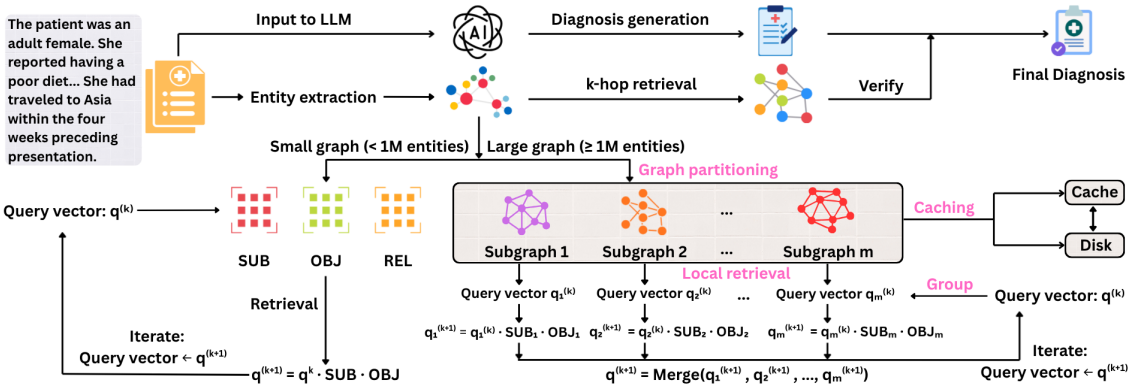

Figure 1: Overview of the LogosKG retrieval framework. LogosKG adopts a linear algebra–based retrieval method that supports both small and large KGs. For large graphs, LogosKG incorporates graph partitioning, cross-graph routing, and on-demand caching to enable efficient multi-hop retrieval across distributed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

suming 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).

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). While symbolic graph reasoning has been explored in prior work (Cohen et al., 2020), LogosKG extends this methodology into an unified, end-to-end system that makes high-hop traversal practical on large, densely connected KGs using single-device hardware, 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 (where $\mathcal{E}$ is the set of entities and $\mathcal{T}$ is the set of triples). 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.

LogosKG removes a fundamental system bottleneck in scalable high-hop retrieval, thereby enabling the study LLM reasoning over high-hop KGs, independent of whether downstream selection or reasoning components are learned or non-learned. We instantiate this capacity through a systematic study of KG-LLM setup for clinical diagnosis prediction, using a two-round interaction setup to examine how LLM responds to KG structure under increasing hop depths. We summarize the main contributions of this paper as follows:

- **Scalable system capability for large KG traversal**: We present LogosKG, a hardware-aligned framework that enables deterministic, interpretable high-hop retrieval at scale on single-device hardware (§3).
- **Comprehensive system evaluation:** We benchmark retrieval fidelity, efficiency, and scalability across CPU and GPU baselines (§5).

- **Analysis of KG–LLM interaction under high-hop regimes:** Leveraging LOGOSKG, we perform a systematic study of how LLM predictions interact with deep KG structures through a two-round interaction investigation (§6).

LOGOSKG serves as a general systems backbone for high-hop KG retrieval and can support both learned and non-learned refinement and reasoning strategies at scale. While KG-LLM interaction is a complex research question in its own right, we include analyses of both non-learned and learned high-hop KG–LLM designs in the Appendix (§A.6–A.8), illustrating that LOGOSKG makes such studies feasible.

2 Related Work

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 but require significant infrastructure for sharding and incur runtime overhead from query parsing pipelines and transaction logs. **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 due to the loss of edge provenance during aggregation. **Graph analysis tools**, such as igraph (Csardi and Nepusz, 2006), NetworkX (Hagberg et al., 2008), and SNAP (Leskovec and Sosič, 2016), offer flexible APIs but rely on memory-bound pointer-chasing algorithms that do not scale effectively. Finally, **GPU-based frameworks** such as cuGraph, DGL (Wang et al., 2019), and PyG (Fey and Lenssen, 2019) enable fast single-machine computation but prioritize dense tensor-based training over retrieval; their reliance on classical search backends limits efficiency for multi-hop reasoning.

2.1 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:

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

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:

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

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:

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

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,

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

Repeating this operation k times yields k**-hop retrieval**, denoted as: $\mathbf{q}^{(k)} = \mathbf{q}^{(0)} \cdot (\mathbf{SUB} \cdot \mathbf{OBJ})^k$, 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 h, we retrieve the activated triplets by

$$\mathbf{t}^{(h)} = \mathbf{q}^{(h-1)} \cdot \mathbf{SUB}. \tag{5}$$

The set of nonzero indices is

$$T_h = \{\, t \mid \mathbf{t}^{(h)}[t] = 1 \,\}. \tag{6}$$

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

$$\begin{aligned} s_h &= \{\, s \mid [\mathbf{SUB}]_{i,t} = 1 \,\}, \\ r_h &= \{\, r \mid [\mathbf{REL}]_{t,r} = 1 \,\}, \\ e_h &= \{\, o \mid [\mathbf{OBJ}]_{t,j} = 1 \,\}. \end{aligned} \tag{7}$$

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.

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)
```

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.

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.

Our framework is not limited to this partitioning algorithm. In fact, LogosKG can support any partitioning strategy and integrate them with the following cross-graph routing and on-demand caching mechanisms, although retrieval latency may vary.

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:

$$\mathbf{t}_i^{(k)} = \mathbf{q}_i^{(k)} \cdot \mathbf{SUB}_i, \tag{8}$$

$$\mathbf{q}_i^{(k+1)} = (\mathbf{q}_i^{(k)} \cdot \mathbf{SUB}_i) \cdot \mathbf{OBJ}_i. \tag{9}$$

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, τ_{mm} the time for in-memory sparse matrix multiplication, and τ_{io} the average disk load time. The expected retrieval cost per subgraph is

$$\mathbb{E}[\tau_{\text{retrieval}}] = h \cdot \tau_{\text{mm}} + (1-h) \cdot (\tau_{\text{mm}} + \tau_{\text{io}}). \tag{10}$$

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.

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 HIPAA-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).

4 Dataset and Setup

Our results are organized into two parts: (1) LogosKG system evaluation, focusing on retrieval accuracy, efficiency, and scalability; and (2) an analysis of KG-LLM interaction in high-hop KG regimes, using clinical diagnosis prediction as a representative problem domain.

We examine three biomedical KGs in our experiments: **UMLS** (Bodenreider, 2004), a large-scale biomedical ontology comprising 407K nodes and 3.4M edges across 133 semantic types; the **PubMed Knowledge Graph (PKG)** (Xu et al., 2020, 2025), a massive citation network connecting 54.4M nodes (including authors, publications, and institutions) via 86.5M edges; and **PrimeKG** (Chandak et al., 2023), which 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. DDX-Plus (Fansi Tchango et al., 2022) offers large-scale symptom–diagnosis pairs and is used in the KG-LLM 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

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.

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 4.1 (OpenAI, 2025b), and GPT-5-mini (OpenAI, 2025a). We employ GPT-5-mini as the LLM-as-a-Judge and for the supplementary experiments detailed in Appendix A.7. All open-sourced LLMs are running on the GPU server. We use GPT-4.1 and GPT-5-mini via HIPAA-compliant Microsoft Azure OpenAI endpoint, thereby complying with the ProbSum Data Use Agreement for MIMIC-III derived content (Johnson et al., 2016).

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 KG 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 re-

trieval is deterministic given the KG structure. We neglect the detailed results here and present them in Table 4. 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, 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.

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.

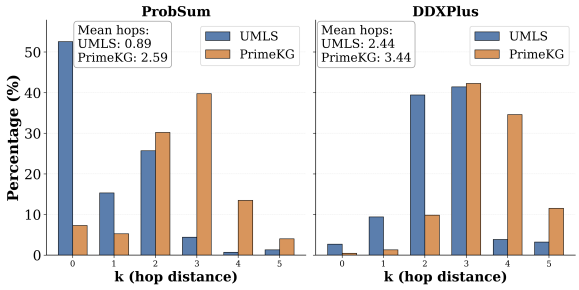

Figure 2: Hop-distance distribution of pair entities for the ProbSum and DDXPlus dataset on UMLS and PrimeKG.

6 Interaction Regimes between High-Hop KGs and LLM for Diagnosis

Existing approaches that integrate KGs with LLMs often restrict reasoning to shallow neighborhoods due to the exponential growth of the search space. However, many real-world reasoning tasks require traversing distal, multi-hop associations that lie beyond these local regions. In the biomedical KG–LLM setting, for instance, prior work often stops at 1-2 hops (and at most 3 hop) as the higher-hop expansion quickly becomes computationally infeasible (Gao et al., 2025; Zuo et al., 2025; Jiang et al., 2024). Our work deviates from the "learning-to-rank" paradigm to explore a structural regime of scalable graph traversal, enabled by LOGOSKG, which makes high-hop reasoning systematically accessible to LLMs. Following prior work (Gao et al., 2025; Zuo et al., 2025), we use clinical diagnosis as a representative setting to characterize how knowledge graph topology induces distinct interaction regimes between KGs and LLM reasoning.

Topological landscape. Figure 2 presents hop distribution across the two KGs on the two datasets we are investigating. On UMLS, gold entities can mostly be found within 1-2 hops, while PrimeKG shows a more long-tail distribution up to 4-5 hops. These structural differences directly govern how KG ontologies interact with LLM predictions.

The precision-recall tradeoffs. We adopt a two-round interaction paradigm to understand the interaction between KG topology and systematic precision-recall behaviors of LLM. In Round 1, the KG acts as a structural constraint that eliminates LLM hallucinations. Because LogosKG can retrieve these neighbors with 100% deterministic fidelity ($Jaccard = 1.0$), the LLM is forced to align its prediction with the medical ontology. In Round 2, LOGOSKG provides the LLM with a long-tail candidate space that was previously invisible, making the optimal "depth" not a fixed parameter.

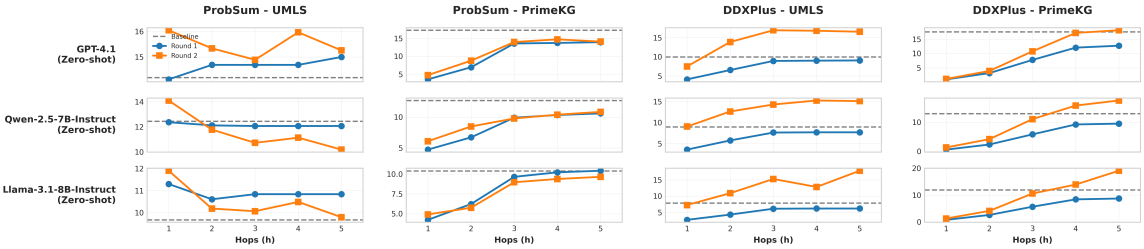

Figure 3: Performance comparison of KG filtering (Round 1) and enhancement (Round 2) across varying hop distances ($k = 1$ to $k = 5$) on ProbSum and DDXPlus datasets using UMLS and PrimeKG. F1-score metrics are shown for **GPT-4.1, Qwen-2.5-7B-Instruct, and Llama-3.1-8B-Instruct** in the zero-shot setting, with baseline performance included for reference.

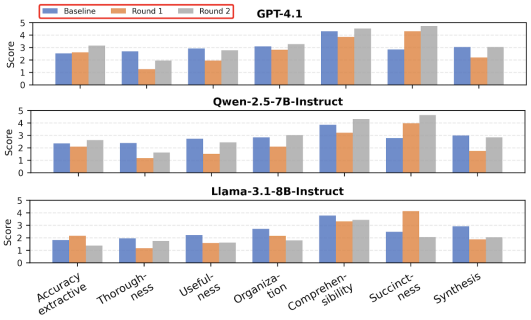

Figure 4: PDSQI-9 comparison for three models on DDXPlus dataset with UMLS (k=5). Each subplot displays seven evaluation dimensions comparing Baseline, Round 1 (KG-filtered), and Round 2 (KG-enhanced) approaches.

As shown in Figure 3, for UMLS, performance gains are concentrated in early hops, and Round 1 captures most of the benefit. In contrast, PrimeKG exhibits delayed gains that emerge only at higher hop distances, where Round 2 expansion consistently provides additional improvements. Importantly, this pattern holds across all three LLMs despite differences in absolute performance, indicating that the observed regimes are induced by KG structure rather than model-specific artifacts. These regimes are not tied to a particular retrieval implementation; rather, they were previously obscured because learning-based models and classical graph libraries encounter a computational barrier at high hop distances. By enabling scalable and deterministic high-hop traversal, LOGOSKG serves as a *structural backbone* that exposes a broader capability space for KG-LLM interaction.

Clinical reasoning quality. We further compare the diagnoses with and without LogosKG (k=5) using the using the clinically validated PDSQI-9 rubric. PDSQI-9 is a nine-criterion framework that evaluates diagnostic documentation along dimensions such as accuracy, comprehensibility, organization, and succinctness, with a HIPAA-compliant Azure GPT-5-mini model serving as the LLM-as-a-Judge (Croxford et al., 2025b,a). As shown in Figure 4, incorporating LOGOSKG leads to consistent improvements across several dimensions, particularly extractive accuracy, comprehensibility, and succinctness. These gains primarily reflect changes in the structure and clarity of diagnostic reasoning rather than raw correctness alone. By constraining predictions to KG-supported evidence in Round 1, LOGOSKG suppresses unsupported or redundant diagnoses, which directly benefits succinctness and organization. Round 2 further improves comprehensibility by reintroducing clinically relevant conditions that may be omitted by the LLM under unconstrained generation. Improvements are observed consistently for GPT-4.1 and Qwen-2.5-7B-Instruct, and are especially pronounced on the DDXPlus dataset when using PrimeKG, where deeper relational context is required.

Additional analysis. We provide qualitative examples of Round 1 filtering and Round 2 enhancement, along with results from fine-tuned LLMs and similarity-based refinement built on LOGOSKG in the Appendix (Figure 6, 7 and Table 6) for interested readers. While using LogosKG for high-hop retrieval followed by a fine-tuned selector, or even a multi-agent workflow, is an extension, our focus here is to analyze how LOGOSKG enables and characterizes the underlying structural interaction regimes because of its efficiency and scalability.

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. Our analysis shows that KG topology strongly shapes how structured knowledge interacts with LLM reasoning, positioning LOGOSKG as both a high-performance retrieval system and a foundation for future research on KG–LLM integration.

Acknowledgments

This work is supported by U.S. National Library of Medicine, National Institute of Health, under award number R00LM014308.

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 graph reasoning 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 more effective ranking or filtering strategies to highlight the most relevant results.

Finally, LOGOSKG serves as a helpful graph retrieval tool, but it does not guarantee consistent performance improvements in every case, as success depends on both the completeness of the KG and the reasoning of the LLM. In Round 1, we are limited by the graph's coverage; if the KG lacks evidence for a specific symptom-diagnosis pair, retrieval offers little benefit. Similarly, Round 2 relies on the LLM accurately reselecting information. If the model makes incorrect choices or hallucinates during this step, the potential for improvement is limited by the LLM's own reasoning capabilities.

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, Elliot First, Nicholas Pellegrino, Miranda Schnier, John Caskey, Madeline Oguss, Graham Wills, Guanhua Chen, Dmitriy Dligach, and 1 others. 2025a. Evaluating clinical ai summaries with large language models as judges. *npj Digital Medicine*, 8(1):640.

Emma Croxford, Yanjun Gao, Nicholas Pellegrino, Karen Wong, Graham Wills, Elliot First, Miranda Schnier, Kyle Burton, Cris Ebby, Jillian Gorski, and 1 others. 2025b. 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).

H Han, Y Wang, H Shomer, K Guo, J Ding, Y Lei, M Halappanavar, RA Rossi, S Mukherjee, X Tang, and 1 others. Retrieval-augmented generation with graphs (graphrag). arxiv 2024. *arXiv preprint arXiv:2501.00309*.

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*.

Pengcheng Jiang, Cao Xiao, Adam Cross, and Jimeng Sun. 2024. Graphcare: Enhancing healthcare predictions with personalized knowledge graphs. In *12th International Conference on Learning Representations, ICLR 2024*.

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)*.

Saksham Khatwani, He Cheng, Majid Afshar, Dmitriy Dligach, and Yanjun Gao. 2025. Brittleness and promise: Knowledge graph based reward modeling for diagnostic reasoning. *arXiv preprint arXiv:2509.18316*.

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. 2025a. Gpt-5 mini model card. OpenAI Platform Documentation. Accessed: 2026-01-05.

OpenAI. 2025b. Introducing gpt-4.1. OpenAI Blog. Accessed: 2026-01-05.

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*.

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*.

Yuzhang Xie, Hejie Cui, Ziyang Zhang, Jiaying Lu, Kai Shu, Fadi Nahab, Xiao Hu, and Carl Yang. 2025. Kerap: A knowledge-enhanced reasoning approach for accurate zero-shot diagnosis prediction using multi-agent llms. *arXiv preprint arXiv:2507.02773*.

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.

Xinjie Zhao, Moritz Blum, Fan Gao, Yingjian Chen, Boming Yang, Luis Marquez-Carpintero, Mónica Pina-Navarro, Yanran Fu, So Morikawa, Yusuke Iwasawa, and 1 others. 2025. Agentigraph: A multi-agent knowledge graph framework for interactive, domain-specific llm chatbots. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, pages 6757–6761.

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).

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

Table 4 shows that all retrieved results are perfectly overlap, proving the fidelity of LogosKG.

A.2 Entity-level Evaluation Metric

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

$$\begin{aligned} \text{Precision} &= \frac{|\,pred \cap gold\,|}{|\,pred\,|}, \\ \text{Recall} &= \frac{|\,pred \cap gold\,|}{|\,gold\,|}. \end{aligned} \tag{11}$$

We compute precision and recall to assess the effect of KG processing. Round 1 (filtered) and Round 2 (enhanced) use the same computation, which can be formulated as follows. Let $pred_f$ denote the subset of predictions after KG processing (filtered in Round 1, enhanced in Round 2):

$$\text{Precision}_f = \frac{|\,pred_f \cap gold\,|}{|\,pred_f\,|}, \quad \text{Recall}_f = \frac{|\,pred_f \cap gold\,|}{|\,gold\,|}. \tag{12}$$

A.3 Supplementary Sensitivity Analysis

We study the impact of LOGOSKG on LLM performance across varying retrieval depths ($k = 1$–5) in Rounds 1 and 2. Figure 5 details precision, recall, and F1 scores, supplementing the main results in Figure 3.

In Round 1, increasing the retrieval depth allows LOGOSKG to uncover more supporting evidence from KGs. We observe this as an increase in recall; however, due to the inherent incompleteness of the KG, recall remains below the baseline. Conversely, precision consistently exceeds the baseline, as LOGOSKG effectively filters out hallucinated diagnoses using retrieval evidence. To recover important diagnoses missed by the LLM, Round 2 prompts the model to select additional evidence from the LOGOSKG retrieval evidence. Here, we observe that recall surpasses the baseline, though this comes with a slight decrease in precision as the inclusion of more evidence inevitably introduces some noise. Nevertheless, the overall improvement in F1 scores demonstrates that LOGOSKG successfully enhances diagnostic performance. These results underscore the critical role that KG structure plays in determining how much LLMs can benefit from integration.

A.4 Diagnosis Quality Evaluation using PDSQI-9 Criteria

Table 5 shows that LOGOSKG generally improves diagnosis quality across models and datasets. The gains are particularly evident in Comprehensibility, Succinctness, and Organization, suggesting that high-hop retrieval yields more concise and comprehensive outputs. Specifically, improvements in Succinctness occur primarily in Round 1, showing the effectiveness of LOGOSKG in removing hallucinated diagnoses. In contrast, the gains in Comprehensibility during Round 2 confirm the system's ability to complement incomplete diagnoses by adding relevant evidence from KGs. Improvements are modest on ProbSum, where most concepts are already covered by the input notes (as shown in Figure 2), but are much stronger on DDXPlus, which requires retrieving evidence at higher hops. Notably, models such as GPT-4.1 and Llama-3.1-8B-Instruct benefit the most, showing improvements across multiple metrics and confirming that LOGOSKG effectively enhances LLM diagnostic quality.

A.5 Case Study of LLM Diagnoses with LOGOSKG Filtering and Enhancement

Figure 6 shows how LOGOSKG works in practice on a real clinical case. The baseline LLM generates ten diagnoses from the patient's progress note, including Plasma Cell Dyscrasia, Hypotension, Afib, ESRD, DM2, and CAD. When we filter these with the KG in Round 1, only CAD remains, while the other five fail to connect to clinical entities extracted from the note within $k = 3$ hops. This strict filtering removes incorrect predictions but also removes potentially relevant diagnoses.

Round 2 addresses this limitation by finding 715 candidate diseases from the KG based on the patient's clinical findings. We then ask the LLM to select the most relevant candidates from this larger set. The model identifies three important conditions: SEPSIS, RESPIRATORY FAILURE, and Hypoxia due to hospital-acquired pneumonia. Notably, these match well with the gold standard, which includes Chronic Respiratory Failure and sepsis-related complications, even though none of them appeared in the baseline prediction.

The final output combines the Round 1 filtered baseline diagnosis with the Round 2 selected candidates, achieving much better coverage than either the baseline alone or the Round 1 result. This example also illustrates the motivation of proposing LOGOSKG: KGs can expand the search space beyond what LLMs initially consider, while LLM reasoning helps select which candidates are actually relevant to the patient.

A.6 Analysis: Fine-Tuned Models with KG Enhancement

Figure 7 shows how task-specific fine-tuned models perform with LOGOSKG enhancement across different hop distances. Unlike zero-shot models that gain substantially from KG guidance, fine-tuned models show minimal improvements or even worse performance. Round 1 KG filtering reduces recall while providing only small precision gains, making the trade-off not worthwhile. Round 2 enhancement helps partially but cannot restore baseline performance across both datasets and KGs. This differs sharply from zero-shot scenarios, where the two-round approach successfully balances precision and recall.

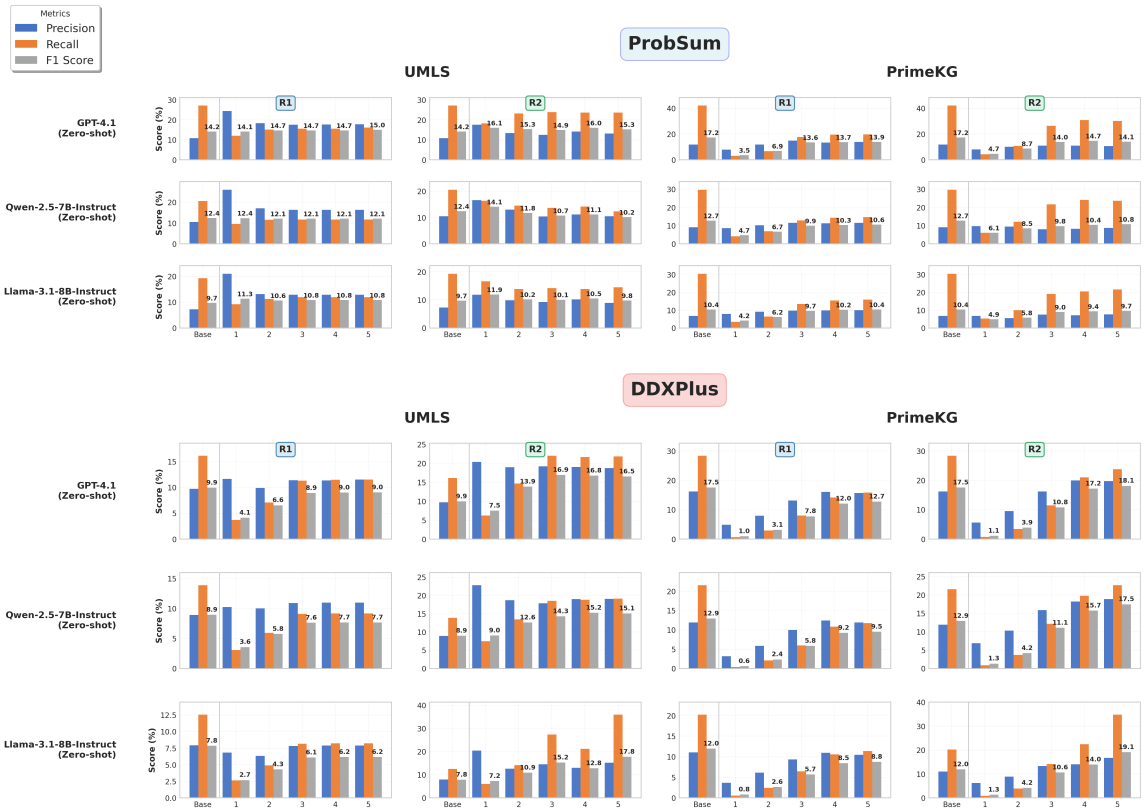

Figure 5: Performance comparison of KG filtering (Round 1) and enhancement (Round 2) across varying hop distances ($k = 1$ to $k = 5$) on ProbSum and DDXPlus datasets using UMLS and PrimeKG. Precision, recall, and F1 score metrics are shown for **GPT-4.1, Qwen-2.5-7B-Instruct, and Llama-3.1-8B-Instruct** in the Zero-shot setting, with baseline performance included for reference.

Dataset	Metric	UMLS									PrimeKG								
		GPT-4.1			Qwen-2.5-7B-Instruct			Llama-3.1-8B-Instruct			GPT-4.1			Qwen-2.5-7B-Instruct			Llama-3.1-8B-Instruct		
		B	R1	R2	B	R1	R2	B	R1	R2	B	R1	R2	B	R1	R2	B	R1	R2
ProbSum	Accuracy extractive	3.42	3.28	1.52	1.96	2.83↑	1.19	1.54	2.74↑	1.59↑	3.19	3.48↑	1.76	1.93	2.56↑	1.57	1.58	2.60↑	1.65↑
	Thoroughness	3.17	1.07	1.50	2.36	1.29	1.21	2.22	1.31	1.28	3.06	1.08	1.46	2.45	1.14	1.34	2.11	1.16	1.28
	Usefulness	3.61	1.70	1.85	2.53	1.88	1.35	2.09	2.03	1.72	3.52	1.52	1.81	2.58	1.56	1.46	2.12	1.66	1.44
	Organization	3.38	2.86	1.98	2.82	2.93↑	1.93	2.31	3.16↑	2.29	3.27	2.81	2.21	2.81	2.59	1.97	2.31	2.82↑	2.08
	Comprehensibility	4.62	4.59	2.42	3.45	4.26↑	3.28	3.33	4.50↑	3.38↑	4.53	4.19	2.76	3.49	3.79↑	2.90	3.34	4.15↑	3.37↑
	Succinctness	3.65	4.76↑	2.05	2.79	4.56↑	3.70↑	2.21	4.78↑	3.12↑	3.61	4.28↑	2.87	2.89	4.35↑	2.76	2.21	4.62↑	3.13↑
	Synthesis	3.24	2.63	2.02	2.97	2.61	2.15	2.73	2.73	2.46	3.16	2.27	2.28	3.02	2.24	2.18	2.74	2.49	2.03
	Average	3.58	2.98	1.91	2.70	2.91↑	2.12	2.35	3.04↑	2.26	3.48	2.80	2.16	2.74	2.60	2.03	2.34	2.78↑	2.14
DDXPlus	Accuracy extractive	2.54	2.62↑	3.16↑	2.36	2.11	2.62↑	1.83	2.17↑	1.37	2.52	2.73↑	2.54↑	2.39	1.73	2.26	1.85	2.05↑	1.71
	Thoroughness	2.70	1.27	1.96	2.40	1.18	1.62	1.96	1.17	1.75	2.76	1.39	1.52	2.26	1.08	1.36	1.99	1.24	1.52
	Usefulness	2.93	1.95	2.79	2.73	1.51	2.45	2.22	1.60	1.63	2.84	2.01	2.24	2.65	1.33	1.89	2.34	1.63	1.77
	Organization	3.09	2.83	3.29↑	2.85	2.11	3.03↑	2.72	2.15	1.80	3.05	2.86	2.87	2.90	1.76	2.66	2.65	2.15	2.30
	Comprehensibility	4.31	3.87	4.53↑	3.87	3.22	4.32↑	3.78	3.32	3.44	4.25	3.98	3.96	3.88	2.79	3.93↑	3.84	3.01	4.01↑
	Succinctness	2.84	4.31↑	4.74↑	2.79	3.98↑	4.64↑	2.48	4.15↑	2.07	2.85	4.45↑	4.53↑	2.85	3.92↑	4.55↑	2.50	3.65↑	3.88↑
	Synthesis	3.05	2.20	3.05↑	3.00	1.75	2.85	2.93	1.88	2.04	3.04	2.34	2.53	2.99	1.55	2.51	2.97	1.80	2.45
	Average	3.07	2.72	3.36↑	2.86	2.27	3.08↑	2.56	2.35	2.01	3.04	2.82	2.88	2.85	2.02	2.74	2.59	2.22	2.52

Table 5: PDSQI-9 evaluation comparing Baseline (B), LogosKG Round 1 (R1), and Round 2 (R2) at $k = 5$. Results for base models across UMLS and PrimeKG. ↑ indicates improvement over baseline.

Why KG Enhancement Cannot Beat Task-Specific Fine-Tuning? This performance gap shows three key problems. First, *knowledge mismatch*: fine-tuned models learn disease connections directly from labeled clinical data, picking up patterns and relationships that may not exist in general biomedical KGs. The KG shows how concepts relate in theory rather than how diseases appear in actual clinical practice. Second, *filtering removes good predictions*: Round 1's strict KG checking throws out predictions that the model correctly learned during training, simply because they don't connect strongly to extracted entities in the KG. Third, *searching space complexity*: While Round 2 finds hundreds of candidate diseases, the model must select from this large set without train-

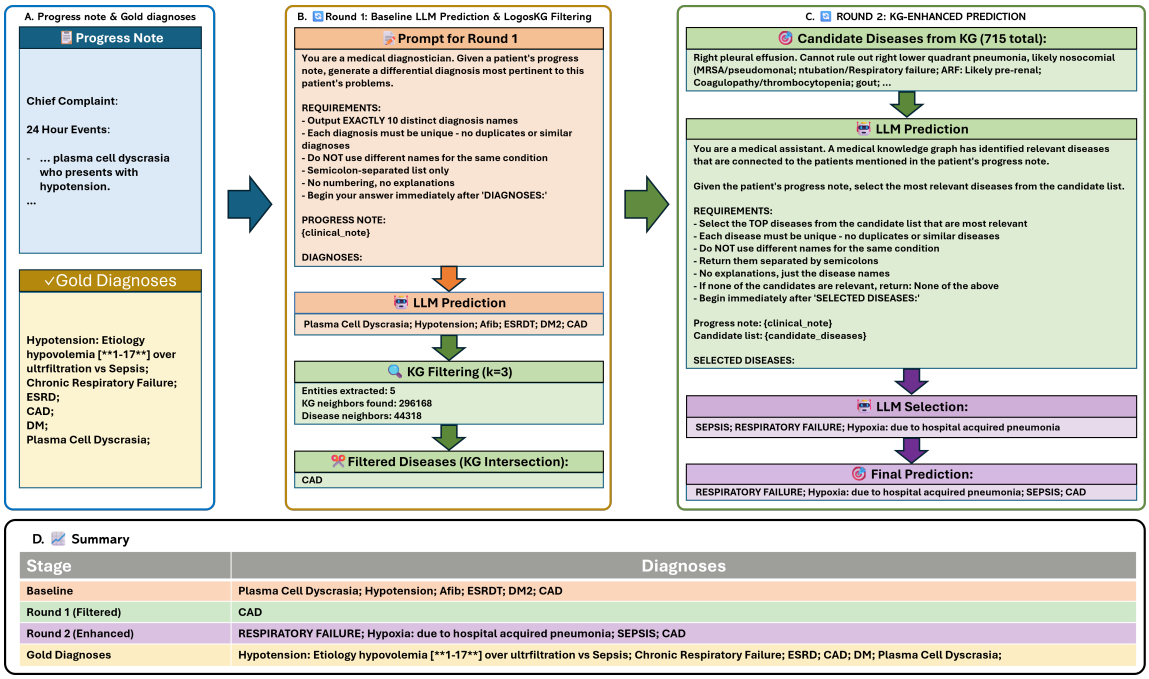

Figure 6: Illustrative example of the LogosKG two-round prediction pipeline on a clinical case from ProbSum. **(A)** Clinical input consists of a patient progress note with the gold standard diagnosis. **(B)** Round 1: The baseline LLM generates differential diagnoses, which are then filtered by validating against KG neighbors retrieved within $k = 3$ hops. **(C)** Round 2: Additional candidate diseases from the KG are presented to the LLM for selection, and the final prediction combines filtered Round 1 diagnoses with enhanced Round 2 candidates. **(D)** Summary table comparing baseline, Round 1 (KG-filtered), Round 2 (KG-enhanced), and gold standard diagnoses.

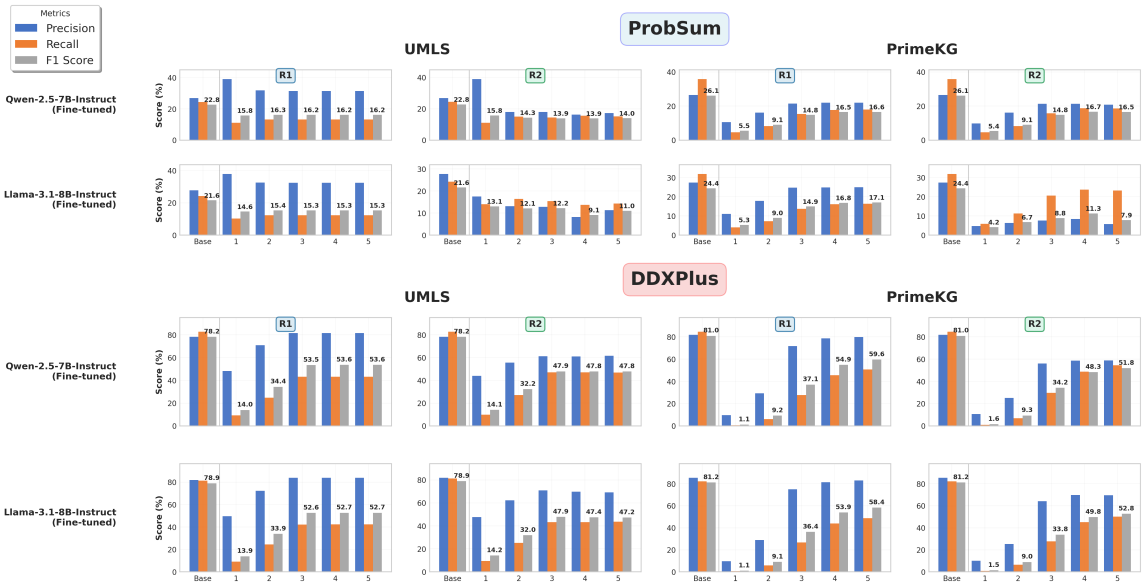

Figure 7: Performance comparison of KG filtering (Round 1) and enhancement (Round 2) across varying hop distances ($k = 1$ to $k = 5$) on ProbSum and DDXPlus datasets using UMLS and PrimeKG. Precision, recall, and F1 score metrics are shown for **Qwen-2.5-7B-Instruct (Fine-tuned) and Llama-3.1-8B-Instruct (Fine-tuned)**, with baseline performance included for reference.

ing.

When Does LogosKG Help? Our results show that LogosKG works best in zero-shot and few-shot settings where models lack task-specific training. While fine-tuning performs well on target domains, it narrows model knowledge and reduces generalization. LogosKG takes a different approach: it maintains broad capabilities while

improving domain-specific performance through external knowledge. This makes it particularly valuable for applications covering diverse medical specialties or rare diseases with limited training data, where preserving model flexibility is as important as achieving strong performance. When task-specific data is scarce or cross-domain generalization is prioritized, KG enhancement offers a practical alternative to fine-tuning.

k	Strategy	Setting	UMLS			PrimeKG		
			P	R	F1	P	R	F1
Baseline			8.58	23.38	11.74	8.27	32.09	12.18
1	Similarity	$\tau = 0.7$	18.02	14.81	14.72	11.61	4.66	5.86
		$\tau = 0.8$	18.51	14.45	14.70	12.05	4.48	5.71
		$\tau = 0.9$	21.00	14.03	15.28	12.41	4.48	5.77
	Top-N	$N = 10$	17.43	15.76	14.98	12.58	6.14	7.38
		$N = 15$	17.08	16.66	15.16	12.58	6.14	7.38
		$N = 20$	17.07	16.74	15.18	12.58	6.14	7.38
		$N = 25$	17.07	16.74	15.18	12.58	6.14	7.38
		$N = 30$	17.07	16.74	15.18	12.58	6.14	7.38
2	Similarity	$\tau = 0.7$	11.97	15.83	12.41	10.31	5.25	6.20
		$\tau = 0.8$	12.62	15.09	12.41	11.87	4.99	6.15
		$\tau = 0.9$	17.02	14.54	14.07	12.17	4.78	6.08
	Top-N	$N = 10$	12.97	16.06	13.17	10.75	12.54	9.93
		$N = 15$	11.36	18.70	13.06	10.69	13.41	10.12
		$N = 20$	11.00	19.65	13.07	10.61	13.59	10.08
		$N = 25$	10.96	20.03	13.11	10.60	13.59	10.07
		$N = 30$	10.96	20.03	13.11	10.59	13.59	10.06
3	Similarity	$\tau = 0.7$	11.61	16.15	12.23	9.69	6.09	6.71
		$\tau = 0.8$	12.10	15.17	12.12	12.04	5.55	6.65
		$\tau = 0.9$	16.65	14.61	13.94	11.94	4.82	6.06
	Top-N	$N = 10$	12.75	16.16	13.03	9.73	18.30	11.40
		$N = 15$	11.10	19.33	13.02	9.80	24.16	12.64
		$N = 20$	10.64	20.46	12.97	9.60	25.68	12.73
		$N = 25$	10.55	20.89	12.99	9.59	26.20	12.78
		$N = 30$	10.54	20.89	12.98	9.51	26.20	12.70
4	Similarity	$\tau = 0.7$	11.51	16.17	12.19	9.23	6.21	6.64
		$\tau = 0.8$	12.01	15.19	12.08	11.31	5.67	6.63
		$\tau = 0.9$	16.57	14.63	13.93	11.90	4.87	6.09
	Top-N	$N = 10$	12.75	16.00	12.97	9.03	17.22	10.61
		$N = 15$	11.02	19.21	12.93	9.42	25.17	12.54
		$N = 20$	10.57	20.54	12.94	8.98	28.04	12.56
		$N = 25$	10.45	20.97	12.93	8.98	29.15	12.71
		$N = 30$	10.45	21.03	12.95	8.92	29.30	12.66
5	Similarity	$\tau = 0.7$	11.31	16.34	12.12	9.30	6.34	6.74
		$\tau = 0.8$	11.78	15.36	12.01	11.48	5.79	6.78
		$\tau = 0.9$	16.33	14.80	13.90	11.90	4.91	6.13
	Top-N	$N = 10$	12.71	16.09	12.99	9.01	17.19	10.58
		$N = 15$	10.83	19.12	12.77	9.51	25.47	12.70
		$N = 20$	10.41	20.78	12.88	9.03	28.81	12.70
		$N = 25$	10.23	21.15	12.80	8.97	30.12	12.80
		$N = 30$	10.23	21.27	12.82	8.94	30.40	12.79

Table 6: Retrieval refinement performance on UMLS and PrimeKG. Results are grouped by refinement strategy (Similarity Threshold τ vs. Top-N Selection) with varying k.

A.7 Evaluation of Refinement Techniques for Filtering in Round 1

We also optimized the retrieval results to reduce noise and examined whether refinement could improve performance. Following a setup common in prior work (e.g., Dr.Knows (Gao et al., 2025) and GraphRAG (Han et al.)), we used SapBERT to compute the maximum cosine similarity between candidate diagnoses and query entities. All results in Table 6 were computed using GPT-5-mini with ProbSum. To strictly investigate the impact of ranking and pruning on initial retrieval, these experiments utilize only LogosKG's Round 1 filtering output, excluding the Round 2 expansion. We explored two refinement strategies: a threshold-based method ($\tau = 0.7, 0.8, 0.9$) and a Top-N selection method ($N = 10, 15, 20, 25, 30$).

We found that the threshold method worked well for reducing noise on UMLS, but it struggled with PrimeKG. It appears that the significant performance drop for PrimeKG was due to the method being too aggressive, likely filtering out useful evidence. On the other hand, the Top-N strategy proved to be much more reliable. By keeping a fixed number of the best candidates instead of relying solely on a strict score, it struck a better balance between filtering out noise and retaining key information. This approach led to consistent gains in Precision and F1 score, especially in later rounds where managing errors is important.

Model	KG	Hops	Precision	Recall	F1
Llama-3.1-8B-Instruct	UMLS	$k = 1$	0.1765	0.2083	0.1590
		$k = 2$	0.1644	0.1763	0.1377
		$k \leq 2$	0.1112	0.1101	0.0900
		$k \leq 3$	0.1011	0.1400	0.0984
	PrimeKG	$k = 1$	0.0630	0.1178	0.0706
		$k = 2$	0.0679	0.1081	0.0701
		$k \leq 2$	0.0579	0.1061	0.0603
		$k \leq 3$	0.0371	0.0800	0.0421
Qwen-2.5-7B-Instruct	UMLS	$k = 1$	0.1269	0.2565	0.1385
		$k = 2$	0.1590	0.1504	0.1271
		$k \leq 2$	0.1430	0.0853	0.0914
		$k \leq 3$	0.0935	0.0382	0.0488
	PrimeKG	$k = 1$	0.0450	0.1128	0.0517
		$k = 2$	0.0660	0.1301	0.0739
		$k \leq 2$	0.0723	0.1244	0.0733
		$k \leq 3$	0.0386	0.0526	0.0379

Table 7: Performance of KG-augmented supervised fine-tuning on UMLS and PrimeKG. We compare Llama-3.1-8B-Instruct and Qwen-2.5-7B-Instruct models trained with specific ($k = 1, 2$) versus cumulative ($k \leq 2, 3$) retrieval contexts.

A.8 LogosKG-Augmented Supervised Fine-Tuning

We also investigated whether LogosKG could enhance training for downstream medical diagnosis. We approached this by retrieving evidence in two distinct modes: "at specific hops" (isolating entities at exact distances) and "within specific hops"

(aggregating all entities up to a certain distance). We then augmented the original training data by appending these retrieved entities to the patient progress notes. Using this enriched input, we performed Supervised Fine-Tuning to teach the models to identify the gold-standard diagnosis within the provided context.

Our results in Table 7 clearly show that restricting retrieval to specific hops ($k = 1, 2$) consistently outperforms cumulative strategies ($k \leq 2, 3$). While cumulative retrieval captures more information, in practice, it overwhelms the model with noise. For example, for each sample in ProbSum, the number of diagnosis entities retrieved from PrimeKG increases drastically with depth, averaging 215 within 2 hops and rising to 42,533 within 5 hops. This noise actually drags performance below the baseline of standard supervised fine-tuning, which has the advantage of a cleaner and more focused search space. These findings underscore that effective KG-LLM training is not trivial. As Khatwani et al. (2025) pointed out, simple fine-tuning is insufficient. Instead, the field is moving toward iterative, multi-agent frameworks, an exciting direction recently explored by Zhao et al. (2025); Zuo et al. (2025); Xie et al. (2025). We consider this as a key area for future work, where LOGOSKG can serve as the efficient retrieval backbone necessary for such advanced systems.