When Thinking Before Retrieval Hurts: TraceBound Diagnostics for Adaptive Knowledge-Graph Retrieval

Partha Sarathi Purkayastha [1]

Abstract

Adaptive retrieval promises to make knowledge-graph question answering more robust by letting a controller search, inspect neighborhoods, revise actions, and stop when evidence is sufficient. We study this premise by introducing **TraceBound**, a lightweight profile- and trace-conditioned diagnostic protocol for an ARK-style retriever on text-rich knowledge graphs. TraceBound exposes a compact query profile before retrieval, issues short trace hints after observable failure symptoms, and logs trajectory counters, while keeping graph data, tools, gold labels, and ranking metrics fixed. Across STaRK validation and held-out subsets, the added conditioning improves inspectability but consistently reduces retrieval quality under open-weight controllers. Paired trajectory analysis localizes the degradation to repeated calls, zero-result calls, and misallocated exploration budget, while stricter interaction budgets shorten trajectories without repairing the policy. The result diagnoses the common failure mode in that "thinking before retrieval" must be evaluated as a control problem over action selection, not as a prompt-format change.

1. Introduction

Retrieval-augmented generation converts a language-model question answering problem into a joint modeling and retrieval problem where the model must not only reason over evidence, but also decide which evidence should be exposed in the first place. This separation becomes especially consequential for text-rich knowledge graphs. A relevant answer may be recoverable from node descriptions, relation paths, type constraints, or a combination of all three. A natural response is to replace one-shot retrieval with an adaptive retrieval agent where the controller searches broadly, follows promising graph neighborhoods, observes intermediate evidence, and then decides what to retrieve next.

This paper studies a deceptively simple question - When an adaptive graph-retrieval agent is given explicit pre-retrieval "thinking" and observation-level trace feedback, does retrieval actually improve? The question matters beyond leaderboard performance. It is also a prerequisite for personalization. A future personalized retriever should use compact user state before retrieval, not only after documents are fetched. However, if profile-like signals are injected into a retrieval controller without a calibrated action policy, they may change the trajectory without improving the ranked answer list.

We introduce **TraceBound**, a diagnostic protocol for adaptive knowledge-graph retrieval. TraceBound has two roles. First, it is an intervention in which the controller receives a compact query profile before acting and receives trace-conditioned hints when the trajectory exhibits symptoms such as repeated calls, zero-result calls, or budget exhaustion. Second, it is an instrumentation layer where every trajectory is converted into process counters that can be paired with final ranking metrics. The protocol is deliberately lightweight. It does not alter graph data, training labels, gold answers, ranking metrics, or the underlying search tools. This makes the comparison strict, so if profile and trace conditioning are useful, they should improve the agent under the same retrieval interface.

The empirical result is strong but not in the direction one might expect. On STaRK, TraceBound makes the retrieval process more inspectable but reduces retrieval quality relative to the unconditioned ARK-style controller in our open-weight setup. The degradation appears in a Qwen3-14B validation ablation, in a Qwen3-30B-A3B held-out subset, and in shared-ID paired comparisons across PRIME, MAG, and Amazon. The trace analysis explains why the result is not merely noise as repeated calls and zero-result calls are associated with substantially larger negative rank deltas, and a stricter budget reduces trajectory length without recovering accuracy.

We organize the study around three research questions.

[1]Department of Computer Science (D-INFK), ETH Zürich, Zürich, Switzerland. Correspondence to: Partha Sarathi Purkayastha <ppurkayastha@ethz.ch>.

Accepted at the Failure Modes in Agentic AI (FAGEN) Workshop at ICML 2026, Seoul, South Korea. Copyright 2026 by the author(s).

RQ1: does profile/trace conditioning improve standard retrieval metrics when the underlying graph tools and evaluator are fixed? **RQ2**: if ranking quality changes, can trajectory counters explain the direction of the change? **RQ3**: does imposing a stricter interaction budget repair the failure, or does the system require a different action policy? These questions make the paper a controlled analysis of adaptive retrieval behavior rather than a comparison of unrelated retriever architectures.

Our contributions are threefold. First, we provide a reproducible profile/trace intervention for adaptive graph retrieval that isolates controller behavior from dataset and metric changes. Second, we report paired ranking and trace diagnostics showing when the intervention hurts. Third, we identify the operational mechanism of failure that profile and trace state are useful for diagnosis, but harmful as unconditional prompt context unless paired with a calibrated retrieval-action policy.

2. Related Work

Retrieval-augmented generation and adaptive retrieval. Retrieval-augmented generation combines parametric generation with non-parametric evidence access (Lewis et al., 2020). Dense passage retrieval established a strong neural retrieval paradigm for open-domain question answering (Karpukhin et al., 2020), while later systems interleave retrieval with reasoning, reflection, or critique (Trivedi et al., 2023; Asai et al., 2024). These methods motivate interactive evidence acquisition, but many assume a document collection rather than a text-rich graph whose structure constrains valid answers.

Language-model agents and tool use. Tool-using agents such as ReAct combine reasoning traces with environment actions (Yao et al., 2023). Reflexion, Self-Refine, and Toolformer explore variants of feedback, iterative improvement, and learned tool use (Shinn et al., 2023; Madaan et al., 2023; Schick et al., 2023). In principle, such methods suggest that a retrieval controller should improve after seeing observations. Our results identify a boundary condition where observation feedback can make the process more measurable while still degrading final retrieval when the next-action policy is not calibrated.

Graph retrieval over textual and relational knowledge bases. Knowledge-graph reasoning methods such as Think-on-Graph expose graph traversal to language models (Sun et al., 2023). STaRK is the most relevant benchmark for our setting because it evaluates retrieval over semi-structured textual and relational knowledge bases across product search, academic paper search, and precision medicine (Wu et al., 2024). Autofocus or AF-Retriever studies multi-hop retrieval over semi-structured knowledge bases by combining entity search, LLM-generated Cypher queries, node-set joins, vector search, and reranking (Boer et al., 2026). GraphRAFT explores retrieval-augmented fine-tuning for graph databases (Clemedtson & Shi, 2025). ARK is the closest architectural reference and it gives a language model two graph tools, global lexical search with one-hop neighborhood exploration, so that the model can balance breadth-oriented discovery with depth-oriented traversal (Polonuer et al., 2026). We build on the same broad premise but focus on trace-conditioned failure analysis rather than claiming a new state-of-the-art retriever.

Personalization before retrieval. Personalized retrieval work argues that user-specific semantics should influence retrieval before documents are fetched. Personalize Before Retrieve (PBR) performs personalized query expansion using style-aware pseudo feedback and corpus-structure alignment (Zhang et al., 2025). PersonaBench evaluates whether systems can answer questions requiring access to synthetic personal information (Tan et al., 2025). Our experiments do not introduce real user profiles; the profile in TraceBound is query-derived. The relevance to personalization is methodological as a user profile is another pre-retrieval signal, and our results show that adding such a signal globally to an agent prompt is not a substitute for learning when it should change the retrieval action.

Failure diagnostics for agents. Final answer metrics can hide process failures as an agent may make redundant calls, chase empty branches, or recover by chance after an inefficient trajectory. Agent evaluation work therefore increasingly emphasizes trajectories, reproducible failure triggers, and process-level measurements (Liu et al., 2024; Xie et al., 2024). TraceBound follows this instrumentation-first view. We pair standard retrieval metrics with trajectory counters so that failure is attributed to observable behavior rather than only to aggregate score changes.

3. TraceBound: Methodology and Implementation Details

3.1. Problem setting and interface

Each example consists of a natural-language query q, a text-rich knowledge graph $G = (V, E)$, and a gold answer set $Y \subseteq V$. A retrieval agent returns an ordered list $\hat{Y} = (v_1, \ldots, v_k)$, which is evaluated with Hit@1, Hit@5, Recall@20, and MRR. The agent can call two graph tools: *global search*, which retrieves candidate nodes from textual descriptors, and *neighborhood exploration*, which expands a selected node through one-hop relations. The controller operates in an action-observation loop until it emits final answers or reaches a maximum interaction budget.

Table 1. **Controller variants.** "Profile" means the controller receives a compact pre-retrieval query profile. "Trace hint" means the controller receives short observation-level feedback derived from trajectory counters. "Logged diagnostics" means the trajectory is converted into process metrics; these metrics are not used by the final ranking evaluator.

Variant	Profile	Trace hint	Logged diagnostics
Unconditioned agent	no	no	final-answer metrics
Profile-only agent	yes	no	profile metadata
Trace-only agent	no	yes	process counters
TraceBound agent	yes	yes	profile + process counters

3.2. The TraceBound intervention

TraceBound adds profile conditioning and trace conditioning to this controller. The input remains the original query and graph; the output remains a ranked node list plus an execution trajectory. The intervention changes the controller context, not the task definition.

Query profile. Before the first tool call, TraceBound asks the controller to form a compact profile containing predicted answer type, important query facets, relation hints, and retrieval risks. The profile is not a user memory; it is a query-derived stand-in for the kind of compact state that a personalized retriever would eventually use. Its purpose is to test whether an explicit pre-retrieval state helps the controller decide how to search.

Trace hints. After each observation, TraceBound updates a trace state with counters for global calls, neighborhood calls, zero-result calls, repeated calls, step count, and remaining budget. The controller receives a short diagnostic hint only when the trace indicates a potential failure pattern, such as repeated identical calls or empty neighborhoods. The hint is intentionally constrained; it can recommend re-anchoring, relaxing a constraint, tightening type focus, or stopping, but it does not reveal labels or gold answers.

Instrumentation. Every trajectory is logged as a sequence of actions, observations, and counters. This lets us compute paired process metrics on the same examples used for ranking evaluation. The key design choice is separation - ranking metrics are computed only from final answer lists, while trace metrics are computed only from trajectories.

3.3. Evaluated controller variants

Table 1 defines the four variants. The unconditioned agent is the baseline: it receives the original query, tool descriptions, and prior observations, but no additional profile or trace hint. Profile-only and trace-only isolate the two ingredients. TraceBound combines both.

3.4. Algorithmic summary

Given a query q, TraceBound first constructs a query profile $p(q)$. At step t, the controller observes the query, profile, previous observations, and current trace summary s_t. It chooses either global search, neighborhood exploration, or final-answer emission. After a tool call, the monitor updates s_{t+1} by checking whether the result was empty, whether the call repeated an earlier call, how much budget remains, and whether enough candidates have been accumulated. The method is therefore not a new retriever index; it is a controller-side protocol for profile-conditioned action selection and trace-conditioned diagnostics.

3.5. Design invariants

A diagnostic intervention is only useful if it does not move the evaluation target. We therefore preserve five invariants. Table 8 summarizes what is fixed and what is changed. The graph corpus, gold labels, retrieval metrics, and tool semantics remain fixed. The only manipulated object is the controller context and the logging attached to each action. This lets us attribute changes in ranking to the behavior of the adaptive policy rather than to a different retriever, different corpus, or different evaluator.

3.6. What the trace can and cannot do

The trace hint is not an oracle. It does not know whether a retrieved node is correct, and it does not inspect the gold set. It only summarizes observable symptoms. For example, if the controller repeats a global search with nearly identical arguments, the hint can recommend changing the anchor or stopping repeated exploration. If a neighborhood call returns no candidates, the hint can recommend relaxing the relation/type constraint or returning to global search. This keeps TraceBound close to deployment settings where a retrieval system can observe its own trajectory but cannot observe the correct answer during inference.

3.7. State representation and action view

The protocol can be written as a small partially observed control problem. At step t, the controller has a hidden retrieval belief induced by the prompt and an explicit trace state

$$s_t = (g_t, n_t, z_t, r_t, b_t, c_t),$$

where g_t counts global searches, n_t counts neighborhood explorations, z_t counts zero-result calls, r_t counts repeated calls, b_t is remaining budget, and c_t summarizes the current candidate set. The controller chooses an action $a_t \in \{\text{SEARCH}, \text{EXPLORE}, \text{ANSWER}\}$. TraceBound changes the textual representation of s_t exposed to the controller, while the evaluator observes only the final ANSWER action. This view is useful because it makes clear why static prompts are

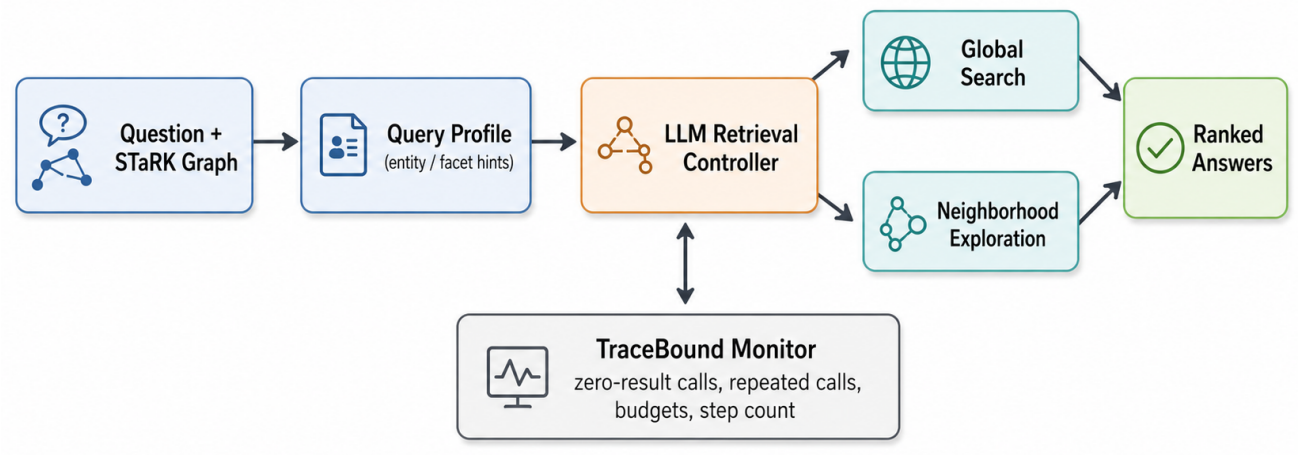

Figure 1. **TraceBound workflow.** The input query and graph are passed to an ARK-style controller. TraceBound adds a query-profile block before retrieval and a monitor that summarizes zero-result calls, repeated calls, budget use, and step count during retrieval. The graph tools and final ranking evaluator are unchanged. The ranked answers produce Hit@1, Hit@5, Recall@20, and MRR, while the monitor produces trace diagnostics used only for analysis.

limited: the important object is the conditional distribution over actions, not the mere presence of a trace string.

4. Experimental Setup

Benchmark and metrics. We evaluate on STaRK PRIME, MAG, and Amazon. These graphs cover biomedical entities, academic papers, and products, respectively. We report the standard retrieval metrics Hit@1, Hit@5, Recall@20, and MRR. Hit@1 and Hit@5 measure whether at least one correct answer appears in the first one or five ranks. Recall@20 measures the fraction of gold answers recovered in the first twenty ranks. MRR measures the reciprocal rank of the first correct answer and is sensitive to whether the correct node moves near the top of the list.

Models and serving. All experiments use open-weight Qwen3 controllers (Yang et al., 2025) served through vLLM. Validation ablations use Qwen3-14B. Held-out subset and constrained-budget experiments use Qwen3-30B-A3B-Instruct-2507 with tensor parallel serving. This local serving choice is reported for reproducibility and cost transparency; the research question is about adaptive retrieval behavior, not about a particular deployment constraint.

Evaluation protocol. The validation ablation uses 100 examples per graph and compares four controller variants under two maximum interaction budgets: 12 and 16. The held-out subset uses the stronger controller and compares the unconditioned agent against TraceBound on PRIME, MAG, and Amazon. Because long-running agent jobs can complete slightly different numbers of examples per condition, we report aggregate results and shared-ID paired results. Paired analysis is the most conservative comparison as it keeps only query IDs present in both conditions and computes TraceBound-minus-unconditioned deltas on the same examples.

Constrained-budget probe. To test whether degradation is caused mainly by excessive interaction length, we run a constrained-budget version of TraceBound with a lower step budget and tighter caps on global and neighborhood calls. This probe is not a separate model; it is a controlled change to the same profile/trace protocol. Its purpose is to distinguish "the agent thinks too long" from "the agent chooses the wrong actions."

Implementation. The implementation overlays TraceBound on an ARK-style code path where graph tools, answer formatting, and metric scripts are preserved, while prompts, trace logging, and run wrappers are extended. We avoid training or test-set tuning since the contribution is evaluated as a controller protocol over fixed tools and fixed metrics.

Naming convention. We avoid shorthand labels in the analysis. "Unconditioned" denotes the original controller without profile or trace hints. "Profile-only" and "Trace-only" are single-component ablations. "TraceBound" denotes the combined profile-plus-trace controller and its instrumentation. A label such as "Unconditioned, 12 steps" therefore means that the same controller is run with a maximum of 12 action-observation iterations; it does not refer to a separate model checkpoint.

Evidence structure. The reported tables are produced from JSON trajectory logs and metric summaries. Each per-query log contains the final ranked list, tool calls, ob-

Table 2. **Validation ablation averaged over PRIME, MAG, and Amazon.** The unconditioned agent is strongest, while the combined TraceBound agent loses both rank precision and recall.

Controller variant	Steps	Hit@1	Hit@5	Recall@20	MRR
Unconditioned agent	12	**.3333**	**.5100**	**.4252**	**.4087**
	16	.3233	.5067	.4192	.4058
Profile-only agent	12	.2833	.4533	.3845	.3511
	16	.2533	.4700	.3848	.3473
Trace-only agent	12	.2333	.3767	.2840	.2944
	16	.2733	.4167	.3195	.3307
TraceBound agent	12	.2579	.3951	.3036	.3141
	16	.2567	.3900	.2998	.3103

servations, and derived counters. The package includes the table summaries and paired-diagnostic outputs used to construct the paper. The raw traces make it possible to audit whether a score difference arises from a ranking change, a missing completion, or a trajectory-level failure such as a repeated call.

5. Results

The central empirical finding is consistent across model scale, graph domain, and comparison method - profile and trace conditioning make trajectories easier to analyze, but they do not improve final ranking in this implementation.

5.1. Validation ablation: profile and trace conditioning hurt MRR

Table 2 reports the Qwen3-14B validation ablation averaged across PRIME, MAG, and Amazon. The "Steps" column is the maximum number of action-observation iterations allowed before final-answer emission. The metric columns are averages across the three graphs. The unconditioned agent with a 12-step budget obtains the best average MRR (.4087), while the complete TraceBound agent obtains .3141 at 12 steps and .3103 at 16 steps. The profile-only and trace-only variants also underperform the unconditioned agent, showing that the degradation is not caused by only one component.

5.2. Held-out subset: the direction persists with a stronger controller

Table 3 reports the Qwen3-30B-A3B held-out subset. The "n" column is the number of completed examples in that graph and condition. The pattern is again negative for TraceBound. On MAG, the most severe case, TraceBound reduces MRR from .7711 to .4974. PRIME and Amazon have smaller but still negative changes. This matters because MAG is the graph where relation-rich academic queries can strongly reward precise anchoring; misdirected adaptive exploration can therefore move the controller toward plausible

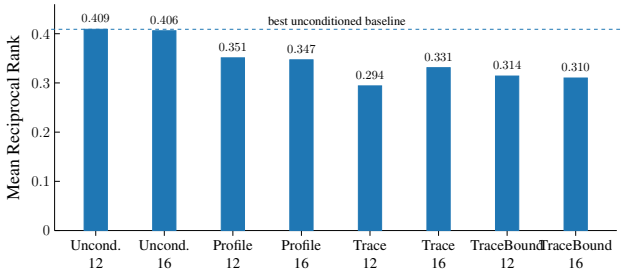

Figure 2. **Validation MRR by controller variant and step budget.** The dashed horizontal reference line marks the best unconditioned baseline. All profile/trace variants fall below that line, so the intervention improves neither the 12-step nor the 16-step validation setting.

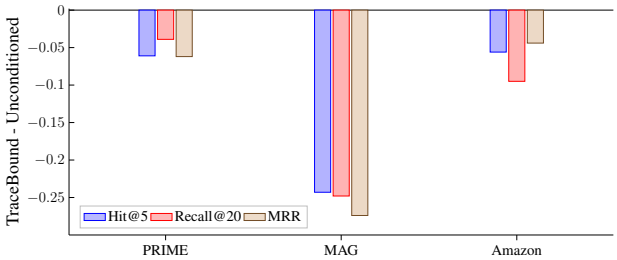

Figure 3. **Held-out metric deltas.** Each grouped bar is TraceBound minus unconditioned on the held-out subset. Bars below zero mean the profile/trace intervention is worse. The three bars per graph correspond to Hit@5, Recall@20, and MRR; all are negative across PRIME, MAG, and Amazon.

but wrong neighborhoods.

5.3. Paired comparison: the result is not an artifact of completed-count differences

Table 4 keeps only shared query IDs available in both unconditioned and TraceBound outputs. "Paired n" is the number of shared examples. Each delta is TraceBound minus unconditioned on the same queries. The paired comparison confirms the aggregate result: all reported deltas are negative. MAG again shows the largest effect, with a -.2711 paired MRR delta.

5.4. Effect-size summary

Across the three held-out graphs, the average paired MRR delta is negative. The magnitude is not uniform. MAG accounts for the largest drop, PRIME is moderate, and Amazon is smaller but still negative. This heterogeneity is scientifically useful: it shows that adaptive failure depends on graph structure and query type. A controller that can recover on product queries may still fail badly on academic graph queries where relation neighborhoods are dense and plausible distractors are common.

Table 3. **Held-out subset with Qwen3-30B-A3B.** Each row is one graph-condition cell. "n" gives completed examples. The TraceBound rows underperform the corresponding unconditioned rows on every graph.

Graph	Controller	n	Hit@1	Hit@5	Recall@20	MRR
PRIME	Unconditioned	290	.2966	.4207	.3681	.3499
	TraceBound	300	.2400	.3600	.3297	.2880
MAG	Unconditioned	299	.7224	.8328	.7483	.7711
	TraceBound	300	.4233	.5900	.5006	.4974
Amazon	Unconditioned	296	.5439	.7128	.4557	.6152
	TraceBound	300	.5033	.6567	.3607	.5714

Table 4. **Shared-ID paired diagnostics.** Rows compare TraceBound and the unconditioned agent on identical query IDs. $\Delta Hit@5$, $\Delta Recall@20$, and ΔMRR are TraceBound-minus-unconditioned. Negative values indicate that TraceBound ranks correct answers lower or retrieves fewer gold answers.

Graph	Paired n	$\Delta Hit@5$	$\Delta Recall@20$	ΔMRR
PRIME	291	-.0653	-.0421	-.0653
MAG	300	-.2400	-.2452	-.2711
Amazon	297	-.0539	-.0911	-.0426

Table 5. **Trace counters for TraceBound on held-out paired examples.** All entries are per-query averages. Higher values are not inherently better, they only indicate more interaction budget consumed before final ranking.

Graph	Global	Neigh.	Zero	Repeat	Events	Steps
PRIME	4.1478	5.7010	4.6151	1.6564	11.8179	8.2577
MAG	2.4133	5.0200	2.2767	0.6633	9.8333	5.8667
Amazon	2.4040	5.4815	3.9293	0.6364	10.3131	6.1111

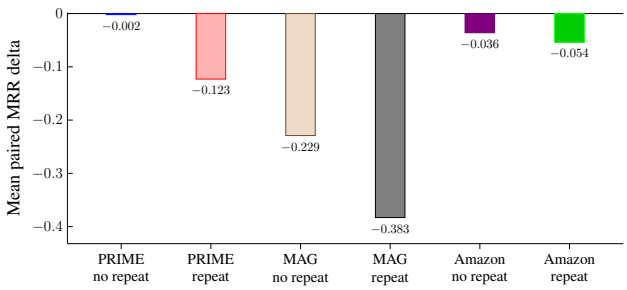

Figure 4. **Conditional rank loss by repeated-call bucket.** Each bar is the mean paired MRR delta, TraceBound minus unconditioned. Here, "repeat" means at least one repeated or near-duplicate call occurred. Repetition predicts a larger loss on PRIME and MAG, and does not help on Amazon either.

6. Discussion

6.1. Trace metrics identify the mechanism

Table 5 reports process counters for the TraceBound trajectories in the held-out paired set. The columns should be read as follows: "Global" is the average number of broad graph-search calls, "Neigh." is the average number of one-hop neighborhood exploration calls, "Zero" is the average number of tool calls returning no useful candidate, "Repeat" is the average number of repeated or near-duplicate calls, "Events" is the average number of logged trace events, and "Steps" is the average controller step count. These counters reveal that TraceBound often spends substantial budget on exploration that does not translate into better ranking.

The repeated-call counter is especially informative. Figure 4 conditions paired MRR deltas on whether a TraceBound trajectory contains any repeated call. Each bar is the mean TraceBound-minus-unconditioned MRR delta for a graph and bucket. PRIME is almost neutral when no repeated call occurs (-.0023), but becomes substantially worse when repetition appears (-.1228). MAG is negative in both buckets, yet repetition makes the degradation larger. The result suggests that repetition is not only an efficiency problem; it is a symptom of a controller that has lost the evidence trail.

6.2. A tighter budget is not sufficient

The constrained-budget probe asks whether TraceBound fails simply because it explores too long. Table 11 lowers the interaction budget and caps the number of global and neighborhood calls. The "Steps" column confirms that the probe shortens trajectories, but the rank metrics remain weak. This separates two hypotheses. If the main failure were only excessive length, tighter budgets should recover most of the unconditioned performance. Instead, the controller remains inaccurate, indicating that the problem is action selection for when to anchor, when to traverse, when to relax, and when to stop.

6.3. Why the result matters

The important finding is not simply that a prompt-level intervention failed. It is that the failure is structured and measurable. TraceBound changes the retrieval trajectory in ways that can be audited by showing how repeated calls indicate stagnation, zero-result calls indicate failed anchoring or traversal, and high exploration counts indicate budget spent without rank improvement. These are exactly the kinds of failures that final retrieval metrics can hide. This matters for adaptive graph retrieval because errors compound over the trajectory. A wrong early anchor can send the controller into a plausible but irrelevant neighborhood. A trace hint can then make the system more active around that bad anchor rather than causing a clean reset. In this setting, more explicit reasoning does not automatically produce better retrieval, instead it only creates more state for the controller to manage. The practical lesson is that profile and trace information should not be treated as globally useful context. They should be inputs to a calibrated action policy that decides when to search broadly, traverse locally, relax constraints, rerank, stop, or restart. TraceBound is therefore useful not because it improves the controller directly, but

because it exposes the failure signals that a better controller must learn to act on.

6.4. Failure taxonomy and corrective policy targets

The trace counters support a taxonomy of operational failures. Table 6 lists the failure class, the observable trigger, its likely ranking effect, and the policy correction suggested by the result. The final column is intentionally framed as a policy target rather than a prompt instruction. The experiments show that simply telling the controller about a failure symptom is insufficient; the system must learn when the symptom should change the next retrieval action.

6.5. Design implications

The study suggests three concrete design principles. First, a profile should be treated as an uncertain signal, not as privileged context. A controller should be able to ignore it when graph constraints are sufficient. Second, trace feedback should be gated by failure confidence. Repetition and empty calls are useful symptoms, but reacting too early can interrupt a correct path. Third, stopping should be a learned action. A system that only searches or explores will keep consuming budget even after the marginal value of another call is low.

6.6. Implications for personalization

The query profile in TraceBound is not a real user model, but it plays the same architectural role as a user profile would play in a personalized retriever - compact context injected before evidence is fetched. The negative result therefore gives a useful warning. Personalization should not be globally concatenated to every retrieval decision. Instead, the controller should learn a conditional policy which uses the profile when it helps disambiguate intent or rank candidates, ignores it when graph constraints dominate, and backtracks when profile-guided expansion enters an empty branch. In other words, personalization is not only a representation problem, it is a control problem over retrieval actions.

7. Limitations

Our evidence is specific to an ARK-style two-tool interface, STaRK graphs, and open-weight Qwen3 controllers served locally. We evaluate query-derived profiles rather than real user memories, so the experiments should not be read as a complete personalized-retrieval benchmark. The held-out evaluation is a subset rather than the full test split, and the constrained-budget probe is diagnostic rather than a tuned production system. These constraints bound our claim that TraceBound shows that a plausible profile/trace-conditioned controller can degrade adaptive graph retrieval and that trace diagnostics explain the degradation. It, however, does not prove that all adaptive or personalized retrieval interventions will fail.

8. Future work

The next step is to replace prompt-level intervention with policy learning. The traces collected here define the state variables: candidate coverage, repeated calls, zero-result calls, relation coverage, budget use, and rank movement. A learned controller can use these variables to choose among global search, neighborhood exploration, relation relaxation, type tightening, reranking, and stopping. Personalization should enter this policy selectively. For example, a user profile may be useful for query anchoring or final reranking but harmful during relation expansion if it distracts from graph constraints. A validation-trained router that decides when to activate profile conditioning is a natural bridge from this diagnostic study to robust personalized retrieval.

A concrete policy-learning formulation. A direct extension is to train a small controller or router over the trace state rather than rely on unconditional textual hints. The training instance would consist of the query profile, current candidate summary, trace counters, and the next action that improves a validation objective. The action space can remain small: broad search, neighborhood exploration, relation relaxation, type tightening, rerank, answer, or restart. The reward can combine rank movement, recall gain, and budget cost. This formulation also makes personalization precise. A user state is not appended everywhere; it becomes one feature available to the policy, and the policy is rewarded only when using that feature improves retrieval.

9. Conclusion

TraceBound turns the common intuition that "more explicit thinking before retrieval should help an adaptive KG retriever" into a measurable experiment. Under our controlled implementation, it does not. The unconditioned agent remains stronger on validation, held-out subsets, and paired shared-ID comparisons. The value of the study is that the reason becomes visible in the fact that repeated calls, zero-result calls, and misallocated exploration budget predict rank loss. This shifts the design target for personalized reasoned retrieval as the central problem is no longer how to add more profile text to the prompt, but how to learn a calibrated retrieval policy that knows when profile and trace information should influence the next action.

Impact Statement

This work studies reliability failures in retrieval agents and is intended to improve transparent evaluation before such

agents are deployed in personalized or private-data settings. It does not introduce new user data or new automated decision pipelines. The main practical risk is overgeneralization as our experiments only show that one plausible profile- and trace-conditioned controller can degrade retrieval quality under a specific open-weight configuration, not that adaptive retrieval or personalization is intrinsically harmful. Trace reporting like these can help researchers identify brittle retrieval policies before they are embedded in user-facing systems.

References

Asai, A., Wu, Z., Wang, Y., Sil, A., and Hajishirzi, H. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *International Conference on Learning Representations*, 2024.

Boer, D., Roth, S., and Kramer, S. Autofocus retrieval: An effective pipeline for multi-hop question answering with semi-structured knowledge. *Transactions on Machine Learning Research*, 2026. arXiv:2505.09246.

Clemedtson, A. and Shi, J. GraphRAFT: Retrieval augmented fine-tuning for knowledge graphs on graph databases. *arXiv preprint arXiv:2504.05478*, 2025.

Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., and Yih, W.-t. Dense passage retrieval for open-domain question answering. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2020.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Kuttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., and Kiela, D. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, 2020.

Liu, X., Yu, H., Zhang, H., Xu, Y., Lei, X., Lai, H., Gu, Y., Ding, H., Men, K., Yang, K., et al. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations*, 2024.

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., et al. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, 2023.

Polonuer, J., Vittor, L., Arango, I., Noori, A., Clifton, D. A., Del Corro, L., and Zitnik, M. Autonomous knowledge graph exploration with adaptive breadth-depth retrieval. *arXiv preprint arXiv:2601.13969*, 2026.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., and Scialom, T. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, 2023.

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.

Sun, J., Xu, C., Tang, L., Wang, S., Lin, C., Gong, Y., Ni, L., Shum, H.-Y., and Guo, J. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. *arXiv preprint arXiv:2307.07697*, 2023.

Tan, J., Yang, L., Liu, Z., Liu, Z., Murthy, R., Awalgaonkar, T., Zhang, J., Yao, W., Zhu, M., Kokane, S., et al. Personabench: Evaluating AI models on understanding personal information through accessing synthetic private user data. In *Findings of the Association for Computational Linguistics*, 2025.

Trivedi, H., Balasubramanian, N., Khot, T., and Sabharwal, A. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the Association for Computational Linguistics*, 2023.

Wu, S., Zhao, S., Yasunaga, M., Huang, K., Cao, K., Huang, Q., Ioannidis, V. N., Subbian, K., Zou, J., and Leskovec, J. STaRK: Benchmarking LLM retrieval on textual and relational knowledge bases. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2024.

Xie, T., Zhang, D., Chen, J., Li, X., Zhao, S., Cao, R., Hua, T. J., Cheng, Z., Shin, D., Lei, F., et al. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*, 2024.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. doi: 10.48550/arXiv.2505.09388. URL https://arxiv.org/abs/2505.09388.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Zhang, Y., Jia, P., Xu, D., Wen, Y., Li, X., Wang, Y., Zhang, W., Li, X., Gan, W., Guo, H., Liu, Y., and Zhao, X. Personalize before retrieve: LLM-based personalized query expansion for user-centric retrieval. *arXiv preprint arXiv:2510.08935*, 2025.

Appendix

Table 6. **Failure taxonomy induced by trace diagnostics.** "Trigger" is computed from the trajectory. "Effect" describes how the failure can damage ranking. "Policy target" states what a learned controller should decide, rather than what a static prompt should always instruct.

Failure class	Observable trigger	Policy target
Wrong anchor	many neighborhoods after weak global hit	re-anchor before deeper traversal
Empty branch	zero-result neighborhood calls	relax relation/type or backtrack
Looping	repeated global/neighborhood call	suppress duplicate action; stop or reset
Over-broad search	many candidates, low rank movement	tighten answer type or facet
Premature repair	trace hint changes a good path	delay intervention until failure confidence is high
Budget drift	many steps without candidate gain	stop, rerank, or restart from a new anchor

Table 7. **From trace diagnostics to policy-learning targets.** Each row maps an observable state signal to a decision that a future controller should learn. The table is not a new result; it distills the empirical failure analysis into implementable policy targets for personalized reasoned retrieval.

State signal	Risk exposed by this paper	Decision to learn
Profile conflicts with graph type	wrong anchor	downweight profile; search by type
Repeated call appears	loop or stagnation	suppress duplicate; reset or answer
Zero-result branch appears	empty traversal	relax relation or backtrack
Candidate set grows without rank gain	over-broad evidence	tighten facets or rerank
Budget nearly exhausted	late noisy repair	stop or return best supported list

Table 8. **Implementation invariants.** The left column lists components that remain unchanged across all controller variants. The right column lists the only components modified by TraceBound. This separation is important because the work studies controller behavior, not a new graph index or new metric.

Component	Fixed across variants	Added by TraceBound
Graph data	Same STaRK SKB and node IDs	none
Gold labels	Same answer sets	none
Tools	Global search; neighborhood explore	no new tool
Evaluator	Hit@1, Hit@5, Recall@20, MRR	no metric change
Controller context	Original query and observations	profile block; trace hint
Logs	final answer list	trajectory counters

A. Graph-Level Validation Details

Table 13 reports selected graph-level validation cells. The table is included to make the averages in the main text auditable. "Global" and "Zero" are trace counters and are zero for conditions without trace instrumentation.

B. Held-Out Delta Tables

Table 14 gives the held-out metric deltas used in Figure 3. Each value is TraceBound minus the unconditioned agent; negative values indicate lower retrieval quality after profile/trace conditioning.

C. Win, Tie, and Loss Counts

Table 15 complements the paired deltas by counting whether TraceBound wins, ties, or loses against the unconditioned agent on each shared query.

D. Trace Schema

Each trajectory log contains the query identifier, graph name, controller variant, final answer list, tool-call sequence, observation summaries, and aggregate counters. The counters used in the main paper are computed from the raw sequence after the run; they are not hand-labeled. This schema is sufficient to reproduce per-query paired deltas, conditional repeated-call analysis, and budget-consumption summaries.

Table 9. **Profile and trace fields exposed to the controller.** These fields are derived from the query and trajectory, not from gold labels. "Purpose" states the intended control role. The results show that exposing these fields is not sufficient; the controller must also learn when to trust or ignore them.

Field group	Example fields	Intended control role
Answer profile	entity type; query facets	decide broad search anchors
Relation hints	relation words; type constraints	choose neighborhood expansion
Trace counters	zero calls; repeats; budget	detect loops and empty branches
Candidate summary	candidate count; recent movement	stop, rerank, or restart

Table 10. **Qualitative reading of the held-out effects.** This table translates the paired numeric deltas into the retrieval behavior they indicate. It is not a separate metric, it is more of a guide to interpreting Tables 3 and 4.

Graph	Observed pattern	Interpretation
PRIME	moderate MRR loss; high zero calls	biomedical anchoring often enters empty or wrong neighborhoods
MAG	largest MRR and recall loss	wrong paper/entity anchors compound through dense relations
Amazon	smaller MRR loss; recall loss remains	top plausible products survive, broader candidate coverage degrades

E. Prompt Skeletons

The exact prompts are implementation artifacts, but the following skeleton captures the logical fields used by TraceBound. The profile block is created before retrieval. The trace block is refreshed after each action and is intentionally limited to observable trajectory facts.

```
[QUERY]
{natural_language_question}

[QUERY PROFILE]
- predicted answer type: {entity_type}
- key facets: {facet_1, facet_2, ...}
- relation hints: {relation_words}
- retrieval risks: {ambiguity_or_missing_filter}

[TRACE SUMMARY]
- global searches used: {g_t}
- neighborhood calls used: {n_t}
- zero-result calls: {z_t}
- repeated calls: {r_t}
- remaining budget: {b_t}
- current candidate summary: {c_t}

[CONTROL INSTRUCTION]
Choose exactly one: Search, Explore, Answer, or Backtrack.
Do not repeat an earlier call unless it is necessary.
Prefer answering when additional calls are unlikely to improve rank.
```

F. Metric Definitions

For completeness, this section states the metric interpretation used throughout the paper. Hit@1 is one when a correct answer appears at rank one. Hit@5 is one when a correct answer appears in the top five. Recall@20 is the fraction of gold answers retrieved in the first twenty positions. MRR is the reciprocal rank of the first correct answer. For paired deltas, each metric is computed for TraceBound and for the unconditioned agent on the same query ID, and then subtracted as TraceBound minus unconditioned.

Table 11. **Constrained-budget TraceBound probe.** The probe uses a lower step budget and stricter call caps. "Steps" is the average number of controller iterations actually used. The probe reduces interaction length but does not recover the unconditioned ranking quality, so the failure is not explained only by excessive trajectory length.

Graph	n	Hit@1	Hit@5	Recall@20	MRR	Steps
PRIME	150	.1200	.2067	.1857	.1556	5.493
MAG	150	.2800	.4867	.4020	.3727	4.860
Amazon	150	.4133	.5000	.2879	.4511	4.927

Table 12. **Recommended reporting checklist for adaptive retrieval.** Standard rank metrics remain necessary, but they should be paired with process metrics that reveal why an agent succeeded or failed. This table summarizes the minimal diagnostics that made the TraceBound result interpretable.

Reported quantity	What it measures	Failure exposed
Paired MRR delta	rank movement on same IDs	hidden aggregate artifacts
Zero-result calls	empty branch frequency	bad anchoring or relation mismatch
Repeated calls	loop tendency	stalled controller state
Search/explore counts	breadth-depth allocation	budget misallocation
Constrained-budget score	robustness to shorter traces	length vs. action-quality failure

Table 13. Graph-level validation details for representative controller variants.

Graph	Controller	Steps	n	Hit@1	Hit@5	Recall@20	MRR	Global	Zero
PRIME	Uncond.	12	100	.2100	.3500	.3489	.2736	.000	.000
	Profile	16	100	.1300	.3600	.3545	.2350	.000	.000
	Trace	16	100	.1500	.2600	.2262	.1957	2.260	1.620
	TraceBound	16	100	.1300	.2000	.1945	.1619	2.130	1.860
MAG	Uncond.	16	100	.3900	.5700	.5127	.4683	.000	.000
	Trace	16	100	.2900	.4800	.4196	.3611	1.770	1.240
	TraceBound	16	100	.2500	.4400	.3835	.3234	1.870	1.350
Amazon	Uncond.	16	100	.4200	.6300	.4338	.5165	.000	.000
	Profile	16	100	.4000	.6100	.4149	.4930	.000	.000
	TraceBound	16	100	.3900	.5300	.3214	.4456	1.710	2.230

Table 14. Held-out aggregate deltas.

Graph	$\Delta Hit@1$	$\Delta Hit@5$	$\Delta Recall@20$	ΔMRR
PRIME	-.0566	-.0607	-.0384	-.0619
MAG	-.2991	-.2428	-.2477	-.2737
Amazon	-.0406	-.0561	-.0950	-.0438

Table 15. Paired win/tie/loss counts for TraceBound versus the unconditioned agent. Counts are computed on the same shared logged query IDs as Table 4.

Graph	MRR win	MRR tie	MRR loss	H1 win	H1 tie	H1 loss
PRIME	38	192	61	21	232	38
MAG	29	138	133	21	169	110
Amazon	43	190	64	25	235	37