Grounding Extended Type-KoPL with KG-Structure Feedback: Anchor-Probe Feedback and Cypher-Informed Reranking

Kotaro Oshima[1,*], Yoichi Takahashi[2]

[1]AIVALIX Inc., Tokyo, Japan
[2]Indigo Corporation, Tokyo, Japan

Abstract

Multi-hop question answering over closed-domain knowledge graphs (KGs) is challenging for local LLMs, which privacy and inference-cost constraints often mandate. Schema-driven search suits this setting because it constrains the candidate space in a single pass. However, a path that is consistent with the schema may be absent from the underlying data, which produces two failure modes: *empty-query failure*, where the generated Cypher returns an empty set, and *semantic-mismatch failure*, where an executable path returns incorrect entities. Extended Type-KoPL (ETK), our prior schema-driven formulation, guarantees schema consistency but does not address these failures. We retain ETK's design and inject instance-level evidence from the actual KG through two lightweight, training-free mechanisms. Anchor-Probe Feedback (APF) uses the anchor's observed edges to regenerate failed queries. Probe-Aware Cypher-Informed Reranking (PA-CIR) filters unreachable candidates and exposes Cypher-trial outputs to the reranker, preventing selection-side failures before execution. Both mechanisms share the same anchor-specific real-edge evidence, and the per-question LLM-call count stays bounded and independent of hop count. With the local gemma3:27B, the full model reaches F1 of 88.3–99.7 on MetaQA and 80.5–99.9 on PrimeKGQA, and improves PcQA (a reconstructed entity-set benchmark we release) from 81.1 to 86.3 F1, matching a cloud LLM on several subsets. Lightweight instance-level grounding thus improves schema-driven KGQA at bounded LLM cost.

Keywords

Knowledge Graph Question Answering, Type-KoPL, Local LLM, Cypher Probe, Schema-Driven Search, Grounding

CEUR-WS.org/Vol-4316/kge-full-2.pdf

1. Introduction

Multi-hop knowledge graph question answering (KGQA) traverses multiple entities and relations on a knowledge graph (KG) to answer a natural-language question, including questions that involve set operations and conditional constraints. In closed-domain KGs for healthcare, pharmaceuticals, and enterprise knowledge, the schema is stable and the types and relations are predefined. Schema-driven search is therefore a strong approach in these domains: it constrains the candidate space efficiently using the schema alone. At the same time, such environments often require deployment on local LLMs because of sensitive data and inference-cost constraints. This penalizes agentic methods, which interleave execution, observation, and correction: they issue at least one LLM call per interaction round, so their cost grows with the number of hops and retries, and each additional call multiplies the latency of local inference. We therefore study the setting of closed-domain KGs combined with local LLMs, where the central challenge is to reach high accuracy with a lightweight design that preserves schema-driven efficiency and avoids iterative loops.

Existing KGQA approaches fall broadly into three groups. One-shot generation methods ask an LLM to produce a query such as SPARQL directly, and are prone to execution failures caused by relation-selection errors or structural inconsistency. Agentic methods [1] improve executability by iterating over execution and correction, at the cost of multiple LLM calls per question. Schema-driven methods [2, 3] reduce search noise by constraining candidate paths with type and relation schemas, but they structurally leave open the failures that arise when the schema diverges from the actual data.

IJCKG 2026 Posters, Demos and Workshops, November 19–21, 2026, Bangkok, Thailand
*Corresponding author.
kotaro.oshima@aivalix.co.jp (K. Oshima); yoit@indigo.co.jp (Y. Takahashi)
© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-11-06

Accurate KGQA on local LLMs thus requires inference that avoids iteration and yet stays consistent with the structure of the actual KG.

Our previous work, Extended Type-KoPL (ETK) [4], belongs to the schema-driven family. It abstracts a question into a type-annotated intermediate representation (Type-KoPL), enumerates candidate relation paths that preserve type consistency on the schema graph, and progressively prunes them with embedding similarity and an LLM reranker, thereby securing schema-level consistency while remaining one-shot. However, when the selected candidate path is executed as Cypher on the actual KG, two residual failure modes remain. In an *empty-query failure*, the selected path has no real edge in the KG, so Cypher returns an empty set. This happens when the anchor lacks an edge for the predicted type pair, or when the reranker picks an unreachable candidate. In a *semantic-mismatch failure*, the reranker, judging candidates by semantic plausibility alone, selects a path that exists in the KG but is misaligned with the question, so Cypher returns the wrong entity set. Both modes arise where the schema diverges from the data, and neither can be resolved by schema-driven search alone.

Viewed by cause rather than by symptom, these two failure modes decompose into three causes. An empty result arises either because the reranker mis-selects an unreachable candidate although a reachable one exists in the candidate set (*selection-side reachability failure*), or because the candidate set contains no path reachable from the anchor at all (*generation-side reachability failure*). A wrong entity set arises because a reachable but semantically misaligned path is chosen (*semantic-selection failure*). This causal decomposition determines where an intervention must act: the first and third causes can be prevented at the selection stage, whereas the second requires regenerating the program itself.

In this work we retain ETK's schema-driven design and add two bounded instance-grounding mechanisms that inject real-edge evidence at inference time: Anchor-Probe Feedback (APF) and Probe-Aware Cypher-Informed Reranking (PA-CIR). APF acts at the correction stage. When execution returns an empty set, it acquires the anchor's real-edge set with a single Cypher probe and injects it into the correction prompt, so the LLM can regenerate a program that follows edges that actually exist. PA-CIR acts at the selection stage. It uses the same real-edge set to filter unreachable candidates before reranking, and exposes lightweight `LIMIT`-bounded Cypher-trial outputs to the reranker, so that selection is informed by what each candidate actually returns. The two mechanisms are distinguished by the failure cause they target: PA-CIR prevents the two selection-side causes, and APF repairs the generation-side cause. Neither uses an iterative agent loop.

Contributions. We (1) formulate the residual errors of schema-driven KGQA as the three failure causes above; (2) introduce APF and PA-CIR, two bounded instance-grounding mechanisms that address these causes without hop-by-hop LLM interaction; (3) integrate both into ETK, combining schema-level candidate construction with anchor-specific real-edge evidence and bounded execution feedback; and (4) evaluate across four LLMs and three benchmarks, including a reconstructed entity-set PcQA benchmark that we release. A preliminary version of ETK appeared in a JSAI SIG-SWO technical report [4]. The present paper is self-contained and substantially extends that work with the failure-cause formulation, APF, PA-CIR, and a broader multi-model, multi-benchmark evaluation.

2. Related Work

LLM–KG integration is a central path toward grounded reasoning [5], and LLM hallucination remains a structural concern [6]. We review the three main LLM-based KGQA families, then position our work against path retrieval methods that do not use LLMs.

2.1. Schema-Driven Methods

Schema-driven methods constrain candidate paths in a single pass using the KG's type and relation schema. SAFE [2] aligns an LLM-generated pseudo-query graph to the KG via a compact schema graph and Approximate Distance Join (ADJ), suppressing degradation from nonexistent relations or erroneous type connections. KGT [3] has the LLM infer a relation path on the schema graph from

key information extracted from the question, retrieves a subgraph, and feeds it back to the LLM to generate the answer. Our previous ETK [4] also secures schema consistency through type-annotated intermediate representations. These methods are one-shot and efficient, but none of them guarantees that a schema-consistent candidate is actually instantiated in the KG. This executability gap between schema and data is the problem we address.

2.2. Agentic Methods

Agentic methods let the LLM interact iteratively with external tools. ReAct [7] interleaves reasoning and acting, and Toolformer [8] learns API calls via self-supervision. Closer to KGQA, SPINACH [1] constructs SPARQL iteratively via execution feedback, Self-Debug [9] refines outputs using execution traces, RoG [10] plans and traverses the KG with the LLM in the loop, and Self-RAG [11] augments retrieval with self-reflective critique tokens. Execution feedback improves grounding, but these methods issue multiple LLM calls per question, which increases latency and cost in local-LLM deployments.

2.3. Intermediate Representation Generation

This family generates and executes explicit program representations. KQA Pro [12] annotates complex questions with both KoPL and SPARQL and is the foundation for program-execution KGQA. SymKGQA [13] constructs KoPL through few-shot incremental generation and grounds the program to KG elements via a RAG-based resolver (QUACK). IR generation shares the executability gap of the schema-driven family: whether the generated program resolves to actual KG entities is not guaranteed. QUACK closes the gap through its resolver, but at the cost of additional LLM calls per program element, so a lightweight design that grounds executability remains open in this family as well.

2.4. Path Retrieval without LLMs

Relation-path retrieval over KGs predates LLM-based KGQA, and several non-LLM families target the same sub-problem that our mechanisms address. Embedding-based methods such as EmbedKGQA [14] score answer entities in a KG-embedding space, sidestepping explicit path enumeration. GNN-based retrievers such as GraftNet [15] and PullNet [16] iteratively expand and reason over a question-specific subgraph. Reinforcement-learning methods such as DeepPath [17] and MINERVA [18] train a policy that walks the KG toward answer entities, and UniKGQA [19] unifies retrieval and reasoning in a single trained model. These methods are grounded in actual KG connectivity by construction: they operate directly on instance-level edges, which is exactly the evidence whose absence causes the failures we study in schema-driven LLM pipelines. The trade-off is that they require supervised or reinforcement training on the target KG, and the trained components are tied to that KG's entities and relations. Our setting instead assumes a frozen, locally deployed LLM and no task-specific training, which is attractive for closed-domain KGs that evolve or that cannot export training data. APF and PA-CIR import the instance-level connectivity evidence that trained path retrievers obtain implicitly, at the cost of a handful of Cypher queries rather than a training pipeline. We do not claim superiority over trained path retrievers on their own benchmarks; a controlled comparison against this family is future work (Sect. 8).

3. Background: Extended Type-KoPL

This section makes the paper self-contained. We first fix notation and the problem setting, then introduce KoPL and its typed extension, and finally walk through the four phases of the ETK pipeline [4], illustrated with a running example that we follow through the rest of the paper. Readers familiar with the previous work can skim to Sect. 3.7, which analyzes the residual failures that motivate our extensions.

3.1. Preliminaries and Problem Setting

Let $G = (V, E)$ be a knowledge graph with node set V and edge set $E \subseteq V \times R \times V$, where R is the set of relation types. Each node v carries a type $\tau(v) \in \mathcal{T}$ (in our Neo4j stores, the node label). The *schema graph* $S \subseteq \mathcal{T} \times R \times \mathcal{T}$ records which relation types connect which node types, aggregated over E. The schema graphs in this paper are small: 5–8 types and 7–13 relation types per KG.

A question q mentions one or more entities. An *anchor* is such a mentioned entity grounded to a KG node; traversal starts at anchors. Every question class we support is anchored and asks for a set of entities of a known target type. The task is to produce the answer entity set $Y \subseteq V$. The supported query classes are (i) relation paths of length L from a single anchor to the target type, and (ii) set combinations (INTERSECTION, UNION, EXCLUDE) of several such paths, each with its own anchor. Our benchmarks cover paths with $L \leq 3$ and intersections of two or three paths; UNION and EXCLUDE are supported by the representation and the implementation but not exercised by the benchmarks. Queries over attribute values, numerical comparison, sorting, and aggregation are out of scope (Sect. 8).

3.2. KoPL and Type-KoPL

KoPL (Knowledge-oriented Programming Language) was introduced with the KQA Pro dataset [12] as a program-style intermediate representation for complex questions. A question is decomposed into a sequence of primitive functions (e.g., find an entity, follow a relation, filter by concept) whose composition yields the answer. Unlike SPARQL or Cypher, KoPL is not a database query language. It is an abstraction designed to be easy for a parser or an LLM to produce, mirroring the reasoning structure of the question; execution requires compiling it to queries over a concrete KG.

Type-KoPL, introduced in our previous work [4], extends each relational step of KoPL with explicit source and target types. A multi-hop query is represented as

$$\mathcal{K} = \langle (\tau_i^{\text{src}}, \tau_i^{\text{tgt}}, h_i) \rangle_{i=1}^{L},$$

where $\tau_i^{\text{src}}, \tau_i^{\text{tgt}} \in \mathcal{T}$ are the types at the i-th hop, h_i is a natural-language *relation hint*, and L is the number of hops. The type sequence specifies the abstract path to follow on the schema graph, and the hints are used as semantic cues in later scoring. Each hop carries a hint rather than a committed relation. This keeps open the option of enumerating multiple candidate relations for the same type pair during schema search, deferring the final edge selection to the vector-similarity and reranking stages.

For example, "Which diseases are caused by the genes AT9283 targets?" becomes the two-hop program `[(drug, gene, "target"), (gene, disease, "cause")]` with anchor AT9283.

Running example. Throughout the paper we follow the PrimeKGQA question "*Which enzymes are targeted by Moxisylyte?*". Its Type-KoPL is the single-hop program `[(drug, gene/protein, "enzymes targeted")]` with anchor `Moxisylyte` and $L = 1$; the gold answer is the entity set {BCHE, CYP3A4, CYP3A43}.

3.3. Phase 1: Type-KoPL Generation

The LLM receives (i) the question and the anchor entities extracted from it; (ii) the KG's type list; (iii) the available relations; (iv) hop-structure constraint rules (for PATH, the type chain $\tau_i^{\text{tgt}} = \tau_{i+1}^{\text{src}}$; for INTERSECTION, the per-path anchor specification); and (v) a few few-shot examples. The output is structured JSON with an `operations` array and a `final_operation` (`path`, `intersection`, `union`, or `exclude`); for set operations the final answer is composed from the per-path entity sets. Anchor extraction is part of this phase: the LLM identifies anchor mentions, and each mention is grounded to a KG node by exact or case-insensitive name matching, without a separately trained entity linker. The robustness implications of this design are discussed in Sect. 8. The complete prompt templates and few-shot examples are released with the reproduction code (Appendix C).

3.4. Phase 2: Schema Graph Search

A Type-KoPL path specifies only type-level transitions. Multiple relations may exist between the same type pair, and sometimes no direct edge exists between the predicted pair. Phase 2 constructs concrete relation-path candidates by a stepwise breadth-first search on the schema graph. At each hop i, it enumerates the relation types connecting τ_i^{src} to τ_i^{tgt} in S; when no direct edge exists, it expands through one intermediate type (bounded depth 2). Each step then keeps only the top-3 relations ranked by embedding similarity between the relation name and the hint h_i, so the number of enumerated paths is bounded by 3^L with $L \leq 3$ regardless of instance size. The search touches only the schema graph, never instance nodes, so there is no instance-level explosion: its cost is a few embedding comparisons over at most tens of relation types. In the running example, Phase 2 enumerates the four schema relations connecting `drug` to `gene/protein` in PrimeKG — `target`, `enzyme`, `carrier`, and `transporter` — as four single-hop candidate paths.

3.5. Phase 3: Progressive Path Pruning

The resulting candidate paths are type-consistent, but many are semantically irrelevant to the question. ETK therefore projects the question and the schema paths into a common embedding space and selects the most plausible path by cosine similarity, using two-layer scoring that leverages relation hints.

3.6. Phase 4: Execution

The selected schema path is compiled to a Cypher pattern anchored at the anchor node, and execution returns the answer entity set. Relations that the schema records as directed are matched bidirectionally at execution time (`MATCH (a)-[r]-(n)`). This is a deliberate choice, for two reasons. First, a closed-domain KG stores each fact once in a canonical direction, while questions traverse relations in both directions ("which drugs target gene X" inverts the stored `drug -[target]-> gene` direction); undirected matching decouples the linguistic direction from the storage direction without doubling the schema with inverse relations. Second, the type constraints at both endpoints of every hop bound the risk that undirected matching admits unintended nodes; the residual risk concerns relations whose endpoints have the same type, and direction information is still tracked and used by the probe filter of Sect. 4.2.

3.7. Residual Failure Modes

When the schema diverges from the data, this pipeline fails in the two modes introduced in Sect. 1. An *empty-query failure* occurs when the selected path has no supporting edges at the anchor. The schema records that some drug has an `IS_A` edge, but a particular anchor may have none; a path through that edge type is schema-consistent yet unexecutable for this anchor. The failure is selection-side when a reachable candidate existed among the pruned candidates but the reranker picked an unreachable one, and generation-side when no candidate in the set was reachable, so no re-selection can help. A *semantic-mismatch failure* occurs when the selected path is executable but wrong, as in the running example: the `target` path returns the receptor genes `ADRA1A`, `ADRA1B`, `ADRA1D` instead of the gold enzymes. Nothing in schema-driven scoring detects this, because the choice is plausible at the schema level; only the actual query outputs reveal the mismatch. These three causes — selection-side reachability, generation-side reachability, and semantic selection — are the targets of the mechanisms we introduce next.

4. Proposed Mechanisms: APF and PA-CIR

Figure 1 shows where the two mechanisms intervene in the pipeline. Both consume the same *anchor probe*: a single Cypher query that aggregates the edges actually incident to the anchor node. The probe returns rows of (relation type, direction, neighbor label, count), ordered by decreasing count and capped

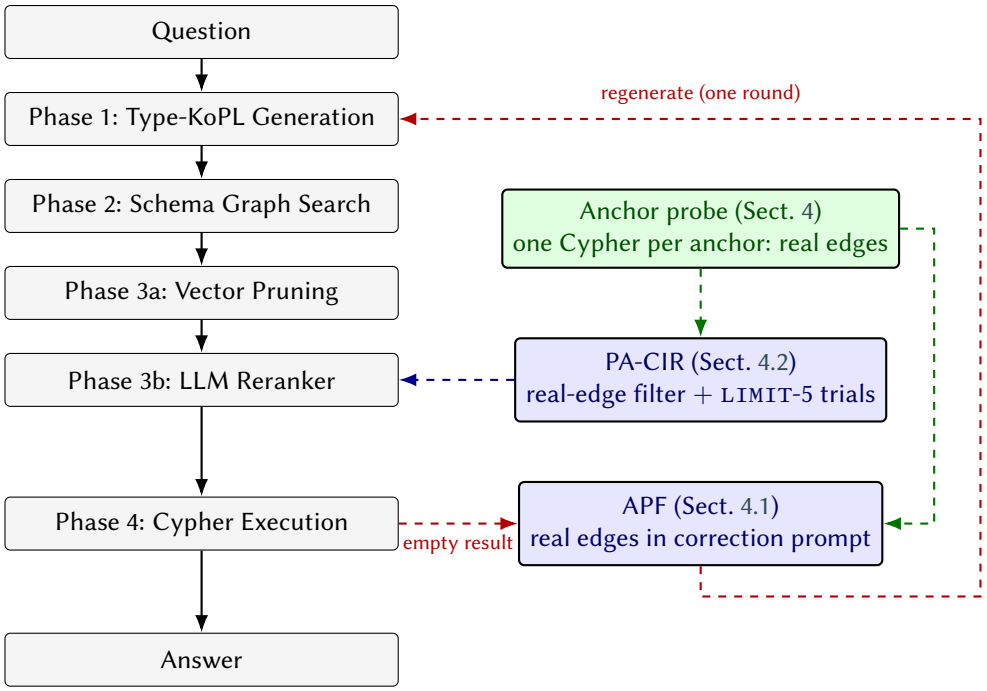


Figure 1: The ETK pipeline (Phases 1–4, gray) [4] with the mechanisms added in this paper. A single anchor probe (green) supplies the anchor's real-edge evidence to both mechanisms: PA-CIR (blue) filters and enriches the candidates given to the Phase 3b reranker, and APF (blue) injects the same evidence into the correction prompt when Phase 4 returns an empty set, triggering one regeneration from Phase 1. Within a correction round the already-obtained probe result is reused, so PA-CIR and APF are always grounded in the same evidence.

at 20 rows, so its output is a compact summary of the anchor's real neighborhood rather than a node dump. The probe is issued once per anchor and its result is reused; multi-anchor INTERSECTION programs contain one anchor per constituent path, and each constituent is probed independently.

4.1. APF: Anchor-Probe Feedback

APF targets generation-side reachability failures, in which the current candidate set provides no viable path from the anchor. When Phase 4 returns an empty set, APF adds the anchor-probe summary to the schema-level diagnosis in the correction prompt and asks the LLM to regenerate the Type-KoPL once. The regenerated program then passes through Phases 2–3 again: schema search, vector pruning, and re-selection. Thus, even when an edge exists for the type pair on the schema but not for the specific anchor (e.g., the schema admits `drug -[IS_A]-> drug` but the node AT9283 has no `IS_A` edge), the LLM can regenerate a program that follows the anchor's real edges. In our experiments APF performs at most one correction round, so the total LLM calls per question stay bounded (generation + reranker + at most one correction).

4.2. PA-CIR: Probe-Aware Cypher-Informed Reranking

PA-CIR makes the Phase 3b reranker execution-aware, preventing selection-side empty-query failure and semantic-mismatch failure before execution. Its name reflects two signals: the "Probe-Aware" component filters candidates using the anchor's real edges, whereas the "Cypher-Informed" component supplies trial outputs to the reranker. Because the reranker would otherwise select by semantic similarity alone, it may pick paths that return an empty set (unreachable from the anchor) or paths that return wrong entities (semantically misaligned with the question). PA-CIR addresses this with three steps in order:

1. Real-edge filter: using the anchor's real-edge set (obtained with the same Cypher probe as

APF; within the correction loop, the already-obtained set is reused), if the first relation leaving the anchor of a candidate path (the source relation) is not contained in that set, the candidate is excluded before reranking. Throughout the pipeline, schema-directed edges are traversed bidirectionally on the data graph (`MATCH (a)-[r]-(n)`); "leaving the anchor" refers to the relation incident to the anchor in either direction. This avoids passing to the reranker candidates that are clearly unreachable from the anchor.

2. Cypher trial samples: each candidate path that passes the real-edge filter is trial-executed with `LIMIT 5`, and the obtained sample entities are listed in the prompt. The reranker is therefore equipped to select based on the observed, empirical outcomes of each candidate path, rather than semantic similarity alone.
3. Retention of empty-trial candidates: candidates whose trials returned an empty set are not discarded but passed to the reranker, with `(no trial results)` explicitly noted in the prompt. This prevents a correct path that passed the real-edge filter from being dropped merely due to a failure to obtain samples.

These steps improve reranking in three ways: (i) early exclusion of unreachable candidates (preventing selection-side empty-query failure); (ii) instance-level execution results on the remaining candidates (reducing semantic-mismatch failure); and (iii) retention of correct answers whose trials returned no samples. Because APF and PA-CIR share the same anchor's real-edge set, both correction and reranking are grounded in the same anchor-specific edge information. A worked example of a rescued semantic-mismatch failure is given in Appendix A.

APF and PA-CIR introduce no additional LLM iterations: the per-question LLM-call count remains that of the ETK pipeline (generation, reranking, and at most one APF correction) and does not grow with the number of hops or KG size. Their overhead is limited to lightweight, LLM-free Cypher operations (an anchor probe and a few `LIMIT`-bounded trials) and a bounded increase in the reranker prompt. In contrast, agentic methods issue multiple LLM calls per question through repeated execution–observation–correction [1, 7, 10]; our mechanisms instead retain ETK's one-shot cost profile.

5. Experimental Setup

5.1. Benchmarks

We evaluate on three KGQA benchmarks with clearly defined type constraints.

- MetaQA [20]: movie domain, 1/2/3 hop, 5 types, 7 relations (public dataset).
- PrimeKGQA-Struct [21, 22]: biomedical domain (derived from PrimeKG), including 1/2 hop and 2/3-anchor intersection, 5 types, 12 relations. We use the version reconstructed for structured-query evaluation in the previous ETK paper [4].
- PcQA (reconstructed in this paper): pan-cancer domain, from the 405 free-form QA pairs of KGT [3]. The free-form answers did not support set matching, as they enumerated only part of the answer set, so we rebuilt entity-set gold. An LLM classifier (GPT-4o-mini, manually reviewed) mapped each question to one of 17 (anchor, target-type) query patterns, seed entities were grounded to KG nodes (36 corrected for synonymy or spelling), and the gold is the entity set returned by the pattern's Cypher. Of the 405 questions, 63 were non-entity-set (yes/no, trial metadata, explanatory), 14 had an anchor absent from the KG, and 87 returned an empty set, leaving 241. A type-consistency audit then removed 36 questions whose target type fell outside the 17 patterns (chiefly mutations associated with a drug's response), so their auto-generated gold was type-mismatched, yielding 205 questions (50.6%). PcQA is thus restricted to anchored, KG-answerable, pattern-covered questions.

KQA Pro [12] (centered on attribute values and modifiers, the domain of specialized methods such as SymKGQA) and WebQSP [23] are out of scope, because the type constraints and the anchored relation-path formulation that ETK assumes do not hold for them; Sect. 8 discusses this restriction.

5.2. LLMs Evaluated

To examine how performance varies across local and cloud models of different capacities, we evaluate ETK with four LLMs: llama3.1:8B, gemma3:12B, and gemma3:27B (local, served via a LiteLLM proxy), and gpt-4.1-mini (cloud). We measure the best configuration (Full) for all four models, and restrict the ablation (4 configurations $\times$ 8 subsets) to gemma3:27B and gpt-4.1-mini.

5.3. Methods Compared

We compare against the schema-driven methods SAFE [2] and KGT [3]. Both are single-pass methods without iterative execution, and they take candidate-generation strategies different from ETK's, so the contrast clarifies the position of ETK + APF + PA-CIR (schema-driven plus lightweight probe). The comparison uses the same conditions as the ETK runs on gpt-4.1-mini and gemma3:27B.

This is not a faithful reproduction of each method's original settings but a same-condition reimplementation aligned to ETK's KG, LLM, and set-based (entity-set) evaluation; the differences in Table 1 are thus relative, not comparisons with originally reported values. We kept each method's retrieval core unchanged (SAFE's schema-graph alignment with Approximate Distance Join; KGT's LLM relation-path inference with subgraph retrieval) and adapted only the final answer stage for set-based scoring: KGT's Phase 5 generation and SAFE's Module 4 selection were wrapped to return the full predicted entity set. INTERSECTION subsets, unsupported by SAFE, were left blank.

5.4. Evaluation Metric and Hyperparameters

The evaluation metric of this paper is the set-based F1 (%), the harmonic mean of precision and recall between the predicted and ground-truth entity sets. For sampling we used MetaQA n=1000, PrimeKGQA n=500, and all 205 samples for PcQA. All LLMs were decoded with temperature 0, making the outputs effectively deterministic; accordingly, we report a single run per configuration.

6. Results

This section presents the comparison with existing schema-driven methods, the main results with respect to LLM scale, and the APF/PA-CIR ablation. The contribution matrix (Δ relative to baseline) is in Appendix B. Across the ablation, the two mechanisms rescue failures of different causes. Under the same empty-set symptom, PA-CIR helps where the problem is selection-side (a reachable candidate exists, as in PrimeKGQA 1-hop), while APF helps where the generation-side dead-end dominates (no reachable candidate, as in PcQA).

6.1. Comparison with Existing Schema-Driven Methods (KGT, SAFE)

Table 1 shows the same-condition comparison with SAFE and KGT (Sect. 5.3). ETK + APF + PA-CIR outperforms both baselines across both LLMs and all evaluated subsets, with the local gemma3:27B reaching MetaQA 88.3–99.7, PrimeKGQA 80.5–99.9, and PcQA 86.3 (all F1). Three observations follow.

First, KGT restricts search to shortest type paths and commits to the single top-1 path by cosine score, with no beam and no backtracking. Once the chosen path returns an empty Cypher result it cannot recover, so it holds F1 77–87 for single-hop but collapses for multi-hop (9–12 for MetaQA 3-hop, 2 for PrimeKGQA 2-hop). The same collapse was observed in [4], indicating that it reflects KGT's single-path design rather than our reimplementation. Second, SAFE returns the endpoints of its top-k (k=10) semantically retrieved subgraphs as the answer set, with only loose type filtering and no precision pruning. It therefore over-predicts (high recall, low precision), stays at F1 62–86 on MetaQA multi-hop, and falls to 41–48 on PrimeKGQA 2-hop; it also does not support intersection queries. Third, ETK + APF + PA-CIR outperforms KGT and SAFE on all evaluated subsets, by $\geq$30 pt on many, with the largest margins on multi-hop and intersection types. Schema-driven KGQA can thus reach practical accuracy on local LLMs when grounded with lightweight execution feedback.

Table 1
Schema-driven baselines (KGT, SAFE) vs. our best configuration (ETK + APF + PA-CIR), F1 (%). MetaQA (n=1000), PrimeKGQA (n=500), PcQA (n=205). "—" marks INTERSECTION queries unsupported by SAFE; KGT/SAFE use a set-evaluation reimplementation of their final inference stage.

	MetaQA			**PrimeKGQA**				**PcQA**
Pipeline	1-hop	2-hop	3-hop	1-hop	2-hop	2-int	3-int	*all*
LLM = gemma3:27B								
KGT [3]	86.8	40.9	8.9	79.6	2.1	44.0	46.3	54.2
SAFE [2]	93.7	74.6	62.6	68.0	40.8	—	—	48.9
ETK + APF + PA-CIR (Ours)	**99.7**	**91.9**	**88.3**	**95.7**	**80.5**	**98.5**	**99.9**	**86.3**
LLM = gpt-4.1-mini								
KGT [3]	86.5	48.1	12.4	76.5	2.4	44.0	50.2	57.8
SAFE [2]	96.2	85.9	65.3	74.7	47.8	—	—	49.8
ETK + APF + PA-CIR (Ours)	**99.5**	**99.9**	**99.4**	**95.3**	**96.1**	**98.2**	**99.0**	**86.1**

Table 2
F1 (%) of the best configuration (baseline + APF + PA-CIR) across four LLMs and eight subsets; best per column in bold. MetaQA (n=1000), PrimeKGQA (n=500), PcQA (n=205).

	MetaQA			**PrimeKGQA**				**PcQA**
LLM	1-hop	2-hop	3-hop	1-hop	2-hop	2-int	3-int	*all*
llama3.1:8B	97.8	4.4	14.7	91.4	53.9	81.2	88.3	78.4
gemma3:12B	**99.9**	82.3	55.2	95.0	48.7	68.3	58.8	82.2
gemma3:27B (local)	99.7	91.9	88.3	**95.7**	80.5	**98.5**	**99.9**	**86.3**
gpt-4.1-mini	99.5	**99.9**	**99.4**	95.3	**96.1**	98.2	99.0	86.1

6.2. Main Results: Scaling with Respect to LLM Scale

Table 2 shows the accuracy of the best configuration (baseline + APF + PA-CIR) across the four LLMs.

The trends differ substantially across KGs (values are F1). On MetaQA multi-hop, F1 rises steeply with model capacity: 2-hop by +95.5 pt from 4.4 (8B) to 99.9 (gpt-4.1-mini), and 3-hop by +84.7 pt from 14.7 to 99.4. Despite the simple movie domain, multi-hop reasoning remains the main bottleneck. PrimeKGQA 1-hop is robust even for small LLMs (8B 91.4, 27B 95.7), but 2-hop reaches a usable range (80–96) only at 27B and above. Intersection types saturate, with several LLMs above 98, yet 12B is unstable (2-int 68.3, 3-int 58.8), indicating that simultaneous resolution of multiple anchors is fragile at that scale. PcQA varies little across models (78.4–86.3, about 86 for both 27B and gpt-4.1-mini), because the Cypher-verified gold makes the pipeline's internal coverage the effective upper bound. Overall, gemma3:27B at its best configuration matches the cloud LLM on intersection types and PcQA, with a gap remaining on MetaQA multi-hop and PrimeKGQA 2-hop.

6.3. Ablation

Tables 3 and 4 show the four-configuration ablation (baseline / +APF / +PA-CIR / Full) for gpt-4.1-mini and gemma3:27B under identical settings. In what follows we call a subset *saturated* for a model when its baseline F1 already exceeds 99, so the failures our mechanisms target are essentially absent, and *unsaturated* otherwise.

For gpt-4.1-mini (Table 3), MetaQA is saturated in all four configurations (even the baseline is at F1 99.6–99.9), so neither mechanism can show a gain there. On the unsaturated PrimeKGQA and PcQA the division of responsibility is clear. PA-CIR dominates on PrimeKGQA 1-hop (89.5 → 95.4, +5.9 pt) while APF alone has no effect there (−0.2 pt). Conversely, APF dominates on PcQA (+4.4 pt). In other words, PA-CIR is decisive where a reachable candidate exists and the failure is selection-side, whereas APF is decisive where no reachable candidate exists and regeneration is required. Integrating both, Full reaches the best score on PcQA (+4.5 pt, F1 81.6 → 86.1) and on PrimeKGQA 2-hop (+3.4 pt), with each

Table 3
Four-configuration ablation with `gpt-4.1-mini` (configurations defined in Sect. 6.3). n=1000 (MetaQA) / 500 (PrimeKGQA) / 205 (PcQA, all samples). Cells are F1 (%); best per column in bold (ties marked).

	MetaQA			PrimeKGQA				PcQA
Config	1-hop	2-hop	3-hop	1-hop	2-hop	2-int	3-int	*all*
baseline (reranker only)	99.7	**99.9**	**99.6**	89.5	92.7	96.4	99.0	81.6
+PA-CIR	99.6	**99.9**	99.5	**95.4**	95.7	98.0	**99.2**	82.8
+APF	**99.8**	**99.9**	99.4	89.3	92.5	96.8	**99.2**	86.0
+APF+PA-CIR (Full)	99.5	**99.9**	99.4	95.3	**96.1**	**98.2**	99.0	**86.1**

Table 4
Four-configuration ablation with local `gemma3:27B` (configurations as in Table 3, served via a LiteLLM proxy). n=1000 / 500 / 205. Cells are F1 (%); best per column in bold (ties marked).

	MetaQA			PrimeKGQA				PcQA
Config	1-hop	2-hop	3-hop	1-hop	2-hop	2-int	3-int	*all*
baseline (reranker only)	93.6	84.4	87.7	94.7	**81.6**	98.3	96.8	81.1
+PA-CIR	99.6	91.0	**88.4**	**96.1**	81.2	98.5	99.8	80.7
+APF	**99.7**	91.3	88.0	94.6	81.4	**98.6**	99.8	85.0
+APF+PA-CIR (Full)	**99.7**	**91.9**	88.3	95.7	80.5	98.5	**99.9**	**86.3**

mechanism working where it is most effective.

For gemma3:27B (Table 4), the net effect is broadly positive. On MetaQA 2-hop, PA-CIR and APF gain +6.6 and +6.9 pt respectively, and on PcQA they move the score by −0.4 and +3.9 pt. PA-CIR alone does not fall below baseline on PrimeKGQA 1-hop (94.7 → 96.1) or 2-int (98.3 → 98.5), and any decrease stays within 0.4 pt (1.1 pt for Full), confined to PrimeKGQA 2-hop and PcQA. The two mechanisms do not conflict, and the integrated Full is the most stable and highest: best or tied on MetaQA 1-/2-hop, PrimeKGQA 3-int, and PcQA, with the gap to the best standalone configuration around 1 pt elsewhere. The gain of Full is not the sum of the standalone gains (MetaQA 2-hop: PA-CIR +6.6, APF +6.9, Full +7.5 pt). This is because the two mechanisms handle different failure modes and cannot double-count the same rescued question, not because they interfere.

7. Discussion

7.1. Correction of the Schema–Instance Gap by APF

When Phase 4 returns an empty set due to KG sparsity, the previous ETK's correction feedback returns only the schema-level diagnosis of whether an edge exists for the type pair. That is the same granularity as Phases 2–3, so the LLM tends to loop, regenerating the same program. The real-edge set provided by APF adds instance-level evidence that the schema alone cannot supply. In this sense, APF moves correction from schema-level diagnosis to anchor-level grounding.

The outcome-level statistics support the interpretation that APF primarily mitigates generation-side failures. As a proxy we track zero-overlap wrong answers (recall 0), a category that includes the cases where Cypher returns an empty set because no candidate path is reachable from the anchor. On PcQA, where KG sparsity is broad, zero-overlap wrong answers fall from 31 at baseline to 22 with APF alone, whereas PA-CIR alone reduces them only to 29. A substantial portion of these errors therefore requires regeneration rather than re-selection. The same account explains why the APF gain is negligible on gpt-4.1-mini's saturated MetaQA: where the baseline already answers almost everything, there are no generation-side failures left for APF's instance-level evidence to repair, so the mechanism has headroom only on unsaturated subsets.

7.2. PA-CIR's Preventive Effect and Mechanism Integration

PA-CIR prevents selection-side reachability and semantic-selection failures at the selection stage. In the standalone configurations of Sect. 6.3, on PrimeKGQA 1-hop zero-overlap wrong answers fall from 41 at baseline to 19 with PA-CIR alone, and over-prediction cases (recall 1 with extra predictions) fall from 28 to 14. These outcome-level changes are consistent with reduced selection-side reachability errors and reduced semantic-selection errors, respectively. Regeneration-only APF leaves both statistics unchanged (42 and 28): once prevention succeeds at the selection stage, correction has nothing to repair. Because PA-CIR excludes only candidates that fail the real-edge filter while retaining the baseline path, it stays at or above baseline on most subsets (e.g., gemma3:27B's PrimeKGQA 1-hop, 94.7 $\rightarrow$ 96.1), and its only decreases are within 0.4 pt (on PrimeKGQA 2-hop and PcQA). It is therefore not a substantive degradation factor in local-LLM environments.

The integrated effect varies by subset, with one mechanism dominating or the two acting complementarily. Because they act on different failure causes — PA-CIR on the selection side, APF on generation-side reachability — the integration outperforms either alone when failures spread across both, as on PcQA, where Full exceeds both standalone configurations. The gap to a strict sum reflects the absence of double-counting and the saturation ceiling, not interference.

8. Limitations

The most consequential limitation is query expressiveness. ETK, with or without our mechanisms, handles anchored relation-path retrieval with typed endpoints, composed by set operations. It does not handle attribute-value references, numerical comparison, sorting, aggregation, or unanchored questions, which is why KQA Pro and WebQSP are out of scope (Sect. 5.1). We consider this restriction acceptable for the closed-domain deployments that motivate the work, where questions overwhelmingly follow anchored patterns, but it separates our results from general-purpose KGQA, and extending Type-KoPL toward attributes and ordering operators is the natural next step (Sect. 9).

Several design choices are heuristic and carry documented risks. APF cannot distinguish a legitimately empty answer from a failure; our benchmarks exclude empty-gold questions, so deployments where the empty set is a valid answer would need an additional decision step before regeneration. One can also argue that schema–data misalignment should be repaired in the data itself, as a data-management task, rather than compensated at query time; we agree that both are needed, and our mechanisms address the reality that closed-domain KGs are rarely perfectly aligned at any given moment. Exposing real edges can bias regeneration toward what exists near the anchor, and PA-CIR's trial samples are the first five rows returned by the engine, so an unlucky sample can misrepresent a candidate. The retention of empty-trial candidates is a conservative mitigation, not a guarantee. Anchor grounding relies on the Phase 1 LLM plus name matching, so paraphrased entity mentions that name matching cannot resolve become failures; we have not quantified robustness to systematic question reformulation.

The evaluation has three caveats. First, the KGT and SAFE numbers come from same-condition reimplementations whose final answer stage we adapted for set-based scoring (Sect. 5.3); despite our efforts to keep each retrieval core intact, and the consistency of KGT's multi-hop collapse with earlier measurements, an underestimation of the originals cannot be fully excluded, which is why the code is released for audit. Second, we report single deterministic runs (temperature 0) without significance tests. The headline margins over the baselines exceed 30 pt on many subsets and are far outside plausible sampling noise, but small deltas within about 1 pt, such as those between Full and the best standalone configuration, should be read as ties. Third, all local-LLM conclusions are drawn from the models available to us (up to gemma3:27B). Stronger future local models will likely saturate more subsets, shrinking the mechanisms' measurable gains, just as we observe on gpt-4.1-mini's MetaQA. However, the schema–instance divergence that the mechanisms compensate is a property of the data, not of the model: no amount of parametric capability tells a model which edges exist around a specific anchor without observing them. We therefore expect the grounding evidence to remain useful wherever KG sparsity exists, while acknowledging that this expectation is not yet verified on newer models. Finally,

a controlled comparison against trained path retrievers (GNN- and RL-based, Sect. 2.4) would sharpen the positioning of the training-free approach and remains future work.

9. Conclusion and Future Work

We proposed APF and PA-CIR, two lightweight mechanisms that ground ETK in the actual KG's edges without an iterative agent loop. APF injects anchor-specific edge evidence into correction to recover from empty-query failure via regeneration. PA-CIR filters unreachable candidates using the anchor's real edges and exposes bounded Cypher-trial outputs to the reranker, preventing selection-side failures before execution. The two are complementary not merely because they intervene at different stages, but because they address different failure causes: APF repairs generation-side reachability failures, whereas PA-CIR prevents selection-side reachability and semantic-selection errors. A single one-shot architecture thus addresses both the iteration-cost burden of agentic methods and the schema–data divergence left by purely schema-driven methods. Across MetaQA, PrimeKGQA, and the reconstructed PcQA subset, the full model improved over the ETK baseline on most subsets and outperformed same-condition reimplementations of KGT and SAFE, with the largest gains on multi-hop and intersection queries and on the local gemma3:27B. Lightweight instance-level grounding can substantially improve schema-driven KGQA, especially in local-LLM settings.

We see three directions for future work. The first is end-to-end verification with text-to-KG conversion: since a structured KG rarely exists a priori, our KG-construction pipeline should be compared under identical conditions against RAG without KG construction on closed domains where proper-noun coverage matters, such as biomedicine and finance. The second is complex downstream tasks: the real-edge grounding of APF and PA-CIR may serve as a fact-retrieval component for reasoning chains, interactive QA, and summarization pipelines. The third is richer query generation: integrating attributes, modifiers, the remaining set operations, and ordering operators into Type-KoPL to reach more complex KGQA such as KQA Pro, which would also lift the expressiveness limitation discussed in Sect. 8.

Declaration on Generative AI

During the preparation of this work, the authors used Claude (Anthropic) in order to: draft and reword parts of the manuscript, improve writing style, check grammar and spelling, and assist with LaTeX formatting. The proposed method, the implementation, the experiments, and the analyses were designed and conducted by the authors. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] Liu, S., Semnani, S., Triedman, H., Xu, J., Zhao, I.D., Lam, M.: SPINACH: SPARQL-based information navigation for challenging real-world questions. In: Findings of EMNLP, pp. 15977–16001 (2024)

[2] Lee, S., Park, S., Han, W.S.: SAFE: Schema-driven approximate distance join for efficient knowledge graph querying. In: Proc. of EMNLP, pp. 17475–17500 (2025)

[3] Feng, Y., Zhou, L., Ma, C., Zheng, Y., He, R., Li, Y.: Knowledge graph-based thought: A knowledge graph-enhanced LLM framework for pan-cancer question answering. GigaScience **14**, giae082 (2025)

[4] Oshima, K., Takahashi, Y., Nakayama, T., Hibarino, S.: Extended Type-KoPL: Type-constrained knowledge program generation with LLMs for knowledge graph question answering (in Japanese). Technical Report SIG-SWO-068, Japanese Society for Artificial Intelligence (2026)

[5] Pan, S., Luo, L., Wang, Y., Chen, C., Wang, J., Wu, X.: Unifying large language models and knowledge graphs: A roadmap. IEEE Trans. Knowl. Data Eng. **36**(7), 3580–3599 (2024)

[6] Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., et al.: Survey of hallucination in natural language generation. ACM Comput. Surv. **55**(12), 1–38 (2023)

[7] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., Cao, Y.: ReAct: Synergizing reasoning and acting in language models. In: Proc. of ICLR (2023)
[8] Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., et al.: Toolformer: Language models can teach themselves to use tools. Adv. Neural Inf. Process. Syst. **36**, 68539–68551 (2023)
[9] Chen, X., Lin, M., Schärli, N., Zhou, D.: Teaching large language models to self-debug. In: Proc. of ICLR (2024)
[10] Luo, L., Li, Y.F., Haffari, G., Pan, S.: Reasoning on graphs: Faithful and interpretable large language model reasoning. In: Proc. of ICLR (2024)
[11] Asai, A., Wu, Z., Wang, Y., Sil, A., Hajishirzi, H.: Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In: Proc. of ICLR (2024)
[12] Cao, S., Shi, J., Pan, L., Nie, L., Xiang, Y., Hou, L., et al.: KQA Pro: A dataset with explicit compositional programs for complex question answering over knowledge base. In: Proc. of ACL, pp. 6101–6119 (2022)
[13] Agarwal, P., Kumar, N., Bedathur, S.: SymKGQA: Few-shot knowledge graph question answering via symbolic program generation and execution. In: Proc. of ACL, pp. 10119–10140 (2024)
[14] Saxena, A., Tripathi, A., Talukdar, P.: Improving multi-hop question answering over knowledge graphs using knowledge base embeddings. In: Proc. of ACL, pp. 4498–4507 (2020)
[15] Sun, H., Dhingra, B., Zaheer, M., Mazaitis, K., Salakhutdinov, R., Cohen, W.W.: Open domain question answering using early fusion of knowledge bases and text. In: Proc. of EMNLP, pp. 4231–4242 (2018)
[16] Sun, H., Bedrax-Weiss, T., Cohen, W.W.: PullNet: Open domain question answering with iterative retrieval on knowledge bases and text. In: Proc. of EMNLP-IJCNLP, pp. 2380–2390 (2019)
[17] Xiong, W., Hoang, T., Wang, W.Y.: DeepPath: A reinforcement learning method for knowledge graph reasoning. In: Proc. of EMNLP, pp. 564–573 (2017)
[18] Das, R., Dhuliawala, S., Zaheer, M., Vilnis, L., Durugkar, I., Krishnamurthy, A., Smola, A., McCallum, A.: Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. In: Proc. of ICLR (2018)
[19] Jiang, J., Zhou, K., Zhao, W.X., Wen, J.R.: UniKGQA: Unified retrieval and reasoning for solving multi-hop question answering over knowledge graph. In: Proc. of ICLR (2023)
[20] Zhang, Y., Dai, H., Kozareva, Z., Smola, A., Song, L.: Variational reasoning for question answering with knowledge graph. In: Proc. of AAAI. vol. 32, pp. 6069–6076 (2018)
[21] Yan, X., Westphal, P., Seliger, J., Usbeck, R.: Bridging the gap: Generating a comprehensive biomedical knowledge graph question answering dataset. In: Proc. of ECAI, pp. 1198–1205 (2024)
[22] Chandak, P., Huang, K., Zitnik, M.: Building a knowledge graph to enable precision medicine. Sci. Data **10**(1), 67 (2023)
[23] Yih, W., Richardson, M., Meek, C., Chang, M.W., Suh, J.: The value of semantic parse labeling for knowledge base question answering. In: Proc. of ACL (Short Papers), pp. 201–206 (2016)

A. Example of a PA-CIR Cypher Trial Execution

We show a concrete example (gpt-4.1-mini, PrimeKGQA 1-hop). For "*Which enzymes are targeted by Moxisylyte?*" the anchor is `Moxisylyte` (drug) and the gold answer is BCHE, CYP3A4, CYP3A43.

Baseline. Pruning ranked `drug -[target]-> gene/protein` first; on semantic similarity alone ("target" vs. "enzyme"), the reranker adopted it but returned receptor genes (ADRA1A, ADRA1B, ADRA1D) and failed.

+ *APF* + *PA-CIR.* The real-edge filter first dropped candidates whose source relation lay outside `Moxisylyte`'s real-edge set; each survivor was trial-executed and passed to the reranker with sample entities. The reranker then selected `drug -[enzyme]-> gene/protein`, and Phase 4 returned BCHE, CYP3A4, CYP3A43. The added cost was one anchor probe plus a few `LIMIT 5` trials, at one LLM call as before.

B. Contribution Matrix: Structural Contrast between LLMs

Table 5
Contribution matrix: Δ F1 (pt) vs. baseline by LLM $\times$ subset. "$\sim$": $|\Delta| < 1$ pt; bold: largest positive row-gain per LLM (ties marked).

	gpt-4.1-mini			**gemma3:27B**		
Subset	+APF	+PA-CIR	+APF+PA-CIR	+APF	+PA-CIR	+APF+PA-CIR
PcQA	**+4**	+1	**+4**	+4	$\sim$	**+5**
PrimeKGQA 1-hop	$\sim$	**+6**	**+6**	$\sim$	**+1**	**+1**
PrimeKGQA 2-hop	$\sim$	**+3**	**+3**	$\sim$	$\sim$	−1
PrimeKGQA 2-int	$\sim$	**+2**	**+2**	$\sim$	$\sim$	$\sim$
PrimeKGQA 3-int	$\sim$	$\sim$	$\sim$	**+3**	**+3**	**+3**
MetaQA 1-hop	$\sim$	$\sim$	$\sim$	**+6**	**+6**	**+6**
MetaQA 2-hop	$\sim$	$\sim$	$\sim$	+7	+7	**+8**
MetaQA 3-hop	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$	$\sim$

Table 5 tabulates the Δ relative to baseline from Tables 3 and 4. For gpt-4.1-mini the saturated band (MetaQA) stays in the noise region ($\sim$), while the unsaturated band splits clearly: PrimeKGQA 1-hop responds to PA-CIR (+6), PcQA responds to APF (+4), and Full tracks the best mechanism on each. gemma3:27B has a broad unsaturated band where all three configurations lift MetaQA 1-/2-hop and PrimeKGQA 3-int by ≥ 3 pt, and Full exceeds both standalone configurations on PcQA and MetaQA 2-hop. In the larger LLM the single-mechanism maximum carries over to Full, whereas in the mid-size local LLM the gains accumulate gradually.

C. Code and Prompt Availability

The implementation, prompts, and seed data are released at

https://github.com/AIVALIX/Extended_Type_KoPL