ARE HYPERVECTORS ENOUGH? SINGLE-CALL LLM REASONING OVER KNOWLEDGE GRAPHS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent advances in large language models (LLMs) have enabled strong reasoning over both structured and unstructured knowledge. When grounded on knowledge graphs (KGs), however, prevailing pipelines rely on heavy neural encoders to embed and score symbolic paths or on repeated LLM calls to rank candidates, leading to high latency, GPU cost, and opaque decisions that hinder faithful, scalable deployment. We propose `PathHD`, a lightweight and encoder-free KG reasoning framework that replaces neural path scoring with *hyperdimensional computing* (HDC) and uses only a *single* LLM call per query. `PathHD` encodes relation paths into block-diagonal *GHRR* hypervectors, ranks candidates with blockwise cosine similarity and Top-K pruning, and then performs a one-shot LLM adjudication to produce the final answer together with cited supporting paths. Technically, `PathHD` is built on three ingredients: (i) an order-aware, non-commutative binding operator for path composition, (ii) a calibrated similarity for robust hypervector-based retrieval, and (iii) a one-shot adjudication step that preserves interpretability while eliminating per-path LLM scoring. On WebQSP, CWQ, and GrailQA, `PathHD` (i) attains comparable or better Hits@1 than strong neural baselines while using *one* LLM call per query, (ii) reduces end-to-end latency by $40-60\%$ and GPU memory by **3-5**$\times$ thanks to encoder-free retrieval, and (iii) delivers faithful, path-grounded rationales that improve error diagnosis and controllability. These results indicate that carefully designed HDC representations provide a practical substrate for efficient KG-LLM reasoning, offering a favorable accuracy-efficiency-interpretability trade-off.

1 INTRODUCTION

Large Language Models (LLMs) have rapidly advanced reasoning over both text and structured knowledge. Typical pipelines follow a *retrieve–then–reason* pattern: they first surface evidence (documents, triples, or relation paths), then synthesize an answer using a generator or a verifier (Lewis et al., 2020; Press et al., 2022; Yao et al., 2023; Wei et al., 2022; Yao et al., 2024). In knowledge-graph question answering (KGQA), this often becomes *path-based reasoning*: systems construct candidate relation paths that connect the topic entities to potential answers and pick the most plausible ones for final prediction (Sun et al., 2018; Jiang et al., 2022; 2023; 2024; Luo et al., 2023). While these approaches obtain strong accuracy on WebQSP, CWQ, and GrailQA, they typically depend on heavy neural encoders (e.g., Transformers or GNNs) or repeated LLM calls to rank paths, which makes them slow and expensive at inference time, especially when many candidates must be examined.

Figure 1 highlights two recurring issues in KG–LLM reasoning. ❶ **Path–query mismatch:** Order-insensitive encodings, weak directionality, and noisy similarity often favor superficially related yet misaligned paths, blurring the question's intended relation. ❷ **Per-candidate LLM scoring:** Many systems score candidates sequentially, so latency and token cost grow roughly linearly with set size; batching is limited by context/API, and repeated calls introduce instability, yet models can still overweight long irrelevant chains, hallucinate edges, or flip relation direction. Most practical pipelines first detect a topic entity, enumerate $10 \sim 100$ length-1–4 paths, then score each with a neural model or LLM, sending top paths to a final step (Sun et al., 2018; Luo et al., 2023; Jiang et al., 2024). This hard-codes two inefficiencies: (i) *neural scoring dominates latency* (fresh encoding/prompt per candidate), and (ii) *loose path semantics* (commutative/order-insensitive encoders conflate *founded_by*$\rightarrow$*CEO_of* with its reverse), which compounds on compositional/long-hop questions.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

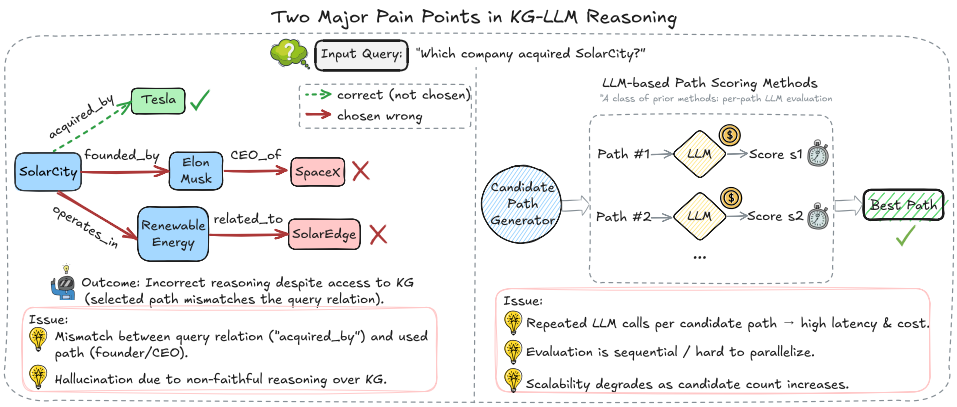


Figure 1: **Two major pain points in KG-LLM reasoning. Left:** A path-based KGQA system selects a candidate path whose relation sequence does not match the query relation ("acquired_by"), leading to an incorrect answer even though the KG contains the correct evidence. **Right:** LLM-based scoring evaluates each candidate path in a separate LLM call, which is sequential, hard to parallelize, and incurs high latency and token cost as the candidate set grows.

Hyperdimensional Computing (HDC) offers a different lens: represent symbols as long, nearly-orthogonal *hypervectors* and manipulate structure with algebraic operations such as *binding* and *bundling* (Kanerva, 2009; Plate, 1995). HDC has been used for fast associative memory, robust retrieval, and lightweight reasoning because its core operations are elementwise or blockwise and parallelize extremely well on modern hardware (Frady et al., 2021). Encodings tend to be noise-tolerant and compositional; similarity is computed by simple cosine or dot product; and both storage and computation scale linearly with dimensionality. Crucially for KGQA, HDC supports *order-sensitive* composition when the binding operator is non-commutative, allowing a path like $r_1 \to r_2 \to r_3$ to be distinguished from its permutations while remaining a single fixed-length vector. This makes HDC a promising substrate for ranking many candidate paths without invoking a neural model for each one. Throughout this paper, we investigate a central question:

Are hypervectors enough for single-call LLM reasoning over knowledge graphs?

We approach this by developing `PathHD` (**Path H**yper**D**imensional Retrieval), a lightweight retrieval-and-reason framework for efficient KGQA with LLMs. We represent each KG relation as a block-diagonal unitary hypervector and encode a candidate path by *order-sensitive* GHRR binding (Yeung et al., 2024); non-commutativity captures relation order, while directed relation symbols encode edge direction. We encode the question into the same space to obtain a *query hypervector*. We then score *all* candidate paths using cosine similarity and keep only the top-K with a simple parallel selection. Finally, rather than calling an LLM for every candidate, we make *one* LLM call: the model sees the question and the verbalized top-K paths, and returns the answer together with the supporting paths. Overall, `PathHD` addresses both pain points in Figure 1: order-aware binding reduces path-query mismatch, and vector-space scoring removes per-path LLM evaluation, reducing latency and token cost.

Our contributions are as follows:

❶ **A fast, order-aware retriever for KG paths.** We present `PathHD`, which uses GHRR-based, non-commutative binding to encode relation *sequences* into hypervectors and ranks candidates with plain cosine similarity, without learned neural encoders or per-path prompts. This design keeps a symbolic structure while enabling fully parallel scoring with $\mathcal{O}(Nd)$ complexity.

❷ **An efficient one-shot reasoning stage.** `PathHD` replaces many LLM scoring calls with a single LLM adjudication over the top-K paths. This decouples retrieval from generation, lowers token usage, and improves wall-clock latency while remaining interpretable: the model cites the supporting path(s) it used.

❸ **Extensive validation and operator study.** On WebQSP, CWQ, and GrailQA, `PathHD` achieves competitive Hits@1 and F1 with markedly lower inference cost. An ablation on binding operators shows that our block-diagonal (GHRR) binding outperforms commutative binding and circular

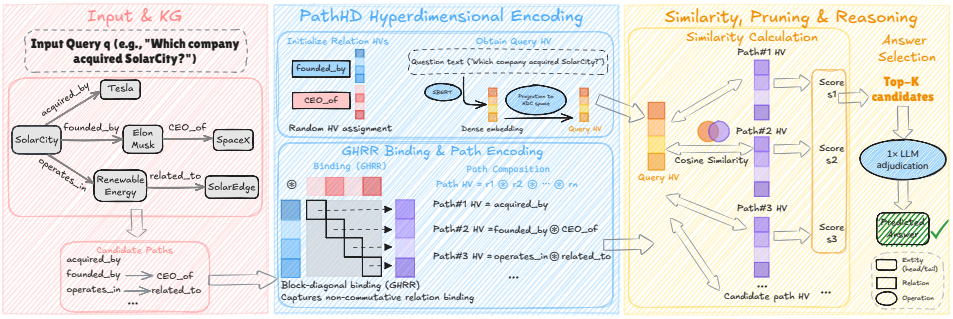


Figure 2: Overview of **PathHD**'s *Plan* → *Encode* → *Retrieve* → *Reason* pipeline. In the main setting, the selected query plan and candidate paths use order-sensitive GHRR encoding; the query-HV inset illustrates a text-projection variant, not the default pipeline. Blockwise similarity and Top-K selection precede a single LLM call.

convolution, and additional studies analyze the impact of top-K pruning and latency–accuracy trade-offs.

2 METHOD

The proposed **PathHD** follows a *Plan* → *Encode* → *Retrieve* → *Reason* pipeline (Figure 2). (i) We first generate or select relation *plans* that describe how an answer can be reached (schema enumeration optionally refined by a light prompt). (ii) Non-commutative GHRR binding enables order-sensitive path composition, while directed relations encode edge direction. (iii) We compute a blockwise cosine similarity in the hypervector space and apply Top-K pruning. (iv) Finally, a *single* LLM call produces the answer with path-based explanations. This design keeps the heavy lifting in cheap vector operations, delegating semantic adjudication to one-shot LLM reasoning.

2.1 PROBLEM SETUP & NOTATION

Given a question q, a knowledge graph (KG) $\mathcal{G}$, and a set of relation schemas $\mathcal{Z}$, the goal is to predict an answer a. Formally, we write $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{R})$, where $\mathcal{V}$ is the set of entities, $\mathcal{R}$ is the set of relation types, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{R} \times \mathcal{V}$ is the set of directed edges (e, r, e'). We denote entities by $e \in \mathcal{V}$ and relations by $r \in \mathcal{R}$. A relation schema $z \in \mathcal{Z}$ is a sequence of relation types $z = (r_1, \ldots, r_\ell)$. Instantiating a schema z on $\mathcal{G}$ yields concrete KG paths of the form $(e_0, r_1, e_1, \ldots, r_\ell, e_\ell)$ such that $(e_{i-1}, r_i, e_i) \in \mathcal{E}$ for all i. For a given question q, we denote by $\mathcal{P}(q)$ the set of candidate paths instantiated from schemas in $\mathcal{Z}$ and by $N = |\mathcal{P}(q)|$ its size. We write d for the dimensionality of the hypervectors used to represent relations and paths.

A key challenge is to efficiently locate a small set of *plausible* paths for q from this large candidate pool, and then let an LLM reason over only those paths. A summary of the notation throughout the paper can be found in Section A.

2.2 HYPERVECTOR INITIALIZATION

We work in a Generalized Holographic Reduced Representations (GHRR) Yeung et al. (2024) space. Each atomic symbol x (relation or, optionally, entity) is assigned a d-dimensional hypervector $\mathbf{v}_x \in \mathbb{C}^d$ constructed as a block vector of unitary matrices:

$$\mathbf{v}_x \;=\; [A_1^{(x)}; \ldots; A_D^{(x)}],\; A_j^{(x)} \in \mathrm{U}(m),\;\; d = Dm^2. \tag{1}$$

Following the original GHRR experimental construction, each block is initialized in $Q\Lambda$ form:

$$A_j^{(x)} = Q_j^{(x)} \Lambda_j^{(x)}, \qquad Q_j^{(x)} \in \mathrm{U}(m), \qquad \Lambda_j^{(x)} = \mathrm{diag}\big(e^{i\phi_{j,1}^{(x)}}, \ldots, e^{i\phi_{j,m}^{(x)}}\big), \quad \phi_{j,\ell}^{(x)} \sim \mathrm{Unif}[0, 2\pi). \tag{2}$$

Here $Q_j^{(x)}$ is generally non-diagonal and unitary, while $\Lambda_j^{(x)}$ supplies random phases. Unlike Λ alone, $Q\Lambda$ blocks need not commute and can encode relation order; normalization occurs only within

similarity. With random unit vectors in a high-dimensional space, different symbols are almost orthogonal, and this effect becomes stronger as the dimension grows (Prop. 1). We sample these hypervectors once and then keep them fixed. The retriever introduces no trainable parameters.

Query hypervector. For a question q, we obtain a query hypervector in two ways, depending on the planning route used in Section 2: (i) *plan-based*-encode the selected relation plan $z_q = (r_1, \ldots, r_\ell)$ using the same GHRR binding as paths (see Eq. (4)); or (ii) *text-projection*—embed q with a sentence encoder (e.g., SBERT) to $\mathbf{h}_q \in \mathbb{R}^{d_t}$ and project it to the HDC space using a fixed random linear map $P \in \mathbb{R}^{d \times d_t}$, then block-normalize:

$$\mathbf{v}_q \;=\; \mathcal{N}_{\text{block}}\big(P\,\mathbf{h}_q\big). \tag{3}$$

Both choices produce a query hypervector compatible with GHRR scoring. Unless otherwise specified, we use the plan-based encoding in all main experiments, and report the text-projection variant only in ablations (Section J.1.4).

2.3 GHRR Binding and Path Encoding

A GHRR hypervector is a block vector $\mathbf{H} = [A_1; \ldots; A_D]$ with $A_j \in \mathrm{U}(m)$. Given two hypervectors $\mathbf{X} = [X_1; \ldots; X_D]$ and $\mathbf{Y} = [Y_1; \ldots; Y_D]$, we define the block-wise binding operator $\circledast$ and the encoding of a length-ℓ relation path $z = (r_1, \ldots, r_\ell)$ by:

$$\mathbf{v}_z = \mathbf{v}_{r_1} \circledast \mathbf{v}_{r_2} \circledast \cdots \circledast \mathbf{v}_{r_\ell},\; \mathbf{X} \circledast \mathbf{Y} = [X_1 Y_1; \ldots; X_D Y_D], \tag{4}$$

Applied left-to-right, generally non-diagonal $Q\Lambda$ blocks need not commute ($X_j Y_j \neq Y_j X_j$), enabling order-sensitive composition; directed relation symbols encode edge direction. Scoring normalization changes neither.

Properties and choice of binding operator. Although `PathHD` uses only forward binding, unnormalized unitary blocks invert exactly: if $Z_j = X_j Y_j$, then $X_j = Z_j Y_j^*$ and $Y_j = X_j^* Z_j$. Normalized copies are used only for scoring. This property supports inspection of composed paths and path-level rationales.

Classical HDC bindings (XOR, element-wise multiplication, circular convolution) are *commutative*, which collapses $r_1 \to r_2$ and $r_2 \to r_1$ to similar codes and hurts ordered path reasoning. GHRR is non-commutative, invertible at the block level, and offers higher representational capacity via unitary blocks, leading to better discrimination between paths of the same multiset but different order. We empirically validate this choice in the ablation study (Table 3), where GHRR consistently outperforms commutative bindings. Additional background on binding operations is provided in Section K.

2.4 Query & Candidate Path Construction

We obtain a query plan z_q via schema-based enumeration on the relation-schema graph (depth $\leq L_{\max}$). In all main experiments reported in Section 3, this planning stage is purely symbolic: we do *not* invoke any LLM beyond the single final reasoning call in Section 2.6. The query hypervector $\mathbf{v}_q$ is then constructed from the selected plan z_q using the plan-based encoding described above (Eq. (4) or, for text projection, Eq. (3)).

Given a query plan, candidate paths $\mathcal{Z}$ are instantiated from the KG either by matching plan templates to existing edges or by a constrained BFS with beam width B, both of which yield symbolic paths. These paths are then deterministically encoded into hypervectors and scored by our HDC module (Sec. 2.5). An optional lightweight prompt-based refinement of schema plans is described in the appendix as an extension; it is *not* used in our main experiments and does not change the single-call nature of the system.

2.5 HD Retrieval: Blockwise Similarity and Top-K

Let $\langle A, B \rangle_F := \mathrm{tr}(A^* B)$ be the Frobenius inner product. Given two GHRR hypervectors $\mathbf{X} = [X_j]_{j=1}^D$ and $\mathbf{Y} = [Y_j]_{j=1}^D$, we define the *blockwise cosine similarity*

$$\mathrm{sim}(\mathbf{X}, \mathbf{Y}) = \frac{1}{D} \sum_{j=1}^{D} \frac{\Re \langle X_j, Y_j \rangle_F}{\|X_j\|_F \, \|Y_j\|_F}. \tag{5}$$

For each candidate $z \in \mathcal{Z}$ we compute $\mathrm{sim}(\mathbf{v}_q, \mathbf{v}_z)$ and (optionally) apply a calibrated score

$$s(z) \;=\; \mathrm{sim}(\mathbf{v}_q, \mathbf{v}_z) \;+\; \alpha\,\mathrm{IDF}(z) \;-\; \beta\,\lambda^{|z|}, \tag{6}$$

where $\mathrm{IDF}(z)$ is a simple inverse-frequency weight on relation schemas. Let $\mathrm{schema}(z)$ denote the relation schema of path z and $\mathrm{freq}(\mathrm{schema}(z))$ be the number of training questions whose candidate sets contain at least one path with the same schema. With N_{train} the total number of training questions, we define:

$$\mathrm{IDF}(z) = \log\left(1 + \frac{N_{\text{train}}}{1 + \mathrm{freq}(\mathrm{schema}(z))}\right). \tag{7}$$

Thus, frequent schemas (large $\mathrm{freq}(\mathrm{schema}(z))$) receive a smaller bonus, while rare schemas receive a larger one. All similarity scores and calibrated scores can be computed fully in parallel over candidates, with overall cost $\mathcal{O}(|\mathcal{Z}|d)$.

2.6 One-shot Reasoning with Retrieved Paths

Putting the pieces together, `PathHD` turns a question into (i) a schema-level plan z_q, (ii) a set of candidate paths ranked in hypervector space, and (iii) a single LLM call that adjudicates among the top-ranked candidates.

We linearize the Top-K paths into concise natural-language statements and issue a *single* LLM call with a minimal, citation-style prompt (see Table 6 from Section C). The prompt lists the question and the numbered paths, and requires the model to return a short answer, the index(es) of supporting path(s), and a 1-2 sentence rationale. This one-shot format constrains reasoning to the provided evidence, resolves near-ties and direction errors, and keeps LLM usage minimal.

Table 1: LLM usage across stages. A checkmark indicates LLM use in that stage. *Candidate Path Gen.* denotes LLM-based path proposal or expansion; *Scoring* denotes LLM-based candidate ranking; *Reasoning* denotes final answer generation. `PathHD` uses the LLM only once, in the final reasoning stage.

Method	Candidate Path Gen.	Scoring	Reasoning
StructGPT (Jiang et al., 2023)	✓	✓	✓
FiDeLiS (Sui et al., 2024)	✓	✗	✓
ToG (Sun et al., 2023)	✓	✓	✓
GoG (Xu et al., 2024)	✓	✓	✓
KG-Agent (Jiang et al., 2024)	✓	✓	✓
RoG (Luo et al., 2023)	✓	✗	✓
`PathHD`	✗	✗	✓ (1 call)

2.7 Theoretical & Complexity Analysis

We analyze an idealized GHRR concentration surrogate and the cost of `PathHD`.

Proposition 1 (Idealized near-orthogonality and distractor bound)**.** *Under the surrogate in Section E, for query z_q and distractor $z \neq z_q$, the cosine similarity $X = \mathrm{sim}(\mathbf{v}_{z_q}, \mathbf{v}_z)$ from Eq. (5) satisfies, for any $\epsilon > 0$,*

$$\Pr(|X| \geq \epsilon) \leq 2\exp\left(-c\,d\,\epsilon^2\right), \tag{8}$$

for an absolute constant $c > 0$ depending on its sub-Gaussian proxy.

Proof sketch. Equation (5) is then a mean-zero sub-Gaussian average over d effective degrees of freedom, so Bernstein/Hoeffding gives the result. □

This surrogate result is not a guarantee for the implemented $Q\Lambda$ blocks without further distributional assumptions on Q.

Corollary 1 (Capacity with union bound)**.** *Under the same idealized surrogate, let $\mathcal{M}$ be a collection of M distractor paths scored against a fixed query. With probability at least $1 - \delta$,*

$$\max_{z \in \mathcal{M}} \mathrm{sim}(\mathbf{v}_{z_q}, \mathbf{v}_z) \leq \epsilon \text{ whenever } d \;\geq\; \frac{1}{c\,\epsilon^2}\,\log\frac{2M}{\delta}. \tag{9}$$

Thus, within this surrogate, the probability of a false match decays exponentially with the dimension d, and the required dimension scales as $d = \mathcal{O}(\epsilon^{-2}\log M)$ for a target error tolerance ϵ.

Complexity comparison with neural retrievers. Let N be the number of candidates, d the embedding dimension, and L the number of encoder layers used by a neural retriever. A typical neural encoder (e.g., Transformer-based path encoder as in RoG) incurs $\mathcal{O}(NLd^2)$ cost for encoding and scoring. Dense block binding and Eq. (5) cost $\mathcal{O}(|z|Dm^3+Dm^2) = \mathcal{O}(|z|dm+d)$ per candidate. Fixed small m and bounded $|z|$ give $\mathcal{O}(Nd)$ overall and an $\mathcal{O}(Ld)$ leading-order reduction.

Beyond the $\mathcal{O}(Nd)$ vs. $\mathcal{O}(NLd^2)$ compute gap, end-to-end latency is dominated by the number of LLM calls. Table 1 contrasts pipeline stages across methods: unlike prior agents that query an LLM for candidate path generation and sometimes for scoring, `PathHD` defers a single LLM call to the final reasoning step. This design reduces both latency and API cost; empirical results in Section 3.3 confirm the shorter response times.

3 Experiments

We evaluate `PathHD` against state-of-the-art baselines on reasoning accuracy, measure efficiency with a focus on end-to-end latency, and conduct ablations followed by illustrative case studies.

3.1 Datasets, Baselines, and Setup

We evaluate on three standard multi-hop KGQA benchmarks: **WebQuestionsSP (WebQSP)** (Yih et al., 2016), **Complex WebQuestions (CWQ)** (Talmor & Berant, 2018), and **GrailQA** (Gu et al., 2021), all grounded in **Freebase** (Bollacker et al., 2008). These datasets span increasing reasoning complexity (roughly 2-4 hops): WebQSP features simpler single-turn queries, CWQ adds compositional and constraint-based questions, and GrailQA stresses generalization across i.i.d., compositional, and zero-shot splits. Our study is therefore scoped to Freebase-style KGQA, extending `PathHD` to domain-specific KGs are left for future work.

We compare against four families of methods: **embedding-based**, **retrieval-augmented**, **pure LLMs** (no external KG), and **LLM+KG hybrids**. All results are reported on dev (IID) splits under a unified Freebase evaluation protocol using the official *Hits@1* and *F1* scripts, so that numbers are directly comparable across systems. Unless otherwise noted, `PathHD` uses the schema-based planner from Section 2.4 (no LLM calls during planning), the plan-based query hypervector in Eq. (4), and a single LLM adjudication call as described in Section 2.6, with all LLMs and sentence encoders used *off the shelf* (no fine-tuning). Detailed dataset statistics, baseline lists, model choices, and additional training-free hyperparameters are provided in Sections G to I.

3.2 Reasoning Performance Comparison

We evaluate under a unified Freebase protocol with the official *Hits@1*/*F1* scripts on WebQSP, CWQ, and GrailQA (dev, IID); results are in Table 2. Baselines cover classic KGQA (embedding/retrieval), recent LLM+KG systems, and pure LLMs (no KG grounding). Our `PathHD` uses hyperdimensional scoring with GHRR, Top-K pruning, and a *single* LLM adjudication step.

Key observations are as follows. **Obs.❶ SOTA on WebQSP/GrailQA; competitive on CWQ.** `PathHD` attains best WebQSP *Hits@1* (86.2) and best GrailQA *F1* (Overall/IID 86.7/92.4), while staying strong on CWQ (*Hits@1* 71.5, *F1* 65.8), close to the top LLM+KG systems (e.g., GoG 75.2 *Hits@1*; KG-Agent 69.8 *F1*). **Obs.❷ One-shot adjudication rivals multi-step agents.** Compared to RoG ($\sim$12 calls) and Think-on-Graph/GoG/KG-Agent (3–8 calls), `PathHD` matches or exceeds accuracy on WebQSP/GrailQA and remains competitive on CWQ with just *one* LLM call, which reduces error compounding and focuses the LLM on a high-quality shortlist. **Obs.❸ Pure LLMs lag without KG grounding.** Zero/few-shot GPT-4 or ChatGPT underperform LLM+KG systems, e.g., on CWQ GPT-4 *Hits@1* 55.6 vs. `PathHD` 71.5. **Obs.❹ Classic embedding/retrieval trails modern LLM+KG.** KV-Mem, NSM, SR+NSM rank subgraphs well but lack a flexible language component for composing multi-hop constraints, yielding consistently lower scores.

Candidate enumeration strategy. Before turning to efficiency (Section 3.3), we briefly clarify how candidate paths are enumerated, since this affects both accuracy and cost. In our current implementation, we use a deterministic BFS-style enumeration of relation paths, controlled by the maximum depth $L_{\max}$ and beam width B. This choice is (i) simple and efficient, (ii) guarantees coverage of all type-consistent paths up to length $L_{\max}$ under clear complexity bounds, and (iii)

Table 2: **Comparison on Freebase-based KGQA.** Our method `PathHD` follows exactly the same protocol. "–" indicates that the metric was *not reported by the original papers under the Freebase+official-script setting*. We bold the best and underline the second-best score for each metric/column.

Type	Methods	WebQSP		CWQ		GrailQA (F1)	
		Hits@1	F1	Hits@1	F1	Overall	IID
Embedding	KV-Mem (Miller et al., 2016)	46.7	34.5	18.4	15.7	–	–
	EmbedKGQA (Saxena et al., 2020)	66.6	–	45.9	–	–	–
	NSM (He et al., 2021)	68.7	62.8	47.6	42.4	–	–
	TransferNet (Shi et al., 2021)	71.4	–	48.6	–	–	–
Retrieval	GraftNet (Sun et al., 2018)	66.4	60.4	36.8	32.7	–	–
	SR+NSM (Zhang et al., 2022)	68.9	64.1	50.2	47.1	–	–
	SR+NSM+E2E (Zhang et al., 2022)	69.5	64.1	49.3	46.3	–	–
	UniKGQA (Jiang et al., 2022)	77.2	72.2	51.2	49.1	–	–
Pure LLMs	ChatGPT (Ouyang et al., 2022)	67.4	59.3	47.5	43.2	25.3	19.6
	Davinci-003 (Ouyang et al., 2022)	70.8	63.9	51.4	47.6	30.1	23.5
	GPT-4 (Achiam et al., 2023)	73.2	62.3	55.6	49.9	31.7	25.0
LLMs + KG	StructGPT (Jiang et al., 2023)	72.6	63.7	54.3	49.6	54.6	70.4
	ROG (Luo et al., 2023)	<u>85.7</u>	70.8	62.6	56.2	–	–
	Think-on-Graph (Sun et al., 2023)	81.8	76.0	68.5	60.2	–	–
	GoG (Xu et al., 2024)	84.4	–	**75.2**	–	–	–
	KG-Agent (Jiang et al., 2024)	83.3	**81.0**	<u>72.2</u>	**69.8**	<u>86.1</u>	<u>92.0</u>
	FiDeLiS (Sui et al., 2024)	84.4	78.3	71.5	64.3	–	–
	`PathHD`	**86.2**	<u>78.6</u>	71.5	<u>65.8</u>	**86.7**	**92.4**

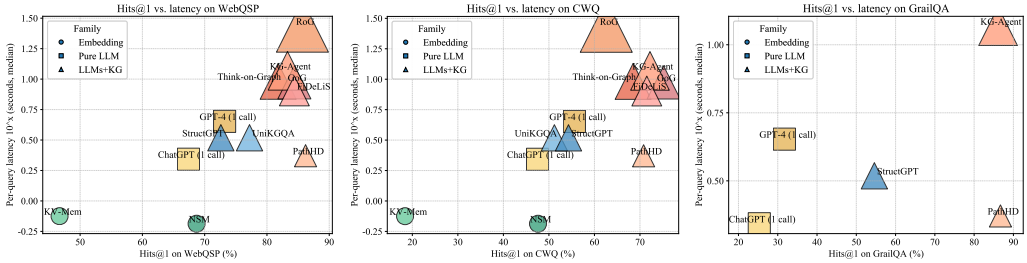


Figure 3: **Visualization of performance and latency.** The x-axis is Hits@1 (%), the y-axis is per-query latency in seconds (median, log scale). Bubble size indicates the average number of LLM calls; marker shape denotes the method family. `PathHD` gives strong accuracy with lower latency than multi-call LLMs+KG baselines.

makes it easy to compare against prior KGQA baselines that also rely on BFS-like expansion. In practice, we choose $L_{\max}$ and B to achieve high coverage of gold answer paths while keeping candidate set sizes comparable to RoG and KG-Agent. More sophisticated, adaptive enumeration strategies (e.g., letting the HDC scores or the LLM guide, which relations to expand next) are an interesting extension, but orthogonal to our core contribution.

3.3 Efficiency and Cost Analysis

We assess end-to-end cost via a *Hits@1–latency* bubble plot (Figure 3) and a lollipop latency chart (Figure 6). In Figure 3, x = *Hits@1*, y = median per-query latency (log-scale); bubble size = average #LLM calls; marker shape = method family. All latencies are measured under a common hardware and LLM-backend setup to enable fair relative comparison. Latencies in Figure 6 follow a shared protocol: in our implementation, each LLM call takes on the order of a few seconds, whereas non-LLM vector/graph operations are typically within 0.3–0.8s. `PathHD` uses vector-space scoring with Top-K pruning and a *single* LLM decision; RoG uses beam search (B=3, depth $\leq$ dataset hops). A factor breakdown (#calls, depth d, beam b, tools) appears in Table 8 (Section I.1).

Key observations are: **Obs.❶ Near-Pareto across datasets.** With comparable accuracy to multi-call LLM+KG systems (Think-on-Graph/GoG/KG-Agent), `PathHD` achieves markedly lower latency due to its single-call design and compact post-pruning candidate set. **Obs.❷ Latency is dominated by #LLM calls.** Methods with 3-8 calls (agent loops) or $\approx d \times b$ calls (beam search) sit higher in

Figure 3 and show longer whiskers in Figure 6; `PathHD` avoids intermediate planning/scoring cost. **Obs.❸ Moderate pruning improves cost-accuracy.** Shrinking the pool before adjudication lowers latency without hurting *Hits@1*, especially on CWQ, where paths are longer. **Obs.❹ Pure LLMs are fast but underpowered.** Single-call GPT-4/ChatGPT has similar latency to our final decision yet lower accuracy, highlighting the importance of structured retrieval and scoring.

3.4 Ablation Study

We analyze the contribution of each module/operation in `PathHD`. Our operation study covers: (1) *Path composition operator*, (2) *Scoring metric for candidate–path ranking* (Section J.1.3), (3) *Path encoding choice* (Section J.1.4), (4) *Single-LLM adjudicator* (Section J.1.1), and (5) *Top-K pruning* (Section J.1.2).

Which path-composition operator works best? We study the effect of the relation-binding operator by keeping everything else fixed (retrieval, scoring, pruning, and the single LLM call) and changing *only* the encoder's path-composition operator. We compare six choices (see Section K): (i) *XOR/bipolar* and (ii) real-valued element-wise product, both *commutative*; (iii) a stronger *commutative* variant that mixes binary and bipolar codes; (iv) *FHRR* (phasors) and (v) *HRR* (circular convolution), which are efficient and largely order-insensitive in practice; and (vi) our GHRR with generally non-diagonal $Q\Lambda$ unitary blocks, whose non-commutative products enable order-sensitive composition. We use the same dimension and normalization for paths of length 1-4. Results in Table 3 show that commutative operators perform worst, HRR/FHRR bring modest improvements, and **GHRR** achieves the best *Hits@1* on WebQSP and CWQ by distinguishing *founded_by→CEO_of* from the reverse.

Table 3: **Effect of the path–composition operator. Note that the Element-wise product operation is using Real-valued.** GHRR yields the best performance.

Operator	WebQSP	CWQ
XOR / bipolar product	83.9	68.8
Element-wise product	84.4	69.2
Comm. bind	84.7	69.6
FHRR	84.9	70.0
HRR	85.1	70.2
GHRR	**86.2**	**71.5**

3.5 Case Study

To better understand how our model performs step-by-step reasoning, we present a representative case from the WebQSP dataset in Table 4. This case highlights the effect of candidate path pruning, while a complementary example illustrating the role of LLM-based adjudication is deferred to the appendix for space (Table 15). Together, these qualitative analyses shed light on how pruning and adjudication improve answer accuracy.

Case 1: Top-K pruning preserves paths aligned with both `film.film.music` and actor cues; the vector-only scorer already picks the correct path, and a single LLM adjudication confirms *Valentine's Day*, illustrating that pruning reduces cost while retaining high-coverage candidates.

Case 2: A vector-only top path misses the actor constraint and yields a false positive (`film.film.edited_by`), but adjudication over the pruned set, now including `performance.actor`, corrects to *The Perks of Being a Wallflower*, showing that LLM adjudication resolves compositional constraints beyond static similarity.

Discussion. As `PathHD` operates in a single-call, fixed-candidate regime, its performance ultimately depends on (i) the Top-K retrieved paths covering at least one valid reasoning chain and (ii) the adjudication LLM correctly ranking these candidates. In practice, we mitigate this by using a relatively generous K (e.g., $K = 3$) and beam widths that yield high coverage of gold paths (see Section K.2), but extreme cases can still be challenging. Note that all LLM reasoning systems that first retrieve a Top-K set of candidates will face the same challenge.

Table 4: **Case study 1 on multi-hop reasoning over WebQSP.** We apply Top-K pruning before invoking the LLM, reducing inference cost while preserving plausible candidates. An additional example (Case study 2) is provided in the appendix (Table 15).

Case 1: *which movies featured Taylor Swift and music by John Debney*	
Top-4 candidates	1) `film.film.music` (0.2567) 2) `person.nationality` → `film.film.country` (0.2524) 3) `performance.actor` → `performance.film` (0.2479) 4) `people.person.languages` → `film.film.language` (0.2430)

Top-K after pruning (K=3)	`film.film.music`
	`person.nationality` → `film.film.country`
	`performance.actor` → `performance.film`

Vector-only (no LLM)	Pick `film.film.music` ✓ directly targets the composer-to-film mapping; relevant for filtering by music.
1×LLM adjudication	*Rationale:* "To find films with both Taylor Swift and music by John Debney, use actor-to-film and music-to-film relations. The chosen path targets the latter directly."
Final Answer / GT	**Valentine's Day (predict) / Valentine's Day ✓**

4 Related Work

4.1 LLM-based Reasoning

LLMs such as GPT, LLaMA, and PaLM have shown strong reasoning abilities across tasks including natural language inference and multi-hop question answering (Radford et al., 2019; Brown et al., 2020; Touvron et al., 2023; Chowdhery et al., 2023; Yang et al., 2018). Recent work improves interpretability by grounding LLM reasoning in symbolic paths over structured knowledge sources (Sun et al., 2018; Cao et al., 2022; Hu et al., 2025), with representative techniques including chain-of-thought prompting (Wei et al., 2022) and self-consistency decoding (Wang et al., 2022). Knowledge graphs are widely used in web search, recommendation, and biomedicine (Ji et al., 2021; Zou, 2020). Recent KG-LLM methods, such as RoG (Luo et al., 2023) and demonstration-based prompting (Khattab et al., 2022), combine graph structure with LLM generation for grounded reasoning. However, their reliance on neural encoders for path matching introduces substantial computation and memory overhead, limiting scalability to large KGs and real-time applications.

4.2 Hyperdimensional Computing

Hyperdimensional Computing (HDC) represents information as fixed-length high-dimensional hypervectors and manipulates symbolic structures through lightweight operations such as binding, bundling, and permutation (Kanerva, 2009; Kanerva et al., 1997; Gayler, 2004). These operations are parallelizable and noise-robust, making HDC attractive for efficient, low-latency computation. HDC has been applied to classification, biosignal processing, language understanding, and graph analytics (Rahimi et al., 2017; Moin et al., 2021; Alonso et al., 2021; Imani et al., 2019a), and has recently been combined with neural models for interpretable and efficient neuro-symbolic learning (Imani et al., 2019b; Rahimi et al., 2017). However, HDC remains underexplored for large-scale KG reasoning with LLMs. We address this gap by using HDC as a scalable and interpretable replacement for neural path matchers in LLM-based reasoning frameworks. Existing KG-LLM frameworks often rely on neural encoders or repeated LLM calls to score candidate paths or subgraphs. `PathHD` instead uses HDC hypervector encoding and similarity, making retrieval encoder-free and training-free. Our goal is not new VSA theory, but to show that HDC can replace neural path scorers while preserving accuracy and improving latency, memory efficiency, and interpretability.

5 Conclusion

`PathHD` is an encoder-free, interpretable framework that ranks order-aware GHRR path hypervectors with simple vector operations and uses one LLM call to return answers with cited paths. On WebQSP, CWQ, and GrailQA, it achieves competitive Hits@1/F1 with lower latency and memory use and debuggable path-grounded rationales. Future work targets domain-specific KGs and tasks beyond QA.

AI USE STATEMENT

Generative AI was used for wording, grammar, and readability improvements; LaTeX formatting and adaptation to the conference template; and assistance in drafting the disclosure statements. All AI-assisted modifications were reviewed by the authors, who take responsibility for the final content.

ETHICS STATEMENT

This work uses existing public Freebase-style KGQA benchmarks and released artifacts, without human-subject experiments or the release of a new pretrained model or dataset. PathHD can reduce inference cost and make decisions more auditable through cited path-level evidence, but its reliability still depends on KG coverage and bias, entity linking, candidate-path coverage, and the final LLM adjudicator. Deployments could otherwise over-trust plausible but incorrect path-grounded answers or expose sensitive attributes in private KGs. Appropriate safeguards include confidence signals and cited evidence, retrieval and provenance logs, access controls and redaction for sensitive graphs, targeted accuracy and bias evaluation, and human oversight in high-stakes settings.

REPRODUCIBILITY STATEMENT

Methodology and experimental settings are documented in Sections 2 and 3.1. Appendix Sections B, D, E, G to I, I.2 and J provides pseudocode, supporting theory and proofs, dataset and baseline descriptions, hardware and hyperparameter settings, calibration details, and additional ablations.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Pedro Alonso, Kumar Shridhar, Denis Kleyko, Evgeny Osipov, and Marcus Liwicki. Hyperembed: Tradeoffs between resources and performance in nlp tasks with hyperdimensional computing enabled embedding of n-gram statistics. In *2021 international joint conference on neural networks (IJCNN)*, pp. 1–9. IEEE, 2021.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 1247–1250, 2008.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 1877–1901, 2020.

Shulin Cao, Jiaxin Shi, Liangming Pan, Lunyiu Nie, Yutong Xiang, Lei Hou, Juanzi Li, Bin He, and Hanwang Zhang. Kqa pro: A dataset with explicit compositional programs for complex question answering over knowledge base. In *Proceedings of the 60th annual meeting of the Association for Computational Linguistics (volume 1: long papers)*, pp. 6101–6119, 2022.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

E. Paxon Frady, Denis Kleyko, and Friedrich T. Sommer. Variable binding for sparse distributed representations: Theory and applications. *Neural Computation*, 33(9):2207–2248, 2021.

Ross W Gayler. Vector symbolic architectures answer jackendoff's challenges for cognitive neuroscience. *arXiv preprint cs/0412059*, 2004.

Yu Gu, Sue Kase, Michelle Vanni, Brian Sadler, Percy Liang, Xifeng Yan, and Yu Su. Beyond iid: three levels of generalization for question answering on knowledge bases. In *Proceedings of the web conference 2021*, pp. 3477–3488, 2021.

Gaole He, Yunshi Lan, Jing Jiang, Wayne Xin Zhao, and Ji-Rong Wen. Improving multi-hop knowledge base question answering by learning intermediate supervision signals. In *Proceedings of the 14th ACM international conference on web search and data mining*, pp. 553–561, 2021.

Haotian Hu, Alex Jie Yang, Sanhong Deng, Dongbo Wang, and Min Song. Cotel-d3x: a chain-of-thought enhanced large language model for drug–drug interaction triplet extraction. *Expert Systems with Applications*, 273:126953, 2025.

Mohsen Imani, Yeseong Kim, Sadegh Riazi, John Messerly, Patric Liu, Farinaz Koushanfar, and Tajana Rosing. A framework for collaborative learning in secure high-dimensional space. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pp. 435–446. IEEE, 2019a.

Mohsen Imani, Justin Morris, Samuel Bosch, Helen Shu, Giovanni De Micheli, and Tajana Rosing. Adapthd: Adaptive efficient training for brain-inspired hyperdimensional computing. In *2019 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pp. 1–4. IEEE, 2019b.

Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S Yu. A survey on knowledge graphs: Representation, acquisition, and applications. *IEEE transactions on neural networks and learning systems*, 33(2):494–514, 2021.

Jinhao Jiang, Kun Zhou, Wayne Xin Zhao, and Ji-Rong Wen. Unikgqa: Unified retrieval and reasoning for solving multi-hop question answering over knowledge graph. *arXiv preprint arXiv:2212.00959*, 2022.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Wayne Xin Zhao, and Ji-Rong Wen. Structgpt: A general framework for large language model to reason over structured data. *arXiv preprint arXiv:2305.09645*, 2023.

Jinhao Jiang, Kun Zhou, Wayne Xin Zhao, Yang Song, Chen Zhu, Hengshu Zhu, and Ji-Rong Wen. Kg-agent: An efficient autonomous agent framework for complex reasoning over knowledge graph. *arXiv preprint arXiv:2402.11163*, 2024.

Pentti Kanerva. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive Computation*, 1(2):139–159, 2009.

Pentti Kanerva et al. Fully distributed representation. *PAT*, 1(5):10000, 1997.

Omar Khattab, Keshav Santhanam, Xiang Lisa Li, David Hall, Percy Liang, Christopher Potts, and Matei Zaharia. Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive nlp. *arXiv preprint arXiv:2212.14024*, 2022.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp. In *NeurIPS*, 2020.

Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. Reasoning on graphs: Faithful and interpretable large language model reasoning. *arXiv preprint arXiv:2310.01061*, 2023.

Alexander H. Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents. *CoRR*, abs/1606.03126, 2016. URL http://arxiv.org/abs/1606.03126.

Ali Moin, Alex Zhou, Abbas Rahimi, Ankita Menon, Simone Benatti, George Alexandrov, Samuel Tamakloe, Joash Ting, Naoya Yamamoto, Yasser Khan, et al. A wearable biosensing system with in-sensor adaptive machine learning for hand gesture recognition. *Nature Electronics*, 4(1):54–63, 2021.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Tony A Plate. Holographic reduced representations. *IEEE Transactions on Neural Networks*, 6(3): 623–641, 1995.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Omer Levy. Measuring and narrowing the compositionality gap in language models. *arXiv:2210.03350*, 2022. Self-Ask.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

Abbas Rahimi, Pentti Kanerva, José del R Millán, and Jan M Rabaey. Hyperdimensional computing for noninvasive brain–computer interfaces: Blind and one-shot classification of eeg error-related potentials. In *10th EAI International Conference on Bio-Inspired Information and Communications Technologies*, pp. 19. European Alliance for Innovation (EAI), 2017.

Apoorv Saxena, Aditay Tripathi, and Partha Talukdar. Improving multi-hop question answering over knowledge graphs using knowledge base embeddings. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pp. 4498–4507, 2020.

Jiaxin Shi, Shulin Cao, Lei Hou, Juanzi Li, and Hanwang Zhang. Transfernet: An effective and transparent framework for multi-hop question answering over relation graph. *arXiv preprint arXiv:2104.07302*, 2021.

Yuan Sui, Yufei He, Nian Liu, Xiaoxin He, Kun Wang, and Bryan Hooi. Fidelis: Faithful reasoning in large language model for knowledge graph question answering. *arXiv preprint arXiv:2405.13873*, 2024.

Haitian Sun, Tania Bedrax-Weiss, and William W Cohen. Open domain question answering using early fusion of knowledge bases and text. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4231–4242, 2018.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel M Ni, Heung-Yeung Shum, and Jian Guo. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. *arXiv preprint arXiv:2307.07697*, 2023.

Alon Talmor and Jonathan Berant. The web as a knowledge-base for answering complex questions. *arXiv preprint arXiv:1803.06643*, 2018.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Yao Xu, Shizhu He, Jiabei Chen, Zihao Wang, Yangqiu Song, Hanghang Tong, Guang Liu, Kang Liu, and Jun Zhao. Generate-on-graph: Treat llm as both agent and kg in incomplete knowledge graph question answering. *arXiv preprint arXiv:2404.14741*, 2024.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*, 2018.

Shunyu Yao, Dian Yang, Run-Ze Cui, and Karthik Narasimhan. React: Synergizing reasoning and acting in language models. In *ICLR*, 2023.

Shunyu Yao, Dian Zhao, Luyu Yu, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *arXiv:2305.10601*, 2024.

Calvin Yeung, Zhuowen Zou, and Mohsen Imani. Generalized holographic reduced representations. *arXiv preprint arXiv:2405.09689*, 2024.

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

Jing Zhang, Xiaokang Zhang, Jifan Yu, Jian Tang, Jie Tang, Cuiping Li, and Hong Chen. Subgraph retrieval enhanced model for multi-hop knowledge base question answering. *arXiv preprint arXiv:2202.13296*, 2022.

Xiaohan Zou. A survey on application of knowledge graph. In *Journal of Physics: Conference Series*, volume 1487, pp. 012016. IOP Publishing, 2020.

A NOTATION

Table 5: Notation used throughout the paper.

Notation	Definition
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	Knowledge graph with entity set $\mathcal{V}$ and edge set $\mathcal{E}$.
$\mathcal{Z}$	Set of relation schemas/path templates.
q, a	Input question and (predicted) answer.
e, r	An entity and a relation (schema edge), respectively.
$z = (r_1, \ldots, r_\ell)$	A relation path; $\lvert z\rvert = \ell$ denotes path length.
$\mathcal{Z}_{\text{cand}}$	Candidate path set instantiated from $\mathcal{G}$.
$N = \lvert Z_{\text{cand}}\rvert$	The number of candidate paths instantiated from the KG for a given query.
$L_{\max}$, B, K	Max plan depth, BFS beam width, and number of retrieved paths kept after pruning.
d, D, m	Hypervector dimension, # of GHRR blocks, and block size (unitary $m \times m$); flattened $d = Dm^2$.
$\mathbf{v}_x$	Hypervector for symbol x (entity/relation/path).
$\mathbf{v}_q$, $\mathbf{v}_z$	Query-plan hypervector and a candidate-path hypervector.
$\mathbf{H} = [A_1; \ldots; A_D]$	A GHRR hypervector with $Q\Lambda$ unitary blocks $A_j \in \mathrm{U}(m)$.
A^*	Conjugate transpose (unitary inverse) of a block A.
$\circledast$	GHRR *blockwise binding* operator (matrix product per block).
$\langle A, B\rangle_F$	Frobenius inner product $\mathrm{tr}(A^*B)$; $\lVert A\rVert_F$ is the Frobenius norm.
$\mathrm{sim}(\cdot, \cdot)$	Blockwise cosine similarity used for HD retrieval.
$s(z)$	Calibrated retrieval score; α, β, λ are calibration hyperparameters; $\mathrm{IDF}(z)$ is an inverse-frequency weight.
$\mathcal{M}$, M	Distractor set and its size $M = \lvert\mathcal{M}\rvert$ (used in capacity bounds).
ϵ, δ	Tolerance and failure probability in the concentration/union bounds.
c	Absolute constant in the sub-Gaussian tail bound.

B ALGORITHM

Algorithm 1 HD-RETRIEVE: Hyperdimensional Top-K Path Retrieval

Require: question q; KG $\mathcal{G}$; relation schemas $\mathcal{Z}$; max depth $L_{\max}$; beam width B; calibration (α, β, λ); Top-K
Ensure: Top-K reasoning paths $\mathcal{P}_K$ and their scores

Plan (schema-level): Construct a relation-schema graph over $\mathcal{Z}$ and run constrained BFS up to depth $L_{\max}$ with beam width B to obtain a small set of type-consistent relation plans $\mathcal{Z}_q \subseteq \mathcal{Z}$ for q.
Encode Query: Pick a plan $z_q \in \mathcal{Z}_q$ and encode it by GHRR binding:
  $\mathbf{v}_q \leftarrow \mathrm{BindPath}(z_q) = \circledast_{r \in z_q} \mathbf{v}_r$
  $\widetilde{\mathbf{v}}_q \leftarrow \mathcal{N}_{\text{block}}(\mathbf{v}_q)$ for scoring. {binding blocks remain unitary}
Instantiate Candidates (entity-level): $\mathcal{P}(q) \leftarrow \emptyset$
for $z \in \mathcal{Z}_q$ **do**
  Instantiate concrete KG paths consistent with schema z by matching its relation pattern to edges in $\mathcal{G}$ or by a constrained BFS on $\mathcal{G}$ (depth $\leq L_{\max}$, beam width B).
  Add all instantiated paths to $\mathcal{P}(q)$.
end for
Deduplicate paths in $\mathcal{P}(q)$ and enforce type consistency.
for $p \in \mathcal{P}(q)$ **do**
  Let $z(p) = (r_1, \ldots, r_\ell)$ be the relation sequence of path p.
  Encode Candidate: $\mathbf{v}_p \leftarrow \mathrm{BindPath}(z(p)) = \circledast_{r \in z(p)} \mathbf{v}_r$
  $\widetilde{\mathbf{v}}_p \leftarrow \mathcal{N}_{\text{block}}(\mathbf{v}_p)$ for scoring.
  Score (blockwise cosine): $s_{\cos}(p) \leftarrow \mathrm{sim}(\widetilde{\mathbf{v}}_q, \widetilde{\mathbf{v}}_p)$ {Eq. (5)}
  Calibrate (optional): $s(p) \leftarrow s_{\cos}(p) + \alpha\, \mathrm{IDF}(\mathrm{schema}(z(p))) - \beta\, \lambda^{|z(p)|}$
end for
return Top-K paths in $\mathcal{P}(q)$ ranked by $s(p)$ as $\mathcal{P}_K$.
{All steps above are symbolic; no additional LLM calls beyond the final reasoning step.}

C PROMPT TEMPLATE FOR ONE-SHOT REASONING

Table 6: Prompt template for KG path–grounded QA.

System	*You are a careful reasoner. Only use the provided KG reasoning paths as evidence. Cite the most relevant path(s) and answer concisely.*
User	**Question:** "$QUESTION" **Retrieved paths (Top-K):** 1. $PATH_1 2. $PATH_2 3. ... 4. $PATH_K
Assistant (required format)	**Answer:** $SHORT_ANSWER **Supporting path(s):** [indexes from the list above] **Rationale (1–2 sentences):** why those paths imply the answer.

D ADDITIONAL THEORETICAL SUPPORT

D.1 NEAR-ORTHOGONALITY AND CAPACITY IN A TOY RADEMACHER MODEL

To build intuition for why high-dimensional hypervectors are effective for retrieval, we first consider a simplified, classical VSA setting: real Rademacher hypervectors with element-wise binding. Our actual method uses complex $Q\Lambda$ GHRR blocks with blockwise binding (Section 2.2, Proposition 1). The Rademacher analysis below is an intuition-building surrogate; transferring its bound to the exact $Q\Lambda$ construction requires distributional assumptions on the non-diagonal Q factors that are not specified here.

We justify the use of high-dimensional hypervectors in `PathHD` by showing that (i) random hypervectors are nearly orthogonal with high probability, and (ii) this property is preserved under binding, yielding exponential concentration that enables accurate retrieval at scale.

Setup. Let each entity/relation be encoded as a Rademacher hypervector $\mathbf{x} \in \{-1, +1\}^d$ with i.i.d. entries. For two independent hypervectors $\mathbf{x}, \mathbf{y}$, define cosine similarity $\cos(\mathbf{x}, \mathbf{y}) = \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\| \, \|\mathbf{y}\|}$. Since $\|\mathbf{x}\| = \|\mathbf{y}\| = \sqrt{d}$, we have $\cos(\mathbf{x}, \mathbf{y}) = \frac{1}{d} \sum_{k=1}^{d} x_k y_k$.

Proposition 2 (Near-orthogonality of random hypervectors)**.** *For any $\epsilon \in (0, 1)$,*

$$\Pr\big(|\cos(\mathbf{x}, \mathbf{y})| > \epsilon\big) \;\le\; 2\exp\big(-\tfrac{1}{2}\epsilon^2 d\big)\,.$$

Proof. Each product $Z_k = x_k y_k$ is i.i.d. Rademacher with $\mathbb{E}[Z_k] = 0$ and $|Z_k| \le 1$. By Hoeffding's inequality, $\Pr\Big(\Big|\sum_{k=1}^{d} Z_k\Big| > \epsilon d\Big) \le 2\exp(-\epsilon^2 d/2)$. Divide both sides by d to obtain the claim. □

Lemma 1 (Closure under binding)**.** *Let $\mathbf{r}_1, \ldots, \mathbf{r}_n$ be independent Rademacher hypervectors and define binding (element-wise product) $\mathbf{p} = \mathbf{r}_1 \odot \cdots \odot \mathbf{r}_n$. Then $\mathbf{p}$ is also a Rademacher hypervector. Moreover, if $\mathbf{s}$ is independent of at least one $\mathbf{r}_i$ used in $\mathbf{p}$, then $\mathbf{p}$ and $\mathbf{s}$ behave as independent Rademacher hypervectors and $\mathbb{E}[\cos(\mathbf{p}, \mathbf{s})] = 0$.*

Proof. Each coordinate $p_k = \prod_{i=1}^{n} r_{i,k}$ is a product of independent Rademacher variables, hence Rademacher. If $\mathbf{s}$ is independent of some r_j, then $p_k s_k$ has zero mean and remains bounded, so the same Hoeffding-type bound as in Proposition 2 applies. □

Theorem 1 (Separation and error bound for hypervector retrieval)**.** *Let the query hypervector be $\mathbf{q} = \mathbf{r}_1 \odot \cdots \odot \mathbf{r}_n$ and consider a candidate set containing the true path $\mathbf{p}^\star = \mathbf{q}$ and M distractors*

$\{\mathbf{p}_i\}_{i=1}^{M}$, *where each distractor differs from* $\mathbf{q}$ *in at least one relation (thus satisfies Lemma 1). Then for any* $\epsilon \in (0,1)$ *and* $\delta \in (0,1)$*, if*

$$d \;\geq\; \frac{2}{\epsilon^2}\,\log\!\Big(\frac{2M}{\delta}\Big),$$

we have, with probability at least $1-\delta$*,*

$$\cos(\mathbf{q},\mathbf{p}^\star)=1 \quad \textit{and} \quad \max_{1\leq i\leq M}\big|\cos(\mathbf{q},\mathbf{p}_i)\big| \leq \epsilon.$$

Proof. By construction, $\mathbf{p}^\star = \mathbf{q}$, hence cosine $= 1$. For each distractor $\mathbf{p}_i$, Lemma 1 implies that $\mathbf{q}$ and $\mathbf{p}_i$ behave as independent Rademacher hypervectors; applying Proposition 2, $\Pr(|\cos(\mathbf{q},\mathbf{p}_i)|>\epsilon) \leq 2e^{-\epsilon^2 d/2}$. A union bound over M distractors yields $\Pr(\max_i |\cos(\mathbf{q},\mathbf{p}_i)| > \epsilon) \leq 2Me^{-\epsilon^2 d/2} \leq \delta$ under the stated condition on d. □

E Additional Proofs and Tail Bounds

Details for Prop. 1. Here, we sketch the argument for the GHRR near-orthogonality bound used in the main text. For this idealized bound, we assume that the normalized similarity can be written as $X = \frac{1}{d}\sum_{j=1}^{d}\xi_j$, where the effective contributions ξ_j are independent, mean zero, and have bounded ψ_2-norm. This is an analytical assumption, not a claim that entries of the implemented $Q\Lambda$ unitary blocks are independent. Applying Hoeffding/Bernstein, $\Pr(|X| \geq \epsilon) \leq 2\exp(-cd\epsilon^2)$ for some absolute constant $c>0$, which yields the stated idealized result. This argument does not prove the same bound for the implemented $Q\Lambda$ construction without further distributional assumptions on Q. Separately, products of the unnormalized unitary binding blocks remain unitary under binding depth; see also Plate (1995); Kanerva (2009) for stability of holographic codes. □

F Optional Prompt-Based Schema Refinement

As described in Section 2.4, all results in Section 3 use only schema-based enumeration of relation schemas, without any additional LLM calls beyond the final reasoning step. For completeness, we describe here an optional extension that refines schema plans with a lightweight prompt.

Given a small set of candidate relation schemas $\{z^{(1)},\ldots,z^{(M)}\}$ obtained from enumeration, we first verbalize each schema into a short natural-language description (e.g., by mapping each relation type r to a phrase and concatenating them). We then issue a single prompt of the form:

> Given the question q and the following candidate relation patterns: (1) `[schema 1]`, (2) `[schema 2]`, ..., which K patterns are most relevant for answering q? Please output only the indices.

The LLM outputs a small subset of indices, which we use to select the top-K schemas $\{z^{(i_1)},\ldots,z^{(i_K)}\}$. These refined schemas are then instantiated into concrete KG paths and encoded into hypervectors exactly as in the main method.

We emphasize that this refinement is *not* used in any of our reported experiments, and that it can be implemented with at most one additional short LLM call per query. The main system studied in this paper, therefore, remains a single-call KG-LLM pipeline in all empirical results.

G Dataset Introduction

We provide detailed descriptions of the three benchmark datasets used in our experiments. All datasets and processed KGQA artifacts used in this work are publicly released by prior studies. We credit the corresponding original papers and release pages, and follow their original licenses and terms of use. We do not construct a new Freebase snapshot or rerun a separate subgraph-extraction pipeline; instead, our experiments build on released Freebase-based KGQA artifacts to ensure comparability with prior KG-LLM systems.

Table 7: Statistics of Freebase-based KGQA datasets used in our experiments.

Dataset	Train	Dev	Test	Typical hops	KG
WebQSP (Yih et al., 2016)	3,098	–	1,639	1–2	Freebase
CWQ (Talmor & Berant, 2018)	27,734	3,480	3,475	2–4	Freebase
GrailQA (Gu et al., 2021)	44,337	6,763	13,231	1–4	Freebase

WebQuestionsSP (WebQSP) (Yih et al., 2016) consists of 4,737 questions, where each question is manually annotated with a topic entity and a SPARQL query over Freebase. The answer entities are within a maximum of 2 hops from the topic entity. Following prior work (Sun et al., 2018; Luo et al., 2023), we use the standard train/validation/test splits and the released processed RoG artifact for WebQSP.[1] This ensures that our evaluation follows the same Freebase-based protocol as prior work.

Complex WebQuestions–SP (CWQ-SP) (Talmor & Berant, 2018) is the Freebase/SPARQL-annotated variant of CWQ, aligning each question to a topic entity and an executable SPARQL query over a cleaned Freebase subgraph. Questions are created by compositional expansions of WebQSP, including constraints, joins, and longer paths, and typically require up to 4-hop reasoning. Following prior work (Luo et al., 2023), we use the standard split and the released processed RoG artifact for CWQ.[2]

GrailQA (Gu et al., 2021) is a large-scale KGQA benchmark with 64,331 questions. It evaluates generalization in multi-hop reasoning across three settings: i.i.d., compositional, and zero-shot. Each question is annotated with a corresponding logical form and answer, and the underlying KG is a cleaned subset of Freebase. Following prior KG-LLM work, we use the official split and the released processed artifact from ToG.[3] In our experiments, we evaluate on the official **dev** set. The dev set is the authors' held-out split from the same cleaned Freebase graph and mirrors the three generalization settings; it is commonly used for ablations and model selection when the test labels are held out.

We follow the unified Freebase protocol (Bollacker et al., 2008), which contains approximately 88 million entities, 20 thousand relations, and 126 million triples. We evaluate with the official *Hits@1*/*F1* scripts. For **GrailQA**, numbers in the main results are reported on the **dev** split and additionally on its *IID* subset; many recent works adopt dev evaluation due to test-server restrictions. WebQSP has no official dev split under this setting. Additional statistics, including the number of reasoning hops and answer entities, are shown in Table 7.

H Detailed Baseline Descriptions

We categorize the baseline methods into four groups and describe each group below.

H.1 Embedding-based methods

- **KV-Mem** (Miller et al., 2016) uses a key-value memory architecture to store knowledge triples and performs multi-hop reasoning through iterative memory operations.
- **EmbedKGQA** (Saxena et al., 2020) formulates KGQA as an entity-linking task and ranks entity embeddings using a question encoder. **NSM** (He et al., 2021) adopts a sequential program execution framework over KG relations, learning to construct and execute reasoning paths.
- **TransferNet** (Shi et al., 2021) builds on GraftNet by incorporating both relational and text-based features, enabling interpretable step-wise reasoning over entity graphs.

H.2 Retrieval-augmented methods

- **GraftNet** (Sun et al., 2018) retrieves question-relevant subgraphs and applies GNNs for reasoning over linked entities.
- **SR+NSM** (Zhang et al., 2022) retrieves relation-constrained subgraphs and runs NSM over them to generate answers.

[1] https://huggingface.co/datasets/rmanluo/RoG-webqsp
[2] https://huggingface.co/datasets/rmanluo/RoG-cwq
[3] https://github.com/DataArcTech/ToG/blob/main/data/WebQSP.json

- **SR+NSM+E2E** (Zhang et al., 2022) further optimizes SR+NSM via end-to-end training of the retrieval and reasoning modules.
- **UniKGQA** (Jiang et al., 2022) unifies entity retrieval and graph reasoning into a single LLM-in-the-loop architecture, achieving strong performance with reduced pipeline complexity.

H.3 Pure LLMs

- **ChatGPT** (Ouyang et al., 2022), **Davinci-003** (Ouyang et al., 2022), and **GPT-4** (Achiam et al., 2023) serve as closed-book baselines using few-shot or zero-shot prompting.
- **StructGPT** (Jiang et al., 2023) generates structured reasoning paths in natural language form, then executes them step by step.
- **ROG** (Luo et al., 2023) reasons over graph-based paths with alignment to LLM beliefs.
- **Think-on-Graph** (Sun et al., 2023) prompts the LLM to search symbolic reasoning paths over a KG and use them for multi-step inference.

H.4 LLMs + KG methods

- **GoG** (Xu et al., 2024) adopts a plan-then-retrieve paradigm, where an LLM generates reasoning plans and a KG subgraph is retrieved accordingly.
- **KG-Agent** (Jiang et al., 2024) turns the KGQA task into an agent-style decision process using graph environment feedback.
- **FiDeLiS** (Sui et al., 2024) fuses symbolic subgraph paths with LLM-generated evidence, filtering hallucinated reasoning chains.
- `PathHD` (ours) proposes a vector-symbolic integration pipeline where top-K relation paths are selected by vector matching and adjudicated by an LLM, combining symbolic controllability with neural flexibility.

I Detailed Experimental Setups

We follow a unified evaluation protocol: all methods are evaluated on the Freebase KG with the official *Hits@1/F1* scripts for WebQSP, CWQ, and GrailQA (dev, IID), so that the numbers are directly comparable across systems. We did not construct a new Freebase snapshot or rerun a separate subgraph extraction pipeline. For WebQSP and CWQ, we directly use the released processed RoG artifacts,[4] [5] and for GrailQA, we use the released processed artifact from ToG.[6] Thus, our contribution is not a new graph preprocessing pipeline, but a new retrieval-and-reasoning framework built on top of these released KGQA artifacts.

Whenever possible, we adopt the official results reported by prior work under the same setting. Concretely, we take numbers for KV-Mem (Miller et al., 2016), GraftNet (Sun et al., 2018), EmbedKGQA (Saxena et al., 2020), NSM (He et al., 2021), TransferNet (Shi et al., 2021), SR+NSM and its end-to-end variant (SR+NSM+E2E) (Zhang et al., 2022), UniKGQA (Jiang et al., 2022), RoG (Luo et al., 2023), StructGPT (Jiang et al., 2023), Think-on-Graph (Sun et al., 2023), GoG (Xu et al., 2024), and FiDeLiS (Sui et al., 2024) from their papers or consolidated tables under equivalent Freebase protocols. We further include a pure-LLM category (ChatGPT, Davinci-003, GPT-4) using the unified results reported by KG-Agent (Jiang et al., 2024); note that its GrailQA scores are on the dev split. For KG-Agent itself, we use the official numbers reported in its paper (Jiang et al., 2024). All PathHD experiments are conducted on a single NVIDIA RTX 4090 GPU with 24GB memory. Latency and memory measurements use the same hardware environment unless otherwise stated.

For `PathHD`, all LLMs and sentence encoders are used *off the shelf* with no fine-tuning. The block size m of the $Q\Lambda$ unitary blocks is chosen from a small range motivated by the VSA literature; following GHRR, we fix a small block size $m = 4$ and primarily tune the overall dimensionality d. Prior work on GHRR (Yeung et al., 2024) shows that, for a fixed total dimension d, moderate changes in m trade off non-commutativity and saturation behaviour, but do not lead to instability. In

[4] https://huggingface.co/datasets/rmanluo/RoG-webqsp
[5] https://huggingface.co/datasets/rmanluo/RoG-cwq
[6] https://github.com/DataArcTech/ToG/blob/main/data/WebQSP.json

our experiments, we therefore treat d as the main tuning parameter (set on the dev split), while fixing m across all runs. Additional hyperparameters for candidate enumeration and calibration are detailed in Sections I.1 and I.2.

I.1 Additional Analytic Efficiency

Table 8: Full analytical comparison (no implementation). Ranges reflect algorithm design; d and b denote planning depth and beam/fanout as specified in RoG, which uses beam-search with $B = 3$ and path length bounded by dataset hops (WebQSP≤ 2, CWQ≤ 4).

Method	# LLM calls / query	Planning depth	Retrieval fanout/beam	Executor/Tools
KV-Mem (Miller et al., 2016)	0	multi-hop (learned)	moderate	Yes (neural mem)
EmbedKGQA (Saxena et al., 2020)	0	multi-hop (seq)	moderate	No
NSM (He et al., 2021)	0	multi-hop (neural)	moderate	Yes (neural executor)
TransferNet (Shi et al., 2021)	0	multi-hop	moderate	No
GraftNet (Sun et al., 2018)	0	multi-hop	graph fanout	No
SR+NSM (Zhang et al., 2022)	0	multi-hop	subgraph (beam)	Yes (neural exec)
SR+NSM+E2E (Zhang et al., 2022)	0	multi-hop	subgraph (beam)	Yes (end-to-end)
ChatGPT (Ouyang et al., 2022)	1	0	n/a	No
Davinci-003 (Ouyang et al., 2022)	1	0	n/a	No
GPT-4 (Achiam et al., 2023)	1	0	n/a	No
UniKGQA (Jiang et al., 2022)	1–2	shallow	small/merged	No (unified model)
StructGPT (Jiang et al., 2023)	1–2	1	n/a	Yes (tool use)
RoG (Luo et al., 2023)	$\approx d \times b$	d	b (per step)	No (LLM scoring)
Think-on-Graph (Sun et al., 2023)	3–6	multi	small/beam	Yes (plan & react)
GoG (Xu et al., 2024)	3–5	multi	small/iterative	Yes (generate-retrieve loop)
KG-Agent (Jiang et al., 2024)	3–8	multi	small	Yes (agent loop)
FiDeLiS (Sui et al., 2024)	1–3	shallow	small	Optional (verifier)
`PathHD` **(ours)**	**1** (final only)	**0**	vector ops only	No (vector ops)

I.2 More Hyperparameter Tuning Details

We tune the calibration weights (α, β, λ) used in our scoring/normalization module by grid search on the validation split. Specifically, we sweep $\alpha, \beta \in \{0, 0.1, 0.2, \ldots, 0.5\}$ and $\lambda \in \{0.6, 0.7, 0.8, 0.9\}$ and select the triple that maximizes validation Hits@1 for each dataset. The chosen settings are summarized in Table Table 9: WebQSP $(0.2, 0.1, 0.8)$, CWQ $(0.3, 0.1, 0.8)$, and GrailQA $(0.2, 0.2, 0.8)$.

Table 9: Calibration hyperparameters (α, β, λ) used for each dataset.

Dataset	α	β	λ
WebQSP	0.2	0.1	0.8
CWQ	0.3	0.1	0.8
GrailQA	0.2	0.2	0.8

Across datasets, we observe from Table 9 that a consistent preference for λ=0.8, indicating that a moderate calibration strength provides the best balance between raw similarity and normalization. The smaller values of β (0.1–0.2) suggest only light regularization is needed, while the data-dependent shift in α (0.2 on WebQSP/GrailQA vs. 0.3 on CWQ) implies CWQ benefits from slightly higher emphasis on the calibrated score, likely due to its noisier multi-hop structure. Overall, performance is robust around the reported optima, with several neighboring settings yielding very similar accuracy, which simplifies tuning in practice.

J Additional Experiments

J.1 Additional Ablatiion Studies

J.1.1 Do we need a final single LLM adjudicator?

We test whether a lightweight LLM judgment helps beyond pure vector scoring. *Vector-only* selects the top path by cosine similarity; *Vector* $\rightarrow$ *1*$\times$*LLM* instead forwards the pruned top-K paths (with scores and end entities) to a single LLM using a short fixed template (no tools/planning) to choose the answer *without* long chains of thought. As shown in Table 10, **Vector** $\rightarrow$ **1**$\times$**LLM** consistently outperforms *Vector-only* on both datasets, especially when the top two paths are near-tied or a high-scoring path has a subtle type mismatch; a single adjudication pass resolves such cases at negligible extra cost.

Table 10: **Ablation on the final decision maker**. Passing pruned candidates and scores to a single LLM for adjudication yields consistent gains over vector-only selection.

Final step	WebQSP	CWQ
Vector-only	85.4	70.8
Vector → 1×LLM	**86.2**	**71.5**

J.1.2 What is the effect of top-K pruning before the final step?

Finally, we study how many candidates should be kept for the last decision. We vary the number of paths passed to the final LLM among $K \in \{2, 3, 5\}$ and also include a *No-prune* variant that sends all retrieved paths. Retrieval and scoring are fixed; latency is the median per query (lower is better).

Table 11: **Impact of top-K pruning before the final LLM.** Small sets (K=2–3) retain or slightly improve accuracy while reducing latency. We adopt K=3 by default.

Pruning	Hits@1 (WebQSP)	Lat.	Hits@1 (CWQ)	Lat.
No-prune	85.8	2.42s	70.7	2.45s
K=2	86.0	1.98s	71.2	2.00s
K=3	**86.2**	**1.92s**	**71.5**	**1.94s**
K=5	86.1	2.05s	71.4	2.06s

As shown in Table 11, K=3 achieves the best Hits@1 on both WebQSP and CWQ with the lowest latency, while K=2 is a close second and yields the largest latency drop. In contrast, *No-prune* maintains maximal recall but increases latency and often introduces near-duplicate/noisy paths that can blur the final decision. We therefore adopt K=3 as the default.

J.1.3 Which similarity metric best scores candidate paths?

We ablate the similarity function used to score candidate relation paths before the final decision. Keeping all other components fixed (GHRR binding, identical candidate set, top-K pruning, and a single LLM adjudication), we replace the default *blockwise cosine* with four variants: *global cosine* (cosine on the full hypervector), *dot product*, *negative L_2 distance*, and *max–block cosine* (maximum cosine over blocks). For each query, all variants score the same candidates, apply the same pruning, and feed the pruned paths plus scores to the one-shot LLM. We report F1 (%) on WebQSP, CWQ, and GrailQA-dev under the unified Freebase evaluation.

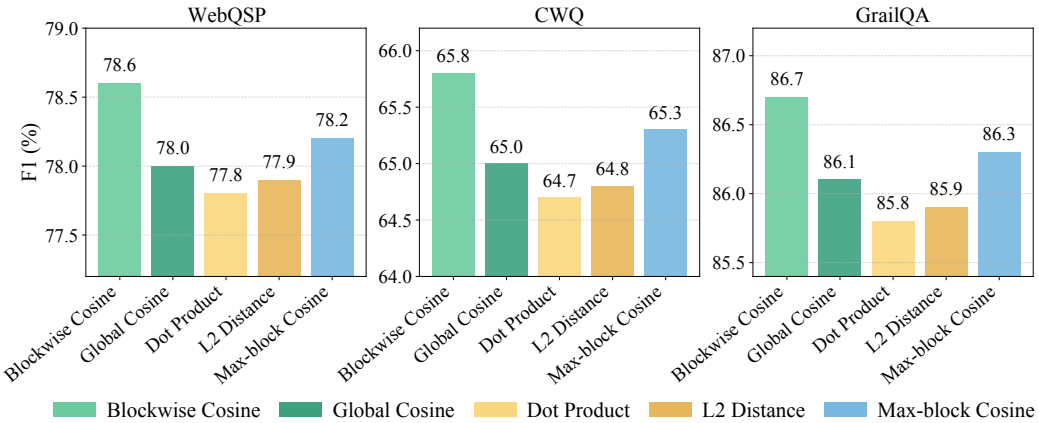


Figure 4: **Scoring measurement ablation.** We evaluate F1 (%) on WebQSP, CWQ, and GrailQA using different scoring strategies in our model. `PathHD` achieves the best or competitive results when using blockwise cosine similarity, highlighting its effectiveness in capturing fine-grained matching signals across vector blocks.

From the results in Figure 4, blockwise cosine is consistently strongest across datasets: 78.6 on WebQSP, 65.8 on CWQ, and 86.7 on GrailQA-dev, which beats the best alternative by roughly $0.3\sim0.6$ F1. *Max–block cosine* is the most competitive runner-up (e.g., $78.2/65.3/86.3$), suggesting that aligning a few informative blocks already captures useful local evidence. *Global cosine* trails by about $0.5\sim0.7$ F1, indicating that pooling over the entire hypervector can wash out position-specific signals created by GHRR's non-commutative composition. *Dot product* and L_2 behave similarly and are slightly worse overall, consistent with their sensitivity to vector scale and variance. Taken together, these results support block-structured scoring when paths are encoded via block-diagonal unitary binding.

J.1.4 PLAN-BASED DEFAULT VS. TEXT-PROJECTION VARIANT

In the default `PathHD` pipeline, both the selected *query plan* and each *candidate path* are encoded structurally with GHRR binding. We compare this configuration with the reported *text-projection variant* while preserving its original scores.

Table 12: Comparison of **encoding variants** in `PathHD`. *Plan-based (GHRR)*, the default, structurally encodes both the selected query plan and candidate paths; the alternative is retained under the neutral label *Text-projection variant*. Metrics are Hits@1/F1 under the unified protocol; bold is best.

Encoding Variant	WebQSP	CWQ
Text-projection variant	83.4	69.8
Plan-based GHRR (default)	**86.2**	**71.5**

As shown in Table 12, across WebQSP and CWQ, the plan-based GHRR default consistently outperforms the text-projection variant (e.g., 86.2 vs. 83.4 on WebQSP; 71.5 vs. 69.8 on CWQ). The result favors structural GHRR encoding of both the selected query plan and candidate paths, but we do not attribute the gap to a specific encoder substitution without confirmation of the variant's exact configuration.

J.2 HYPERVECTOR DIMENSION STUDY

We study how the hypervector dimension D influences `PathHD`. Keeping all other components fixed (GHRR binding, blockwise–cosine scoring, top-K pruning with $K=20$, and a single final LLM adjudication), we sweep $D \in \{512, 1024, 2048, 3072, 4096, 6144, 8192\}$. For each dataset (WebQSP, CWQ, GrailQA-dev), we run under the same Freebase protocol and report F1 (%). Relation and query hypervectors are randomly initialized and kept fixed across D to isolate the effect of dimensionality.

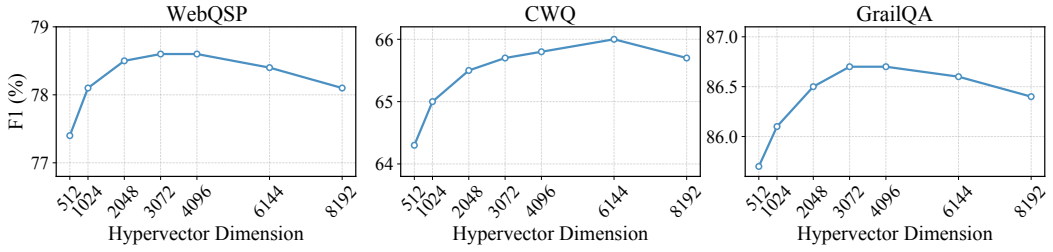


Figure 5: **Hypervector dimension study.** Each panel reports F1 (%) of `PathHD` on WebQSP, CWQ, and GrailQA as a function of the hypervector dimension. Overall, performance rises from 512 to the mid-range and then tapers off: WebQSP and GrailQA peak around 3k–4k, while CWQ prefers a slightly larger size (6k), after which F1 decreases mildly.

From the results in Figure 5, performance rises from $D=512$ to a mid–high range and then tapers off. WebQSP peaks around $D=3\text{k}-4\text{k}$ (about $78.6-78.8$ F1) and declines mildly by ≈ 0.5 points at 8k. GrailQA shows a similar shape, peaking near $3\text{k}-4\text{k}$ ($\approx 86.7-86.8$) with a small drop at 8k. CWQ benefits from a slightly larger space, peaking near 6k (≈ 66.0) and then decreasing modestly. The upward trend at small D reflects improved separability and reduced collisions as the representational capacity grows; the late decline likely comes from diluted blockwise-cosine margins and extra noise

in very high dimensions (fewer candidates per dimension, weaker concentration), which lessens the advantage of the final LLM tie–breaking. CWQ's preference for larger D is consistent with its broader relation vocabulary and longer reasoning chains, whereas the more constrained WebQSP/GrailQA saturate earlier.

J.3 Additional Visualization

We compare per-query latency on **CWQ** using the lollipop chart. To make methods comparable, latencies are *estimated under a unified setup*: each LLM call has a median of ≈2.2 s and p90 of ≈3.4 s; non-LLM vector/graph operations add 0.3–0.8 s. RoG is instantiated with beam width B=3 and depth bounded by dataset hops ($D \leq 4$, thus ≈12 LLM calls). `PathHD` performs blockwise-cosine scoring in HDC space plus *top-K pruning* (moderate setting) and issues a *single* LLM adjudication call; all other details follow the main protocol.

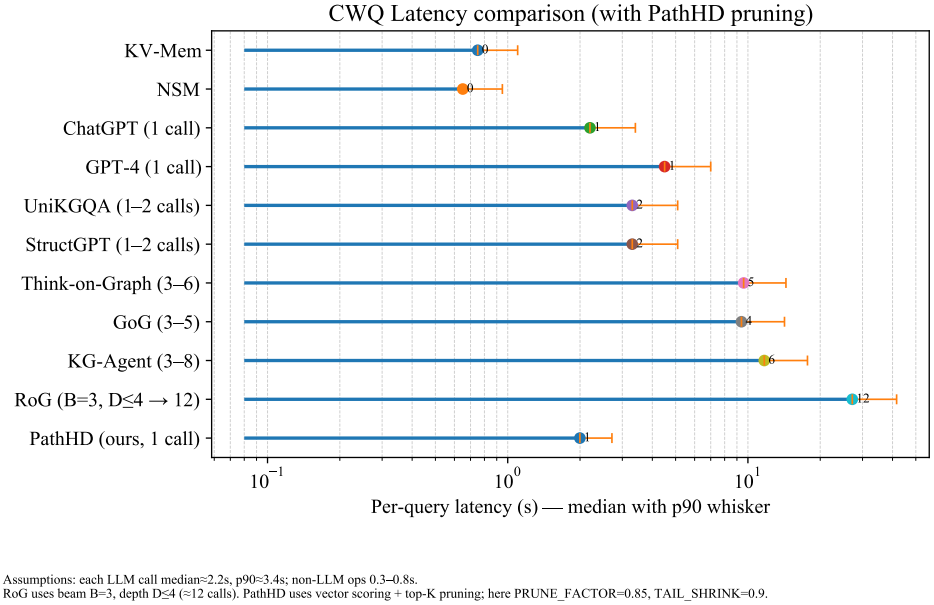


Figure 6: CWQ latency comparison (lollipop). Dots indicate median per-query latency; right whiskers show the 90th percentile (p90). The x-axis is log-scaled. Values are *estimated* under a unified setup: per-LLM-call median ≈ 2.2 s and p90 ≈ 3.4 s; non-LLM operations add 0.3-0.8 s. RoG follows beam width B=3 with depth bounded by dataset hops ($D \leq 4$, ≈12 calls), whereas `PathHD` uses a single LLM call plus vector operations for scoring.

From Figure 6, we can observe that `PathHD` achieves *near single-call latency* (median ≈2.3 s with a short p90 whisker), close to pure one-call LLMs (ChatGPT/GPT-4) and substantially faster than multi-call LLMs+KG agents: StructGPT/UniKGQA (1–2 calls) sit around ∼3.3 s, while Think-on-Graph/GoG/KG-Agent (3–8 calls) are in the ∼9–12 s range, and RoG (beam-search, ∼12 calls) reaches ∼27 s. The shorter tail for `PathHD` reflects two design choices: (i) most computation is handled by vector operations before the single adjudication call, and (ii) moderate top-K pruning trims low-quality candidates, reducing both the number and variability of expensive LLM interactions without harming accuracy.

J.4 Effect of Backbone Models

Performance across different LLM backbones is shown in Table 13.

Table 13: Performance across different LLM backbones. Each block fixes the backbone and varies the reasoning framework: a pure LLM control (CoT), our single-call `PathHD`, and 1-2 multi-step LLM+KG baselines. Metrics follow the unified Freebase setup.

Backbone	**Method**	**WebQSP** Hits@1 / F1	**CWQ** Hits@1 / F1	**GrailQA (F1)** Overall / IID	**#Calls** (/query)
GPT-4 (API)	CoT (Wei et al., 2022)	73.2 / 62.3	55.6 / 49.9	31.7 / 25.0	1
	RoG (Luo et al., 2023)	85.7 / 70.8	62.6 / 56.2	– / –	≈ 12
	KG-Agent (Jiang et al., 2024)	83.3 / **81.0**	72.2 / **69.8**	86.1 / 92.0	3–8
	`PathHD` (single-call)	**86.2** / 78.6	**71.5** / 65.8	**86.7** / **92.4**	**1**
GPT-3.5 / ChatGPT	CoT (Wei et al., 2022)	67.4 / 59.3	47.5 / 43.2	25.3 / 19.6	1
	StructGPT (Jiang et al., 2023)	72.6 / 63.7	54.3 / 49.6	54.6 / 70.4	1–2
	RoG (Luo et al., 2023)	85.0 / 70.2	61.8 / 55.5	– / –	≈ 12
	`PathHD` (single-call)	85.6 / 78.0	70.8 / 65.1	85.9 / 91.7	**1**
Llama-3-8B-Instruct (open)	CoT (prompt-only)	62.0 / 55.0	43.0 / 40.0	20.0 / 16.0	1
	ReAct-Lite (retrieval+CoT)	70.5 / 62.0	52.0 / 47.5	48.0 / 62.0	3–5
	BM25+LLM-Verifier (1×)	74.5 / 66.0	55.0 / 50.0	52.0 / 66.0	1
	`PathHD` (single-call)	84.8 / 77.2	69.8 / 64.2	84.9 / 90.9	**1**

J.5 Prompt Sensitivity of the LLM Adjudicator

Since `PathHD` relies on a single LLM call to adjudicate among the Top-K candidate paths, it is natural to ask how sensitive the system is to the exact phrasing of this adjudication prompt. To investigate this, we compare our default adjudication prompt (Prompt A) with a slightly rephrased variant (Prompt B) that uses different wording but conveys the same task description.[7]

Table 14: Prompt sensitivity of the LLM adjudicator. We compare the default adjudication prompt (Prompt A) with a paraphrased variant (Prompt B). Numbers are Hits@1 and F1 for WebQSP and CWQ, and Overall / IID F1 for GrailQA.

Prompt	WebQSP	CWQ	GrailQA (Overall / IID)
Prompt A (default)	**86.2 / 78.6**	**71.5 / 65.8**	**86.7 / 92.4**
Prompt B (paraphrased)	85.7 / 78.3	70.9 / 63.4	85.2 / 90.8

Table 14 reports *Hits@1* and *F1* on the three datasets under these two prompts. We observe that while minor prompt changes can occasionally flip individual predictions, the overall performance remains very close for all datasets, and the qualitative behavior of the path-grounded rationales is also stable. This suggests that, in our setting, `PathHD` is reasonably robust to small, natural variations in the adjudication prompt.

K Detailed Introduction of the Modules

K.1 Binding Operations

Below, we summarize the binding operators considered in our system and ablations. All bindings produce a composed hypervector $\mathbf{s}$ from two inputs $\mathbf{x}$ and $\mathbf{y}$ of the same dimensionality.

[7] For example, Prompt A asks the model to "select the most plausible reasoning path and answer the question based on it", whereas Prompt B paraphrases this as "choose the best supporting path and use it to answer the question".

(1) XOR / Bipolar Product (*commutative*). For binary hypervectors $\mathbf{x}, \mathbf{y} \in \{0,1\}^d$,

$$\mathbf{s} = \mathbf{x} \oplus \mathbf{y}, \qquad s_i = (x_i + y_i) \bmod 2.$$

Under the bipolar code $\{-1, +1\}$, XOR is equivalent to element-wise multiplication:

$$s_i = x_i \cdot y_i, \qquad x_i, y_i \in \{-1, +1\}.$$

This is the classical *commutative bind* baseline used in our ablation.

(2) Real-valued Element-wise Product (*commutative*). For real vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$,

$$\mathbf{s} = \mathbf{x} \odot \mathbf{y}, \qquad s_i = x_i\, y_i.$$

Unbinding is approximate by element-wise division (with small ϵ for stability): $x_i \approx s_i/(y_i + \epsilon)$. This is another variant of the *commutative bind*.

(3) HRR: Circular Convolution (*commutative*). For $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$,

$$\mathbf{s} = \mathbf{x} \circledast \mathbf{y}, \qquad s_k = \sum_{i=0}^{d-1} x_i\, y_{(k-i) \bmod d}.$$

Approximate unbinding uses circular correlation:

$$\mathbf{x} \approx \mathbf{s} \circledast^{-1} \mathbf{y}, \qquad x_i \approx \sum_{k=0}^{d-1} s_k\, y_{(k-i) \bmod d}.$$

This is the *Circ. conv* condition in our ablation.

(4) FHRR / Complex Phasor Product (*commutative*). Let $\mathbf{x}, \mathbf{y} \in \mathbb{C}^d$ with unit-modulus components $x_i = e^{i\phi_i}$, $y_i = e^{i\psi_i}$. Binding is element-wise complex multiplication

$$\mathbf{s} = \mathbf{x} \odot \mathbf{y}, \qquad s_i = x_i y_i = e^{i(\phi_i + \psi_i)},$$

and unbinding is conjugation: $\mathbf{x} \approx \mathbf{s} \odot \mathbf{y}^*$. FHRR is often used as a complex analogue of HRR.

(5) Block-diagonal GHRR (*non-commutative*, ours). We use Generalized HRR with block-unitary components. A hypervector is a block vector $\mathbf{H} = [A_1; \ldots; A_D]$, $A_j \in \mathrm{U}(m)$ (so total dimension $d = Dm^2$ when flattened). Each implemented block has the form $A_j = Q_j \Lambda_j$, where Q_j is a generally non-diagonal unitary matrix and Λ_j is a diagonal phase matrix. The diagonal phase factor alone is commutative; non-commutativity comes from the general unitary block structure supplied by Q_j. Given $\mathbf{X} = [X_1; \ldots; X_D]$ and $\mathbf{Y} = [Y_1; \ldots; Y_D]$, binding is the block-wise product

$$\mathbf{Z} = \mathbf{X} \circledast \mathbf{Y}, \qquad Z_j = X_j Y_j \;\; (j = 1, \ldots, D).$$

Since these generally non-diagonal unitary blocks need not commute ($X_j Y_j \neq Y_j X_j$), GHRR enables order-sensitive composition; edge direction is carried by the directed relation symbols. For the unnormalized binding blocks, unitarity gives the exact inverses

$$X_j = Z_j Y_j^*, \qquad Y_j = X_j^* Z_j.$$

Normalized copies are used only for cosine scoring. This **Block-diag (GHRR)** operator is our default choice and achieves the best performance in the operation study (Table 3), compared to *Comm. bind* and *Circ. conv*.

K.2 Additional Case Study

Table 16 presents *additional* WebQSP case studies for `PathHD`. Unlike the main paper's case table (Top-4 candidates with pruning to K=3), this appendix visualizes the **Top-3** highest-scoring relation paths for readability and then prunes to K=2 before a single-LLM adjudication.

Across the four examples (Cases 3-6), pruning to K=2 often retains the correct path and achieves strong final answers after LLM adjudication. However, we also observe a typical failure mode of the vector-only selector under K=2: when multiple plausible paths exist (e.g., country vs. continent, or

Table 15: **Case study 2 on multi-hop reasoning over WebQSP.** Top-K pruning is applied before invoking LLM, reducing cost while retaining plausible candidates.

Case 2 : *in which movies does Logan Lerman act in that was edited by Mary Jo Markey*	
Top-4 candidates	1) `film.film.edited_by` (0.2548) 2) `person.nationality` → `film.film.country` (0.2527) 3) `performance.actor` → `performance.film` (0.2505) 4) `award.award_winner.awards_won` → `award.award_honor.honored_for` (0.2420)

Top-K after pruning (K=3)	`film.film.edited_by`
	`person.nationality` → `film.film.country`
	`performance.actor` → `performance.film`

Vector-only (no LLM)	Pick `film.film.edited_by` ✗ identifies edited films, but lacks actor constraint; leads to false positives.
1×LLM adjudication	*Rationale:* "The question requires jointly filtering for actor and editor. While `film.edited_by` is relevant, combining it with `performance.actor` improves precision by ensuring Logan Lerman is in the cast."
Final Answer / GT	**Perks of Being a Wallflower (predict) / Perks of Being a Wallflower** ✓

actor vs. editor constraints), the vector-only choice can become brittle and select a high-scoring but *underconstrained* path, after which the LLM must recover the correct answer using the remaining candidate (see Case 4). In contrast, the main-paper setting with K=3 keeps one more candidate, which *more reliably preserves a constraint-satisfying path* (e.g., explicitly encoding actor or continent relations). This extra coverage reduces reliance on the LLM to repair mistakes and improves robustness under compositional queries.

While K=2 is cheaper and can work well in many instances, K=3 **offers a better coverage-precision trade-off** on average: it mitigates pruning errors in compositional cases and lowers the risk of discarding the key constraint path. This aligns with our main experimental choice of K=3, which we use for all reported metrics in the paper.

Case-study note. For the qualitative case studies only, we manually verified the final entity answers using publicly available sources (e.g., film credits and encyclopedia entries). This light-weight human verification was used *solely* to present readable examples; it does not affect any quantitative metric. All reported metrics (e.g., Hits@1 and F1) are computed from dataset-provided supervision and ground-truth paths without human annotation.

Table 16: Case studies for `PathHD` with an *illustrative* display of candidates. For each query, we list the three highest-scoring relation paths (Top-3) for readability, then prune to $K = 2$ before the vector-only choice and a single-LLM adjudication.

Case 3: *where are the gobi desert located on a map*	
Top-3 candidates	1) `location.location.containedby` (0.3410) 2) `location.location.partially_containedby` (0.3335) 3) `location.location.contains` (0.3255)
Top-K after pruning (K=2)	`containedby` `partially_containedby`
Vector-only (no LLM)	Pick `containedby` ✓ returns parent region; predicts *Asia*.
1×LLM adjudication	*Rationale:* "Gobi Desert lies across *Mongolia* and *China*, which are *contained by* the continent of *Asia*; 'contains' would flip direction."
Final Answer / GT	**Asia (predict) / Asia** ✓
Case 4: *in which continent is germany*	
Top-3 candidates	1) `location.location.containedby` (0.3405) 2) `base.locations.countries.continent` (0.3325) 3) `location.location.contains` (0.3270)
Top-K after pruning (K=2)	`containedby` `countries.continent`
Vector-only (no LLM)	Pick `containedby` ✗ tends to surface *EU* or administrative parents, hurting precision.
1×LLM adjudication	*Rationale:* "The target is a country → continent query; use `countries.continent` to directly map *Germany* to *Europe*."
Final Answer / GT	**Europe (predict) / Europe** ✓
Case 5: *what is the hometown of the person who said "Forgive your enemies, but never forget their names?"*	
Top-3 candidates	1) `quotation.author` → `person.place_of_birth` (0.3380) 2) `family.members` → `person.place_of_birth` (0.3310) 3) `quotation.author` → `location.people_born_here` (0.3310)
Top-K after pruning (K=2)	`quotation.author` → `place_of_birth` `family.members` → `place_of_birth`
Vector-only (no LLM)	Pick `quotation.author` → `place_of_birth` ✓ direct trace from quote to person to birthplace.
1×LLM adjudication	*Rationale:* "The quote's author is key; once identified, linking to their birthplace via person-level relation gives the hometown."
Final Answer / GT	**Brooklyn (predict) / Brooklyn** ✓
Case 6: *what is the name of the capital of Australia where the film "The Squatter's Daughter" was made*	
Top-3 candidates	1) `film.film_location.featured_in_films` (0.3360) 2) `notable_types` → `newspaper_circulation_area.newspapers` → `newspapers` (0.3330) 3) `film_location.featured_in_films` → `bibs_location.country` (0.3310)
Top-K after pruning (K=2)	`film.film_location.featured_in_films` `notable_types` → `newspaper_circulation_area.newspapers`
Vector-only (no LLM)	Pick `film.film_location.featured_in_films` ✓ retrieves filming location; indirectly infers capital via metadata.
1×LLM adjudication	*Rationale:* "The film's production location helps localize the city. Although not all locations are capitals, this film was made in Australia, where identifying the filming city leads to the capital."
Final Answer / GT	**Canberra (predict) / Canberra** ✓