AG-RAG: ROBUST AGENTIC GRAPH RAG THROUGH STRESS-AWARE SELF-IMPROVEMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph retrieval-augmented generation (Graph RAG) organizes external knowledge into entities, relations, and supporting documents to improve LLM reasoning. However, existing Graph RAG systems are typically optimized and evaluated on clean queries, where their advantages over text retrieval can be modest. We argue that graph structure can be especially valuable when retrieval fails. We introduce AG-RAG, an agentic Graph RAG framework designed for such failures. AG-RAG exposes a corpus-derived knowledge graph through MCP tools for fuzzy entity resolution, graph traversal, and provenance-bound document retrieval, together with stress-aware self-improvement that optimizes tool policies, retrieval budgets, ranking, and instructions while rejecting changes that sacrifice robustness. We evaluate three retrieval failures: misspelled entities, unavailable source text, and adversarially injected passages. With the same answering model, AG-RAG remains competitive with HippoRAG 2 and LinearRAG on clean HotpotQA, 2WikiMultiHopQA, and MuSiQue, while substantially outperforming both under stress. On 2Wiki-Extra, a set of 2WikiMultiHopQA questions never used during development, AG-RAG achieves only 2.2% poisoning attack success, compared with 57.2% for HippoRAG 2 and 48.0% for LinearRAG. These results show that graph structure provides complementary robustness against retrieval failures and suggest that self-improving retrieval agents should optimize capability subject to explicit stress-aware constraints.

1 INTRODUCTION

Large language models (LLMs) can reason over a prompt, but answering questions about external knowledge requires access to evidence. Retrieval-augmented generation (RAG) provides that evidence by searching a text collection and placing retrieved passages in the model's context (Lewis et al., 2020). *Graph RAG* adds a graph of entities, relations, and their source documents, giving retrieval a structured route through related evidence (Edge et al., 2024; Gutiérrez et al., 2025; Zhuang et al., 2025). An LLM *agent* can go further and choose which retrieval step to take next, using each observation to decide what to do (Yao et al., 2023).

Graph RAG is usually built and compared on clean benchmarks, where it does not consistently beat text RAG (Xiang et al., 2025). Yet a graph has properties that matter most when retrieval goes wrong. Entity names can suggest a correction for a misspelled query; edges can keep a relation when its source page disappears; and document links record which sources were present when the graph was built. Typos, missing pages, and poisoned passages are all documented retrieval problems (Cho et al., 2024; Pew Research Center, 2024; Zou et al., 2025). Therefore, we argue that robustness is where a graph earns its cost.

Consider a question asking which of two films was released more recently. As shown in Figure 1, when both titles are misspelled, a text agent fails to find them, while a graph agent repairs the names and retrieves their release years. In a separate poisoned rollout, newly indexed passages claim the older film was released later. An agent that reads free text repeats that claim; an agent that reads only graph-linked documents answers correctly.

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

Typos: "Which film was released more recently, 3 Nfaas Likaas or The Barvest Way?" (gold: 3 Nafas Likas)
Text agent: six searches for the misspelled titles → cannot determine ✗
AG-RAG: suggestions `3 nafas likas` and `the bravest way` → linked documents (2014 and 1918) → **3 Nafas Likas** ✓

Poisoning: Two injected passages claim *The Bravest Way* was released in 2015.
Text agent, and an agent that added text search under clean-only tuning → The Bravest Way ✗
AG-RAG (graph-linked reading) → **3 Nafas Likas** ✓

Figure 1: **Two retrieval failures on the same underlying question.** Separate typo and poisoning rollouts from the 2Wiki-Extra test illustrate fuzzy name repair and exposure to injected passages. The poisoned rollout uses the correctly spelled names.

In this work, we propose AG-RAG, an agentic Graph RAG framework to deal with these issues. AG-RAG has two parts. Its *MCP server* makes a corpus-derived graph available to an existing agent through standard tool calls (Anthropic, 2024), with three graph-native safeguards: fuzzy entity resolution, relation edges that outlive their passages, and *provenance-bound reading*, where the agent reads a document only through an entity link in the graph. Tools alone do not tell the agent which to use, how much evidence to request, or when to fall back to text. The *self-improvement harness* tunes that tool policy without retraining the model. We use self-improvement in a narrow sense: a bounded, iterative search that proposes typed changes to tools, budgets, ranking, and instructions, tests them, and builds on accepted ones.

The harness must decide which changes to keep, and here Graph RAG raises a specific risk. On clean questions, adding free-text search often retrieves useful passages, so a clean score rewards it. But the same tool reads passages that bypass the graph, including newly injected poison. Self-improvement scored on clean questions therefore can erode the robustness the graph provides. AG-RAG keeps a change only if it also holds up under typos, missing text, and poisoning.

Overall, the main contributions of this work are as follows:

1. *A robust agentic Graph RAG system.* AG-RAG exposes a graph as MCP tools with fuzzy name repair, batched neighbour expansion, and provenance-bound document reading. Section 3 describes the system.
2. *Stress-aware self-improvement.* A typed harness search keeps an edit only if it passes clean and stressed checks. It raises clean F1 from 0.625 to 0.665 and cuts turns from 4.66 to 3.94 without losing robustness. Sections 4 and 5.5 present the method and results.
3. *Comparison with Graph RAG systems.* With the same answering model, AG-RAG is competitive with HippoRAG 2 and LinearRAG on three clean benchmarks and far more robust under stress; ablations assign each gain to a graph component. Sections 5.3 and 5.4 report these results.
4. *Why clean tuning is not enough.* Under a clean-only rule, both an LLM proposer and matched-budget random search add free-text search and raise poisoning success, as shown in Section 5.6.

2 Related work

Graph RAG. GraphRAG Edge et al. (2024) was first proposed to turn a corpus into an entity graph, partitioning it into communities, and using community summaries to support global, query-focused reasoning over large text collections. RAPTOR retrieves from a hierarchy of summaries (Sarthi et al., 2024); HippoRAG and HippoRAG 2 rank evidence with personalized PageRank (Gutiérrez et al., 2024; 2025); LinearRAG builds a relation-free entity graph that scales to large corpora (Zhuang et al., 2025); GFM-RAG learns retrieval over a corpus-derived graph and supplies our extraction pipeline (Luo et al., 2025). Graph Chain-of-Thought, Think-on-Graph, and G-Retriever let the model traverse or select subgraphs (Jin et al., 2024; Ma et al., 2024; He et al., 2024), and Graph-R1 and GraphRAG-R1 learn traversal with reinforcement learning (Luo et al., 2026; Yu et al., 2026). These systems are evaluated on clean questions, where graph retrieval is not uniformly better than text retrieval (Xiang et al., 2025). AG-RAG instead exposes graph operations as tools an existing agent can call, and designs and tunes them for retrieval failures.

Tool use and self-improving agents. ReAct interleaves reasoning with actions and IRCoT interleaves it with retrieval (Yao et al., 2023; Trivedi et al., 2023), so the model can plan, retrieve information, update its state, and continue reasoning based on observations from the environment. MCP further standardizes tool calls (Anthropic, 2024). OPRO and DSPy optimize prompts and pipelines against a score (Yang et al., 2024; Khattab et al., 2024), and ADAS and the Darwin Gödel Machine search over agent designs and code (Hu et al., 2025; Zhang et al., 2025). Two related directions are becoming important. *Agent harnesses*, the runtime that supplies an agent's tools, state, permissions, and loop, are now a design target of their own; DeepSeek Harness, for example, makes the model adapter, tool registry, and agent loop swappable plugins (DeepSeek-AI, 2026). *Recursive self-improvement*, where a system improves the procedure that improves it, has been shown for code generation (Zelikman et al., 2024) and agent code (Zhang et al., 2025). Both make the choice of what an agent is optimized for more consequential. Our loop is narrower: a constrained configuration search over a typed grammar. We show that its objective matters (Skalse et al., 2022): a clean score can reward a tool that bypasses the graph.

Retrieval failures. Typos break entity matching and passage retrieval (Cho et al., 2024); PoisonedRAG steers retrievers with answer-targeted passages (Zou et al., 2025); RobustRAG isolates passages before aggregating answers (Xiang et al., 2024). In our main poisoning test, malicious passages enter the text index after graph construction; Appendix K tests poison that enters through graph ingestion.

3 AG-RAG: A GRAPH TOOL SERVICE FOR AGENTS

AG-RAG has two major components. An *MCP server* turns a corpus-derived graph into tools an existing agent can call, as described in Section 3.1. A *harness* fixes which tools the agent may use and how, and a search over harnesses can change that configuration, as described in Section 4.

3.1 GRAPH RETRIEVAL TOOLS

AG-RAG begins with a corpus $\mathcal{D}$ and a text-derived graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, built with the GFM-RAG extraction pipeline (Luo et al., 2025); its statistics are in Appendix D. Edges record relations between entities, and an entity-to-document map links each node to passages that mention it. The graph thus offers two kinds of evidence: relations that survive independently of a source passage, and text reached through an entity link. On each turn the agent may call `get_neighbor_entities` (one entity, optionally one relation), `get_batch_neighbor_entities` (several entities in one call), or `retrieve_documents` (documents linked to the selected entities, ranked by how many of them each document mentions). Batching avoids a separate turn for each branch of a question.

Figure 2 shows how the agent calls these tools during reasoning. With S the resolved entities and $M_{vd} = 1$ when entity v links to document d, `retrieve_documents` ranks by $s(d \mid S) = \sum_{v \in S} M_{vd}$, which favors a passage shared by two branches of a multi-hop question. The harness can instead weight rare entities by inverse document frequency and cap the number or length of returned documents; these options change the evidence budget without changing the graph.

3.2 FUZZY ENTITY RESOLUTION

When a name is not found, the server returns suggestions instead of an empty result. An inverted index of character trigrams keeps at most 5,000 candidates by overlap, which RapidFuzz WRatio reranks (RapidFuzz, n.d.). The agent sees the candidates and chooses which to explore; in Figure 1, misspelled titles yield `3 nafas likas` and `the bravest way`. The prefilter avoids scoring every name on every call; its latency is reported in Appendix E.

3.3 PROVENANCE-BOUND DOCUMENT RETRIEVAL

AG-RAG reads text only through the graph: `retrieve_documents` returns passages linked to graph entities, and a passage gains links only when the graph is built or updated. The graph is therefore a versioned index of which documents the agent may read. A passage that later enters the text collection has no entity link and cannot be returned. Native Graph RAG pipelines such as HippoRAG 2 and LinearRAG instead index every passage added to the corpus. An optional free-text

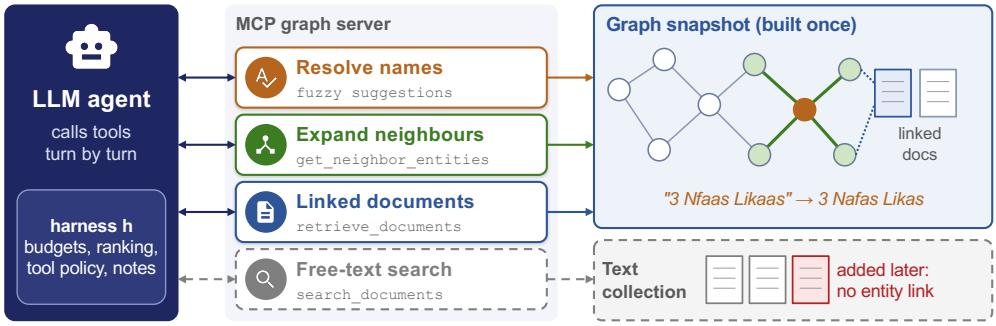

Figure 2: AG-RAG exposes graph retrieval as callable tools. Left: the LLM agent calls tools turn by turn; its harness h sets budgets, ranking, tool policy, and notes. Middle: the MCP graph server offers three graph tools and one optional free-text search tool, drawn dashed. Right: the graph tools read only the graph snapshot.

search tool uses BM25 over the current text collection; the harness decides whether the agent may call it.

This protection comes from *which documents* a tool may reach, not from edges: a BM25 tool restricted to the graph's snapshot would share it. The graph makes the boundary natural, because the same build step that extracts relations defines the snapshot. It holds only while graph updates are vetted; Appendix K tests graph ingestion of malicious text, where it weakens.

3.4 Retrieval Stress Conditions

We call the unmodified question and original text index *clean*. A *stress* is a controlled change to that question or index; all conditions use the same underlying questions. *Typos* adds a character-level error to every entity mention in the question and its supplied hint. The hint is the list of entity mentions an LLM named-entity step extracted from the question; every agent, including the text agent, receives the same hint. *50% text* makes a random half of the documents unreadable, while the graph built earlier is kept. Linked-document retrieval also loses these passages, so only relations stored as graph edges remain. The condition models unavailable sources such as link rot, not intentional deletions, as discussed in Appendix A. *Poisoned* adds two generated passages per question to the text collection under natural titles. They reuse question entities and assert a plausible wrong answer, following PoisonedRAG (Zou et al., 2025); the graph predates this injection. *Attack success rate* (ASR) is the percentage of questions for which the system returns the injected wrong answer. The *stress average* is the mean F1 under the three stresses.

The suite is asymmetric. It contains a condition in which new text is harmful (poisoning) but none in which new text is required, such as a legitimate update whose answer appears only after the graph was built. Under such an update, provenance-bound reading would need a graph update first. Our results therefore describe the security side of a trade-off between trusting new evidence and resisting injected evidence.

4 Stress-aware self-improvement

The harness improves through an iterative propose–evaluate–promote cycle, as Figure 3 shows. A *proposer* is an LLM that reads the current agent's question traces and suggests configuration edits. A *generation* tests those edits and makes one accepted configuration the *incumbent* for the next generation. The answering model and proposer remain fixed. We call this repeated configuration search the *harness loop*. The optional *memory loop* writes supported facts into the graph; *both loops* means that harness search and graph writes operate together. A *promotion rule* decides which harness edit survives. The *stress-aware rule* used by AG-RAG checks clean questions and the three retrieval stresses; the *clean-only rule* checks clean questions only and serves as a control.

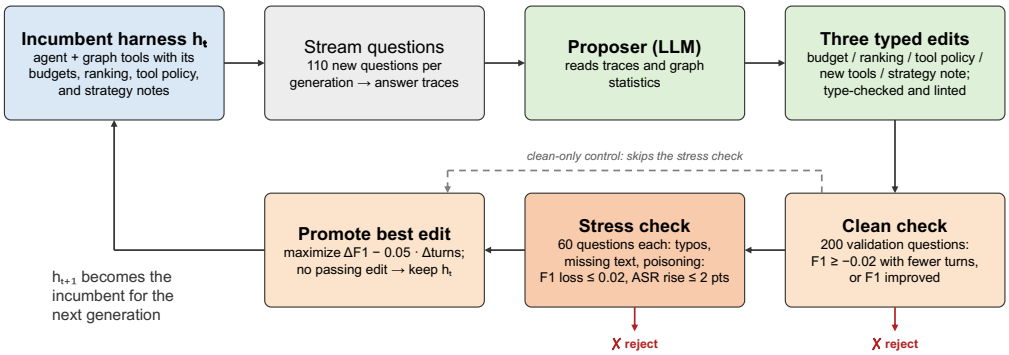

Figure 3: **The stress-aware harness loop.** Top row: in each generation, the incumbent harness h_t answers 110 new stream questions, and an LLM proposer reads the traces and graph statistics and returns three typed edits. Bottom row: each edit must pass a clean check on 200 validation questions and then a stress check on 60 questions per stress (typos, missing text, poisoning); an edit that fails either check is rejected. Among passing edits, the one with the largest ΔF1 $-$ 0.05 Δturns becomes the next incumbent h_{t+1}; if none passes, h_t is kept. The clean-only control (dashed) skips the stress check.

Algorithm 1: Stress-aware promotion rule; the clean-only control omits StressCheck

Input: incumbent h, candidates C, validation split, stress shards
$best \leftarrow h$; $u^* \leftarrow 0$;
foreach $c \in C$ **do**
  if *not* CleanCheck(c, h) **then continue**;
  if *stress-aware* ***and not*** StressCheck(c, h) **then continue**;
  $u \leftarrow \Delta F1(c, h) - 0.05\,\Delta\text{turns}(c, h)$;
  if $u > u^*$ **then** $best \leftarrow c$; $u^* \leftarrow u$;
return $best$

4.1 Harness Search Space

The proposer changes configuration rather than executable code. Its five edit families control budgets, document and neighbour ranking, tool-use policy, optional text search or path finding, and strategy notes of at most 60 words. Type checks keep edits within the grammar; a linter rejects notes that change the answer format or contain entity names or numbers. Table 1 shows the grammar; its discrete fields have about 9,000 settings. Each generation supplies 110 new question traces and graph statistics to the proposer, which returns three edits. For example, reducing linked documents per call and changing their ranking can save turns; adding free-text search can improve clean answers while exposing the agent to newly indexed poison.

4.2 Stress-Aware Promotion Rule

Candidates first face 200 clean validation questions disjoint from the stream. A paired bootstrap check requires either non-inferior F1 (lower confidence bound above -0.02) with fewer turns, or improved F1 (lower bound above zero) with at most 1.25 times as many turns. Passing candidates then face 60 questions for each stress. The rule permits at most a 0.02 F1 loss in each and a two-percentage-point ASR increase. These stress checks use point estimates because intervals at 60 questions would reject nearly every candidate. At this size the F1 part has little power (identical reruns move F1 by 0.015–0.036); of five candidates that passed the clean check but failed the stress check, four failed on ASR and one on F1 alone. The stress shards use the same generators as the test conditions, on different questions. Among passing edits, the rule maximizes clean F1 gain minus 0.05 times the change in turns. The clean-only control omits the stress checks; the other rules are identical. Algorithm 1 summarizes the rule.

Table 1: **Harness grammar.** The discrete part has about 9,000 settings; strategy notes are free text, bounded and linted.

Rule	Field	Values
Budget	max turns	4, 6, 8, 10
	documents per call	1, 2, 3, 5
	neighbours per call	all, 20, 50, 100
	characters per document	all, 400, 800
Ranking	document scorer	entity count, IDF
	neighbour order	none, question overlap
Tool policy	tool-use policy	graph first, balanced, documents early
New tools	text search tool	off, on
	path-finding tool	off, on
Strategy	strategy notes	$\leq$60 words, linted (no answer format, names, numbers)

4.3 Graph Memory and Verification

The graph server supports optional triple writes after answering. In the memory loop studied here, a writer proposes a triple and the span of a retrieved document that supports it. A grounding check confirms the span and entity mentions; an LLM verifier judges support. Each accepted edge retains provenance and can be rolled back. These checks test whether a source says a fact, not whether it is truthful. Memory writes were not subject to the promotion rule; we treat memory as a separate extension in Appendix K.

5 Evaluation

We aim to answer the following research questions: **RQ1:** how does AG-RAG compare with native Graph RAG systems on clean benchmarks and under stress? **RQ2:** which graph component explains each robustness gain? **RQ3:** does stress-aware self-improvement improve the agent, and what happens when it is scored on clean questions only?

5.1 Dataset

Standard test sets (clean comparison). To compare with published Graph RAG systems on clean questions, we use the standard GFM-RAG test sets of HotpotQA (Yang et al., 2018) (944 unique questions), 2Wiki (1,000), and MuSiQue (Trivedi et al., 2022) (1,000).

2Wiki-Extra (main study). Self-improvement keeps whatever scores best on the questions it sees, so the main study needs questions that played no part in building the system. We built *2Wiki-Extra* from the 2WikiMultiHopQA development set (Ho et al., 2020) without the 1,000 questions of the standard 2Wiki test set. Its graph is built from their 54,957 passages with the GFM-RAG extraction pipeline (Luo et al., 2025); Appendix D gives its statistics. Three disjoint splits were fixed before the main study. The harness search reads traces of the *stream* questions and checks candidates on the *validation* questions; it never sees the 500 *test* questions, on which all robustness, ablation, and self-improvement results are reported.

5.2 Baselines

Graph RAG baselines. We run HippoRAG 2 (Gutiérrez et al., 2025) and LinearRAG (Zhuang et al., 2025) with their official code, published retrieval settings, and prompts. A local proxy routes every LLM call to the same answering model used by AG-RAG, and all systems share our scorer. HippoRAG 2 reads the same extracted graph as AG-RAG, so the two differ in retrieval and reading. Appendix G gives the details.

AG-RAG variants. *AG-RAG (static)* is the graph agent before self-improvement; *AG-RAG (self-improved)* uses the harness selected by the stress-aware rule. Controls include a BM25 *text agent* with the same model and loop, a clean-only harness search, systems that add graph memory, and matched-budget random search. Searches run three generations with two proposer seeds. In-house

Table 2: **AG-RAG against native Graph RAG systems, all with the same answering model.** Left: clean F1 on the standard test sets; EM is in Appendix G. Right: 2Wiki-Extra test ($n = 500$) under each stress, with poisoning attack success (ASR, lower is better).

	Standard sets, clean F1			2Wiki-Extra test, F1 and ASR				
System	HotpotQA	2Wiki	MuSiQue	Clean	Typos	50% text	Poisoned	ASR ↓
HippoRAG 2	**75.8**	64.1	48.0	50.1	41.7	27.0	31.9	57.2%
LinearRAG	73.9	**70.9**	46.4	56.7	35.1	28.0	35.0	48.0%
Text agent (BM25)	–	–	–	**69.1**	53.3	44.3	24.2	68.4%
AG-RAG (static)	72.9	66.0	50.3	62.5	59.6	60.4	61.8	2.4%
AG-RAG (self-improved)	72.7	70.5	**52.0**	66.5	**63.0**	**63.2**	**67.2**	**2.2%**

single-shot, PPR, IRCoT-style (Trivedi et al., 2023), and isolate-then-vote baselines are in Appendix F; they are not official implementations, and our isolate-then-vote variant lacks the secure aggregation of RobustRAG (Xiang et al., 2024).

The answering model is `gpt-6-luna` at temperature zero; the proposer uses `gpt-6-sol`. We measure F1 and EM score against gold answers and aliases, agent turns, and poisoning ASR. Evaluation tables and result text report F1 score; the promotion rule above retains the metric's native 0–1 units. Identical reruns agree on 73–87% of answers and we use one test rollout per question, so key differences carry paired 95% bootstrap intervals, listed in Appendix C. Building the 2Wiki-Extra graph (54,957 passages) cost $14.33; per clean question AG-RAG costs about 30% more than the text agent: 7,419 vs 4,808 tokens, or $0.00047 vs $0.00037, with details in Appendix B.

5.3 Comparison with Graph RAG Baselines

Table 2 shows that no system dominates on the clean standard sets. The self-improved AG-RAG leads on MuSiQue, nearly matches LinearRAG on 2Wiki, and trails HippoRAG 2 on HotpotQA by 3.1 points. Part of HippoRAG 2's lower 2Wiki score comes from its reader: with its prompt, our answering model replies "not specified" on 22.4% of questions even though 91.9% of gold documents are in its top five, as shown in Appendix G.

Stress reveals a much larger gap. Relative to their clean scores, the self-improved AG-RAG loses at most 3.5 F1 points; HippoRAG 2 and LinearRAG lose at least 8.4 and 21.6 points, respectively. Its poisoning ASR is also much lower: 2.2%, compared with 57.2% for HippoRAG 2 and 48.0% for LinearRAG. The text agent shows that agency alone does not provide this robustness: although it has the highest clean score, its poisoned F1 falls to 24.2 and its ASR reaches 68.4%.

5.4 Effect of Graph Components

Table 3 links each stress result to one graph component. Removing fuzzy suggestions cuts typo F1 by 29.5 points. Keeping only the entity-to-document tool still limits poison exposure, because newly added documents have no entity links; ASR remains 2.8%. The same ablation, however, cuts missing-text F1 by 30.5 points, because relation edges retain facts from unavailable passages. With intact text, the entity-index-only agent is 3.5–5.8 points better because it avoids the extra turns required for graph traversal.

Thus, each safeguard in Section 3 covers a different failure: fuzzy matching repairs typos, edges retain facts when passages are missing, and provenance-bound reading excludes passages added after graph construction. The last property does not require edges; a text tool restricted to the same snapshot might provide it with higher clean accuracy. An LLM spelling fixer improves the text agent by only 2.9 points on typos, to 56.2, still below AG-RAG at 63.0; a fuzzy text index remains untested.

As Figure 4 shows, the graph advantage over the text agent grows with stress strength. Under poisoning, the graph leads on all four tested data sources, including a 17.2-point lead on MuSiQue. The missing-text advantage is clear on the two pilot sets and on 2Wiki-Extra but uncertain on MuSiQue; the typo result transfers less consistently.

Table 3: **Each graph component covers a different failure.** 2Wiki-Extra test F1 score. Fuzzy suggestions repair typos; retained edges matter when text is absent; provenance-bound reading limits exposure to newly poisoned text.

System	Clean	Typos	50% text	Poisoned	ASR ↓
AG-RAG (static)	62.5	59.6	**60.4**	61.8	2.4%
without fuzzy suggestions	57.5	30.1	–	–	–
without edges (entity index only)	66.4	**63.1**	29.9	**67.6**	2.8%
Text agent	**69.1**	53.3	44.3	24.2	68.4%

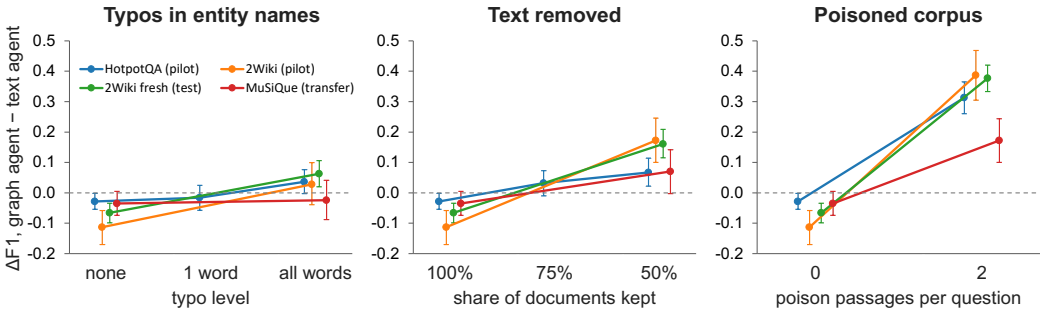

Figure 4: **Static graph versus text retrieval across stress strengths.** ΔF1 of the static graph agent over the text agent; on the plotted 0–1 axis, 0.10 equals 10 F1 points. The typo effect is weaker across data sources than the missing-text and poisoning effects. MuSiQue stress points use 200 questions; its clean point uses 500. Uncertainty calculation is detailed in Appendix B.

5.5 Effect of Stress-Aware Self-Improvement

Table 4 shows a consistent gain from self-improvement: clean F1 rises by 4.0 points and stress-averaged F1 by 3.9, while mean turns fall from 4.66 to 3.94 and ASR remains near 2%. As shown in Table 2, the gain also transfers to the standard sets, improving 2Wiki by 4.5 points and MuSiQue by 1.7 while leaving HotpotQA essentially unchanged. Accepted edits adjust linked-document budgets, ranking, and tool policy without changing the provenance boundary. By contrast, text-search proposals gain 5–7 clean validation points but raise ASR by 12–18 percentage points; the stress check rejects them.

With the same promotion rule and rollout budget, random search produces an improved final harness in one of two seeds; the LLM proposer does so in both. Relative to random search, the proposer gains 2.2 F1 points with missing text and 2.1 under poisoning; its clean and typo advantages are smaller and uncertain.

5.6 Effect of Clean-Only Self-Improvement

The two combined-loop systems differ only in their promotion rule. By generation three, the clean-only rule selects free-text search in both seeds; the stress-aware rule does not. This trade-off is easy to see in Table 4: clean-only tuning gains 2.8 clean F1 points but loses 2.3 points under poisoning, while ASR rises almost sixfold, from 2.8% to 16.6% (+13.8 percentage points, 95% interval $[+11.0, +16.7]$). Across the pilot and main clean-only runs, four of five adopted text search; none of four stress-aware runs did. These are counts from few runs, not rates. Both arms also run the memory loop; a harness-only clean-only LLM arm was not run.

The proposer does not explain the choice. With the same grammar and rollout budget, clean-only random search, which uses no memory, also selected text search in both seeds. As Table 5 shows, its ASR was 13.8%, versus 2.3% under the stress-aware rule. Seven of ten random candidates that passed the clean check added text search; each failed the stress check. On MuSiQue, the frozen clean-only harness reaches 51.2% ASR, compared with 5.0% for the self-improved AG-RAG.

Table 4: **Self-improvement under the two promotion rules.** 2Wiki-Extra test ($n = 500$), with F1 score. The two "both loops" rows also include graph memory. Full results are in Appendix F.

System	Clean	Typos	50% text	Poisoned	Stress avg.	ASR ↓
Text agent	**69.1**	53.3	44.3	24.2	40.6	68.4%
AG-RAG (static)	62.5	59.6	60.4	61.8	60.6	2.4%
AG-RAG (self-improved)	66.5	63.0	**63.2**	**67.2**	**64.5**	**2.2%**
Both loops, stress-aware rule	64.9	61.3	62.8	63.8	62.6	2.8%
Both loops, clean-only rule	67.7	**63.8**	62.5	61.5	62.6	16.6%

Table 5: **Random search separates the rule from the proposer.** Each random search (RS) seed used approximately the same 3,000 agent rollouts as a harness-search seed and the same promotion rule. 2Wiki-Extra test, with F1 score and two seeds averaged; clean-only RS was evaluated only on clean and poisoned questions.

System	Clean	Stress avg.	Poisoned	ASR ↓	Turns ↓	Seeds improved
AG-RAG (static)	62.5	60.6	61.8	2.4%	4.66	–
RS, stress-aware rule	65.0	62.6	65.0	2.3%	4.24	1 / 2
AG-RAG (self-improved)	**66.5**	**64.5**	**67.2**	2.2%	3.94	2 / 2
RS, clean-only rule	68.3	–	62.7	13.8%	3.80	2 / 2

This also bounds the rule's role. Every text-search edit that passed the clean check failed the stress check, and no accepted edit changes which documents the agent can reach. Removing the text-search tool from the grammar might recover most of the gain (untested). The stress-aware rule is still useful when such tools cannot all be listed in advance: it detects any edit that lets the agent read around the graph.

6 Conclusion

We introduced AG-RAG, an agentic Graph RAG framework that treats retrieval robustness as a first-class design and optimization objective. AG-RAG combines fuzzy entity resolution, graph traversal, and provenance-bound document retrieval with a stress-aware self-improvement loop that optimizes the agent's harness across both clean and stressed conditions. Our experiments show that graph structure is particularly valuable when retrieval assumptions break: fuzzy matching repairs noisy entity names, graph relations preserve information when source text becomes unavailable, and provenance-bound retrieval limits exposure to newly injected passages. Stress-aware self-improvement raises clean F1 by 4.0 points, reduces agent turns, and keeps poisoning attack success near 2%. In contrast, optimizing the same search process only for clean performance favors free-text retrieval and substantially increases vulnerability to poisoning.

More broadly, our results show that the objective used for agent self-improvement is itself part of the system's trust boundary: optimizing nominal task performance alone can remove constraints that provide robustness. This motivates self-improving agents that evaluate candidate improvements not only by capability, but also by whether they preserve desired behavior under distribution shifts and adversarial conditions. Future work can extend this principle to unseen failure modes, continual graph updates, and objectives that jointly balance accuracy, efficiency, and robustness.

AI use statement

Large language models are part of the method studied in this paper: they answer the questions, propose harness edits, extract the graph's triples, write and verify graph facts, and generate the poisoned passages used in the poisoning condition, as described in Sections 3 and 4.

Beyond these roles, in this work, we used generative AI tools to generate synthetic data, namely the poisoned passages; to implement methods, including reproducing the baseline methods HippoRAG 2 and LinearRAG; to design experiments and provide feedback on them; to refine hypothe-

ses and the paper's conceptual framing; to clean and reformat datasets, including building the 2WikiExtra question set and converting data for the baseline systems; and to interpret results.

We have not used generative AI tools for translation. Developing theoretical models, formulating or proving mathematical claims, and conducting qualitative or thematic data analysis are not applicable to this work.

We have reviewed all AI-assisted work. Every reported number can be regenerated by scripts from the saved rollout logs and cost ledger in the research artifact, without new model calls. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Anthropic. Introducing the model context protocol, November 2024. URL https://www.anthropic.com/news/model-context-protocol.

Sukmin Cho, Soyeong Jeong, Jeongyeon Seo, Taeho Hwang, and Jong C. Park. Typos that broke the RAG's back: Genetic attack on RAG pipeline by simulating documents in the wild via low-level perturbations. In *Findings of EMNLP*, 2024.

DeepSeek-AI. DeepSeek Harness: Everything is a plugin. https://github.com/deepseek-ai/deepseek-harness, 2026.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.

Bernal J Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. Hipporag: Neurobiologically inspired long-term memory for large language models. *Advances in neural information processing systems*, 37:59532–59569, 2024.

Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. From rag to memory: Non-parametric continual learning for large language models. *arXiv preprint arXiv:2502.14802*, 2025.

Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh V Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, and Bryan Hooi. G-retriever: Retrieval-augmented generation for textual graph understanding and question answering. *Advances in Neural Information Processing Systems*, 37:132876–132907, 2024.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 6609–6625, 2020.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *ICLR*, 2025.

Bowen Jin, Chulin Xie, Jiawei Zhang, Kashob Kumar Roy, Yu Zhang, Zheng Li, Ruirui Li, Xianfeng Tang, Suhang Wang, Yu Meng, et al. Graph chain-of-thought: Augmenting large language models by reasoning on graphs. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 163–184, 2024.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into self-improving pipelines. In *ICLR*, 2024.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33: 9459–9474, 2020.

Haoran Luo, Haihong E, Guanting Chen, Qika Lin, Yikai Guo, Fangzhi Xu, Zemin Kuang, Meina Song, Xiaobao Wu, Yifan Zhu, and Anh Tuan Luu. Graph-r1: Towards agentic graphRAG framework via end-to-end reinforcement learning. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=YXnFGsSkCC.

Linhao Luo, Zicheng Zhao, Gholamreza Haffari, Dinh Phung, Chen Gong, and Shirui Pan. Gfm-rag: graph foundation model for retrieval augmented generation. *arXiv preprint arXiv:2502.01113*, 2025.

Shengjie Ma, Chengjin Xu, Xuhui Jiang, Muzhi Li, Huaren Qu, Cehao Yang, Jiaxin Mao, and Jian Guo. Think-on-graph 2.0: Deep and faithful large language model reasoning with knowledge-guided retrieval augmented generation. *arXiv preprint arXiv:2407.10805*, 2024.

Pew Research Center. When online content disappears. https://www.pewresearch.org/data-labs/2024/05/17/when-online-content-disappears/, 2024.

RapidFuzz. rapidfuzz: Home, n.d. URL https://rapidfuzz.com/.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D Manning. Raptor: Recursive abstractive processing for tree-organized retrieval. In *The Twelfth International Conference on Learning Representations*, 2024.

Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking. In *NeurIPS*, 2022.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 10014–10037, 2023.

Chong Xiang, Tong Wu, Zexuan Zhong, David Wagner, Danqi Chen, and Prateek Mittal. Certifiably robust RAG against retrieval corruption. *arXiv preprint arXiv:2405.15556*, 2024.

Zhishang Xiang, Chuanjie Wu, Qinggang Zhang, Shengyuan Chen, Zijin Hong, Xiao Huang, and Jinsong Su. When to use graphs in RAG: A comprehensive analysis for graph retrieval-augmented generation. *arXiv preprint arXiv:2506.05690*, 2025.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *ICLR*, 2024.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Chuanyue Yu, Kuo Zhao, Yuhan Li, Heng Chang, Mingjian Feng, Xiangzhe Jiang, Yufei Sun, Jia Li, Yuzhi Zhang, Qingyun Sun, et al. Graphrag-r1: Graph retrieval-augmented generation with process-constrained reinforcement learning. In *Proceedings of the ACM Web Conference 2026*, pp. 1398–1409, 2026.

Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. Self-taught optimizer (STOP): Recursively self-improving code generation. In *Conference on Language Modeling (COLM)*, 2024.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin Gödel machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*, 2025.

Luyao Zhuang, Shengyuan Chen, Yilin Xiao, Huachi Zhou, Yujing Zhang, Hao Chen, Qinggang Zhang, and Xiao Huang. Linearrag: Linear graph retrieval augmented generation on large-scale corpora. *arXiv preprint arXiv:2510.10114*, 2025.

Wei Zou, Runpeng Geng, Binghui Wang, and Jinyuan Jia. PoisonedRAG: Knowledge corruption attacks to retrieval-augmented generation of large language models. In *USENIX Security Symposium*, 2025.

A Scope and threats to validity

Model and data. The main study uses one closed answering model at temperature zero, one proposer, two proposer seeds, three generations, and Wikipedia-style multi-hop QA. The templated relations in 2Wiki may favor a graph, and the typo benefit does not clearly transfer to MuSiQue. We have not tested an open-weight answering model or a non-Wikipedia corpus.

What the poisoning result shows. The generated poison targets the text index after graph construction. Resistance comes from retrieving only pre-attack documents, which a snapshot-restricted text tool would also provide; poisoning ingested into the graph substantially weakens the boundary, as shown in Appendix K.

An asymmetric stress suite. The suite penalizes reading new text but never rewards it. With a legitimate update, where the answer appears only in post-snapshot text, a snapshot-bound route would fail. Choosing a route is a trade-off between trusting new evidence and resisting injected evidence, and its operating point depends on the deployment.

Unavailable versus deleted text. The missing-text condition models sources that become unavailable. Many real deletions are intentional: retractions, corrections, takedowns, or erasure requests. A system that keeps answering from relations extracted from such content is not robust in a desirable sense; it may violate the reason for the deletion. Graph maintenance should propagate deletions to derived edges.

Statistics and scope of selection. The promotion rule uses point estimates on 60-question stress subsets; its F1 part has little power, and ASR did most of the work. Selection and testing use the same stress families on different questions, so the rule retains robustness to known failures; generalization to unseen ones is untested. The headline matched comparison includes graph memory, and a harness-only clean-only LLM arm and a grammar without text search were not run. The memory verifier's apparent write-quality gains require independent human calibration. The shift study also assumes labeled examples under each new condition, which a deployment would need to obtain.

B Experimental details

2Wiki-Extra uses a seed-1 permutation of its question pool. Its first three batches of 110 questions supply the stream traces and optional memory writes; the next 200 form the validation split; the next 500 form the main test. The first 60 validation questions also supply each stress subset after perturbation. Remaining questions do not enter the main study. The test split stays fixed across agents and stresses. We use one test rollout per state and question. Per clean test question, the static graph agent used 7,419 tokens ($0.00047, 4.66 turns), the stress-aware harness 7,039 tokens ($0.00049, 3.92 turns), and the text agent 4,808 tokens ($0.00037, 3.30 turns); graph construction for 54,957 passages cost $14.33 ($0.00026 per passage). The paired bootstrap resamples questions 10,000 times. An answer counts as a poisoning success when its F1 against the attacker's target answer is at least 80.

The main typo condition applies level-2 character errors to question entities and hints. Missing-text evaluation keeps half the indexed passages. The poison condition adds two generated answer-targeted passages per question to the text index after the graph snapshot. All three perturbations use fixed seeds and inputs. Graph memory states, when present, are replayed across conditions. The

answering model is `gpt-6-luna` at temperature zero; harness proposals use `gpt-6-sol`. The in-house baselines below share the question split and answering model.

C PAIRED CONFIDENCE INTERVALS

Table 6 lists the paired 95% bootstrap intervals for the key differences in the main text. Each interval resamples the 500 2Wiki-Extra test questions 10,000 times; seeded systems average two seeds per question.

Table 6: Paired differences on the 2Wiki-Extra test with 95% bootstrap intervals.

Comparison	Measure	Difference [95% interval]
AG-RAG (self-improved) − LinearRAG	clean F1	$+9.8$ $[+5.9, +13.7]$
	typos F1	$+27.9$ $[+23.4, +32.3]$
	50% text F1	$+35.2$ $[+30.6, +39.8]$
	poisoned F1	$+32.2$ $[+27.8, +36.5]$
AG-RAG (self-improved) − HippoRAG 2	clean F1	$+16.3$ $[+12.0, +20.7]$
	typos F1	$+21.2$ $[+16.7, +25.9]$
	50% text F1	$+36.2$ $[+31.7, +40.7]$
	poisoned F1	$+35.2$ $[+31.1, +39.6]$
AG-RAG (self-improved) − AG-RAG (static)	clean F1	$+3.9$ $[+1.4, +6.5]$
	turns	-0.72 $[-0.85, -0.58]$
Text agent − AG-RAG (self-improved)	clean F1	$+2.7$ $[+0.2, +5.2]$
Both loops: clean-only − stress-aware rule	ASR	$+13.8$ points $[+11.0, +16.7]$

D DATASET AND GRAPH STATISTICS

The main study uses 2Wiki-Extra: 11,576 questions from the 2WikiMultihopQA development set that are not in the standard 2Wiki test set, with 54,957 passages and 569,676 extracted triples. Its 500-question test split is never seen during development or harness search. HotpotQA (Yang et al., 2018) and released 2Wiki (Ho et al., 2020) supplied development and pilot questions. MuSiQue (Trivedi et al., 2022), whose questions require compositional multi-hop reasoning, supplied the frozen-system transfer test.

Table 7 reports statistics for the *released benchmark graphs*; the 2Wiki column is not the 2Wiki-Extra graph used for the main results. These graphs have median out-degree three or four and a much longer high-degree tail. This motivates a neighbour budget in the harness: an unrestricted expansion of a hub can consume much of the agent's context window.

Table 7: Released benchmark graph statistics. The 2Wiki-Extra graph used in the main study is a separate build.

Statistic	HotpotQA	MuSiQue	Released 2Wiki
Entities	87,768	100,853	48,779
Relations	45,112	55,944	20,748
Edges	558,224	639,236	321,900
Documents	9,221	11,656	6,119
Mean out-degree	6.36	6.34	6.60
Median out-degree	3	4	4
95th-percentile out-degree	21	20	20
Maximum out-degree	613	1,086	242
Mean entity-name length	18.3	17.9	17.4
Distinct indexed trigrams	12,304	12,672	10,344
Entity–document density	0.0002	0.0001	0.0002
Mean documents per entity	1.41	1.44	1.37

E ENTITY-MATCHING LATENCY

For framework profiling, we measured fuzzy name lookup on synthetic vocabularies sampled from a dictionary, not on the 2Wiki-Extra graph. At 100,000 names, searching every name with RapidFuzz took 51.1–82.1 ms for five- or fifteen-character queries; trigram-prefiltered RapidFuzz took 2.6–4.2 ms. At one million names, the corresponding ranges were 535.2–857.5 ms and 4.8–7.3 ms. These microbenchmarks explain why candidate filtering is part of the tool interface. They do not establish an end-to-end QA gain on the current held-out test.

F COMPLETE BASELINE RESULTS

Table 8 reports the in-house retrieval and agent baselines on the same 500-question 2Wiki-Extra test. The PPR, IRCoT, and isolation-and-vote rows are in-house implementations over our corpus and model, not reproductions of native systems. Single-shot rows read the top five passages (1,200 characters each). The isolate-then-vote row answers from each of the top five hybrid passages separately and takes a plain majority of non-abstaining answers, with ties going to the higher-ranked passage and no abstention threshold. Poison passages were retrieved for every question; because a single genuine passage rarely answers a multi-hop question, genuine passages mostly abstain and the two poison passages decide the vote. RobustRAG's secure aggregation is designed to avoid this failure, so this row should not be read as a RobustRAG result. The memory-only row is included to complete the main-study arm comparison. The spelling-fix intervention was run on clean and typo questions only.

Table 8: Full 2Wiki-Extra comparison. ASR is attack success under poisoning. Seeded systems average two seeds.

System	Clean	Typos	50% text	Poisoned	Stress avg.	ASR
BM25, single-shot	35.8	13.7	19.1	16.5	16.4	83.6%
Dense, single-shot	27.5	11.9	14.7	14.2	13.6	85.4%
BM25 + dense	37.3	16.1	21.5	16.0	17.9	83.2%
BM25 + dense + PPR	42.0	16.2	22.9	14.6	17.9	84.0%
PPR over graph	41.0	12.6	19.9	40.2	24.2	11.0%
IRCoT-style	39.3	17.5	27.5	19.0	21.3	77.8%
Isolate and vote	24.0	12.8	18.9	13.0	14.9	87.4%
Text agent + spell fix	68.8	56.2	–	–	–	–
Text agent	69.1	53.3	44.3	24.2	40.6	68.4%
Static graph agent	62.5	59.6	60.4	61.8	60.6	2.4%
Memory only	62.5	61.6	61.4	62.6	61.9	–
Harness, stress-aware rule	66.5	63.0	63.2	67.2	64.5	2.2%
Both loops, stress-aware rule	64.9	61.3	62.8	63.8	62.6	2.8%
Both loops, clean-only rule	67.7	63.8	62.5	61.5	62.6	16.6%

G NATIVE GRAPH RAG SYSTEMS WITH THE SAME ANSWERING MODEL

We ran HippoRAG 2 and LinearRAG with their official code; Table 2 summarizes the results in the main text. Every LLM call went through a local proxy that forced `gpt-6-luna` at temperature zero with no extended reasoning, so all systems share one answering model. Each system keeps its published retrieval settings, prompts, and embedding model (NV-Embed-v2 for HippoRAG 2; all-mpnet-base-v2 and 1,000-token chunks for LinearRAG). HippoRAG 2 reads the same extracted graph as AG-RAG, so the two differ only in retrieval. Two compatibility patches were needed, neither of which changes a method: HippoRAG 2's filter parser now accepts an end marker written without a space, and its embedding batches are sized by text length. Stresses follow Section 3: unavailable passages stay in each index but cannot be read, and poison passages are added through each system's normal document path. All rows are scored with our EM/F1 scorer.

The ports reproduce published levels. On its own released 2Wiki data, LinearRAG scores 61.0 EM / 71.4 F1 against a reported 63.3 / 69.7 with a different reader. HippoRAG 2 places 91.9% of gold documents in its top five on 2Wiki, near its reported recall. Its lower answer F1 with our model comes

from the reader: with HippoRAG 2's prompt, `gpt-6-luna` answers "not specified" on 22.4% of 2Wiki questions (LinearRAG 13.7%, our agent 5.2%). On the 2Wiki-Extra test HippoRAG 2's reader answers "not specified" on 9.8% of clean questions. HotpotQA uses the 944 unique questions of the GFM-RAG test set; other papers use a different HotpotQA sample, so published HotpotQA numbers are not directly comparable.

Table 9: Native Graph RAG systems with the same answering model; this is the full version of Table 2. Top: clean EM / F1 on the standard test sets. Bottom: 2Wiki-Extra test F1 under stress and poisoning ASR.

System	HotpotQA	2Wiki	MuSiQue
HippoRAG 2	**59.3 / 75.8**	56.1 / 64.1	33.9 / 48.0
LinearRAG	57.5 / 73.9	**62.8 / 70.9**	33.8 / 46.4
AG-RAG (static)	57.0 / 72.9	56.9 / 66.0	**39.0** / 50.3
AG-RAG (self-improved)	55.5 / 72.7	60.7 / 70.5	38.6 / **52.0**

System	Clean	Typos	50% text	Poisoned	ASR
HippoRAG 2	50.1	41.7	27.0	31.9	57.2%
LinearRAG	56.7	35.1	28.0	35.0	48.0%
AG-RAG (static)	62.5	59.6	60.4	61.8	2.4%
AG-RAG (self-improved)	**66.5**	**63.0**	**63.2**	**67.2**	**2.2%**

H MuSiQue transfer of frozen harnesses

We evaluated the fixed main-study harnesses on 200 MuSiQue questions without proposing or promoting MuSiQue-specific edits. The stress-aware harness gains 3.2 stress-averaged F1 points over the static graph agent, with paired 95% CI $[+0.2, +6.2]$. Its difference on clean questions is uncertain. The static graph agent exceeds the text agent under poisoning by 17.2 points ($[+10.0, +24.4]$), while its typo interval includes zero.

Table 10: Frozen-harness transfer to MuSiQue ($n = 200$). The two optimized harness rows average two main-study seeds; ASR is measured under poisoning. †Harness selected by the both-loops clean-only arm; only the harness transfers, not its memory edits, which name 2Wiki entities.

System	Clean	Typos	50% text	Poisoned	Stress avg.	ASR
Static graph agent	43.6	41.9	40.5	42.4	41.6	4.0%
Text agent	47.4	44.3	33.5	25.2	34.3	70.5%
Stress-aware harness	45.9	47.1	40.1	47.1	44.8	5.0%
Clean-only harness†	47.1	48.3	40.1	33.6	40.7	51.2%

I Self-improvement trace

Each loop proposed 18 edits (2 seeds $\times$ 3 generations $\times$ 3 candidates); all 54 were type-valid. The harness loop with the stress-aware rule: 7 passed the clean check, 3 passed both checks, 2 were promoted. Both loops with the stress-aware rule: 2, 1 and 1. Both loops with the clean-only rule: 5 passed and 3 were promoted. Table 11 shows one chain; the full log is in the supplementary material.

J Regime-shift deployment

Stream batches for each phase were new questions; the rule and test splits are the main study's. Seed 0 adopted text search in the clean phase (+4.8 validation F1 points) and kept it in all later phases (poisoned-phase ASR 11.2%, 21.6% if frozen). Seed 1 chose a graph-only harness in the clean phase and never changed it, so Adaptive equals Frozen in that seed.

Table 11: One chain (harness loop, stress-aware rule, seed 1). Δ values are against the incumbent on the validation split and F1 changes are in points. Both text-search edits improved clean F1 but raised ASR by 12–18 percentage points, and were rejected. Stress ΔF1 lists typos / missing text / poisoning.

Gen	Proposed edit	Clean ΔF1	Stress ΔF1	ΔASR	Decision
1	balanced policy + text search	+6.8	+2.0 / −6.0 / −7.0	**+18.3**	reject (stress)
1	IDF scorer + balanced policy	+1.9	–	–	reject (clean)
2	2 docs, IDF, balanced	+4.5	+6.0 / +2.0 / +4.0	−1.7	**accept**
2	documents early + text search	+5.0	+5.0 / −1.0 / +1.0	**+11.7**	reject (stress)
3	text search + strategy note	+3.3	–	–	reject (clean)

Table 12: Re-tuning on the current condition does not beat tuning once on clean data, and neither matches a promotion rule that saw the stresses from the start. F1 uses a 0–100 scale; 125 test questions per phase, 2 seeds averaged; ASR is measured in the poisoned phase.

System	Clean	Typos	50% text	Poisoned	All 500	ASR
Static graph	60.5	63.5	**63.9**	59.3	61.8	3.2%
Frozen after clean	**65.8**	65.5	**63.9**	59.1	63.6	12.0%
Adaptive	**65.8**	**66.5**	62.6	61.0	63.9	6.8%
Stress-aware harness	65.3	65.8	63.4	**63.3**	**64.5**	**2.4%**

K MEMORY: POLLUTION AND POISONING THROUGH INGESTION

Pollution (HotpotQA, 6 generations of 90 stream questions): the share of committed writes that fail grounding or the verifier is 19–21% with no checks, 4–8% with grounding, and 0–3% with grounding plus the verifier, flat across generations. These rates are judged by the same LLM verifier. Independent human calibration is not yet available.

Poisoning through ingestion: when the graph ingests new documents that include poison, the static graph agent's ASR rises from 2.7% to 40%. Grounding, the verifier and a contradiction check reduce it only to 35%, because the poison documents really state the false facts. Cross-source corroboration reduces it to 10% but rejects 96% of genuine new facts, since most facts appear in only one document. The write policy is a security–utility trade-off whose operating point cannot be set on clean data.

An initial augmentation check wrote triples from a question's own answer and asked the same question again, which measures lookup, not learning. With a disjoint question stream and temperature 0, the step saving shrinks from 2.24 to 0.33 turns.