

Graph-free Retrieval with Graph-Modified Embeddings

Federica Santi* Vasilije Marković Daniel Scott Lazar Obradović
Cognee Inc.

Abstract

Dense vector retrieval can miss evidence connected through intermediate entities, while direct graph retrieval requires access to the graph at query time. We compare five offline graph-based embedding transformations and strategies for using the resulting graph-modified embeddings alongside the originals, without query-time graph access. On a synthetic professional-network benchmark with complete supporting-evidence sets, direct replacement of the original embeddings reduces evidence recovery. Multi-stage strategies improve recovery by using graph-modified embeddings to guide the search while retaining the originals for semantic relevance. On questions requiring linked facts or comparisons among candidates, the strongest configuration improves recovery by up to 25.9 % relative to plain vector retrieval. It also matches baseline recovery with 40 % fewer returned nodes. Controls confirm that graph information contributes beyond multi-stage retrieval alone. Across the tested methods, retrieval performance depends more on how graph-modified embeddings are used than on how they are produced.

1 Introduction

Retrieval is a central component of retrieval-augmented generation (RAG), where external evidence is retrieved to support large language model (LLM) outputs [1]. It also supports LLM agents and long-term memory systems by identifying relevant documents, passages, or stored facts. Dense retrieval embeds queries and passages in a shared vector space and ranks candidates by vector similarity [2]. This approach is effective when supporting evidence is semantically close to the query, but may miss relevant passages connected through intermediate entities or multi-step relations. Knowledge graphs represent these connections explicitly, but using them for retrieval introduces two challenges. Traversing or reranking the graph for every query increases query-time complexity, while modifying embeddings according to graph structure may distort the semantic relationships captured by the original embeddings. This leads to the central question of this work: *can graph structure be incorporated offline into graph-modified embeddings that improve retrieval without requiring graph access at query time or replacing the original embeddings?*

Prior work has either incorporated relational knowledge into embeddings [3, 4, 5], used graphs directly at query time [6, 7], or refined retrieval through textual feedback alone [8, 9, 10]. Our work connects these directions by examining both how graph-modified embeddings should be constructed and how they should be used in multi-stage retrieval.

We construct graph-modified embeddings offline using several transformations, including local neighbourhood aggregation, random-walk endpoints, community centroids [11], and convex neighbour pull. We then compare retrieval strategies ranging from replacing the original embeddings with graph-modified embeddings to multi-stage strategies that combine both. At

*Correspondence: federica@topoteretes.com.

query time, every strategy uses only cosine-similarity searches over stored embeddings; the graph is neither traversed nor queried.

Our central hypothesis is that graph-modified embeddings are more useful for guiding the expansion of retrieval than for measuring relevance to the original query. The best-performing strategies therefore assign distinct roles to the two sets of embeddings. An initial search over the original embeddings retrieves semantically relevant results (seeds). Their graph-modified embeddings are then aggregated to guide the selection of candidates that are both relevant to the query and related to the seeds through the graph. Finally, the original embeddings of the selected candidates are used to reformulate the query for another search over the original embeddings [8, 9, 10]. We refer to this use of graph-modified embeddings to guide retrieval as *graph conditioning*.

To evaluate these strategies, we introduce a synthetic professional-network benchmark in which the corpus, graph, questions, answers, and supporting evidence are generated from the same underlying facts. The benchmark includes single-fact, relational and collection, and extremisation questions, together with exact gold answers and complete gold-evidence sets containing the intermediate and competing records required to establish each answer [12].

We select configurations on a development corpus and report results on separately generated validation data (Section 3.1). All methods use the same original embeddings, cosine similarity, and the same 40-node retrieval limit. Multi-stage controls without graph conditioning allow us to distinguish improvements produced by graph-modified embeddings from those produced by multi-stage retrieval alone.

None of the graph-modified representations improves direct retrieval over the original embeddings in our experiments. The original embeddings recover 73.1 % of the gold evidence, whereas the five transformations recover between 52 % and 72.9 %. In contrast, multi-stage retrieval improves performance across all five transformations, and the choice of retrieval strategy has a greater effect than the transformation used to construct the graph-modified embeddings. The strongest configuration improves gold-evidence recovery relative to plain vector retrieval by 25.9 % on extremisation questions and by 10.0 % on relational and collection questions (19.2 % across the two groups). It also matches the gold-evidence recovery achieved by plain vector retrieval with 40 nodes after returning only 24, corresponding to 40 % fewer returned nodes.

Contributions.

- We provide a controlled comparison of multiple graph-based embedding transformations and retrieval strategies under a common cosine-similarity framework, without requiring graph access at query time.
- We introduce a synthetic benchmark with questions of different relational complexity, exact gold answers, and complete question-specific supporting evidence.
- We show that retrieval performance depends more on how graph-modified embeddings are used than on the particular transformation that produces them, and that they are most effective as intermediate signals within a multi-stage retriever.
- We introduce a retrieval strategy in which graph-modified embeddings guide exploration while the original embeddings preserve semantic alignment with the query, and we quantify its performance.

2 Related Work

Our work connects research on graph-modified embeddings, feedback-based and multi-hop retrieval, and graph-based retrieval. We also draw on evidence-level evaluation to assess whether these approaches recover the passages needed to answer a question.

Encoding graph structure into embeddings. Retrofitting adapts pretrained embeddings using external relational knowledge. Faruqui et al. [3] introduced an objective that balances fidelity to the original embeddings with attraction between connected concepts. Counter-fitting adds attraction and repulsion constraints [4], while Functional Retrofitting introduces relation-specific transformations for heterogeneous knowledge graphs [5]. Retrofitting has also been applied beyond word embeddings, including multilingual sentence representations aligned through Abstract Meaning Representation [13] and sense representations specialised using WordNet relations [14]. Related approaches use entity descriptions and language models to construct representations for knowledge-graph completion, incorporating structural information through different training objectives or architectures [15, 16, 17, 18, 19].

Another line of work derives node representations from graph connectivity. DeepWalk learns embeddings from contexts generated by truncated random walks [20], while node2vec uses biased walks to capture different network neighbourhoods [21]. These methods are conceptually related to our random-walk transformation, which instead averages existing textual embeddings at walk endpoints without learning new representations from walk sequences. Simple Graph Convolution separates graph-based feature propagation from a learned classifier [22], providing a related perspective on our neighbourhood-based aggregation of existing embeddings.

We draw on these ideas to study graph-modified embeddings for evidence retrieval, rather than intrinsic similarity, node classification, or knowledge-graph completion. In particular, we compare their use as direct search representations with their use as intermediate signals in a multi-stage retriever.

Feedback-based and multi-hop dense retrieval. Pseudo-relevance feedback refines a query using the results of an initial retrieval stage and then performs a second search. NPRF (neural pseudo-relevance feedback) brought the idea into learned ranking, feeding the top-ranked documents of a first pass into a neural ranking model [8]. More recent approaches apply the same principle to dense retrieval, where the query is a vector rather than a set of terms, and differ in what that vector is allowed to be. ColBERT-PRF keeps a multi-vector query and extends it, extracting representative embeddings from the pseudo-relevant documents and adding them to the query representation before re-retrieval [9]. ANCE-PRF keeps a single query vector but learns an encoder that consumes the initially retrieved passages, leaving the document index unchanged [10]. Multi-hop retrieval extends this sequential approach to questions whose supporting evidence cannot be identified from the original query alone. Multi-Hop Dense Retrieval conditions successive searches on previously retrieved passages to recover chains of supporting evidence [23]. Other approaches use generated text to guide subsequent retrieval. IRCOT interleaves retrieval with intermediate chain-of-thought reasoning steps [24], while IterRetGen alternates retrieval and generation, using each generated output to inform the next retrieval iteration [25].

Our method uses offline graph-modified embeddings to guide intermediate retrieval, followed by a Rocchio-style update that blends the query with the mean embedding of retrieved nodes in the original space [26], without query-time graph access or LLM generation. Controls without graph conditioning (Section 4.5.2) isolate the contribution of graph conditioning.

Graph-based retrieval. Graph-based systems use explicit relations to find evidence that semantic similarity may miss, such as education records linked to a company through its employees. GRAFT-Net reasons over question-specific subgraphs of facts and text [6]; PullNet expands them through iterative retrieval [7]; and HippoRAG propagates relevance through entity connections using Personalized PageRank [29]. These systems use the graph during query processing, whereas we encode its relations into embeddings offline and retrieve evidence through stored vectors alone.

GraphRAG uses community summaries to answer global questions about a corpus [30]. Although related to our use of graph communities, it produces text for corpus-level synthesis, whereas we produce node-level vectors for retrieving question-specific evidence under a fixed budget.

Evidence-level evaluation. Multi-hop question-answering benchmarks support evaluation beyond final-answer accuracy by identifying the evidence needed for reasoning. HotpotQA provides sentence-level supporting facts alongside questions and answers, including both bridge and comparison questions [27]. MuSiQue constructs multi-hop questions by composing single-hop questions whose reasoning steps depend on one another, explicitly addressing shortcuts that allow apparently multi-hop questions to be answered without combining the required evidence [28]. Ground-truth-subgraph benchmarks such as SynthKGQA and GTSQA further support the evaluation of graph retrieval independently of answer generation [12]. We adopt this evidence-centred perspective: our synthetic corpus and questions are generated from the same structured facts, enabling us to specify question-specific gold-evidence sets and measure their recovery directly. The benchmark includes relational and collection questions and extremisation questions, allowing us to examine retrieval of both indirectly connected evidence and the competing records required to establish an extremum.

3 Experimental Setup

We evaluate how graph-based embedding transformations and retrieval strategies affect evidence recovery on a synthetic professional-network benchmark.

3.1 Dataset and Knowledge Graph

We use separate development and validation corpora, generated by the same pipeline and containing synthetic resumes and organisational personnel rosters.

Each corpus is represented as a tripartite graph with three node types: documents, chunks (textual evidence from those documents), and entities (people and organisations). Three relation types connect them: **part_of** links chunks to documents, **about** links documents to their primary entities, and **mentions** links chunks to entities referenced in their text. The schema admits only these three edge patterns. Each corpus contains approximately 1,200 chunks, 370 entities, 230 documents, and 3,700 edges. The development and main validation corpora are node-disjoint: none of the 1,759 node identifiers in the validation corpus occurs in the development corpus. We use Coguee¹ to ingest and store the generated graph and manage its associated node embeddings.

We select configurations on the development corpus and report results on the validation set of 977 questions, including 90 generated on a supplementary corpus from the same pipeline. Appendix A provides generation details, exact validation statistics, and the schema with an example (Figure 10).

¹An open-source AI memory framework combining knowledge graphs and vector search: <https://github.com/topoteretes/coguee>.

3.2 Questions and Gold Evidence

We generate questions and ground-truth (gold) answers from the structured facts underlying the dataset. Gold evidence includes every chunk required to establish an answer. The benchmark contains 11 question families, detailed in Appendix A. We group them into three categories:

Single-fact questions: These ask for attributes directly associated with entities mentioned in the query and generally require a single evidence chunk or an immediate relation. For example, “What skills did a person use at a given employer?” can be answered using a job chunk that mentions the person, employer, and skills.

Relational and collection questions: These ask for attributes or entities connected to a named entity through intermediate entities. For example, “Which universities were attended by the employees of an organisation?” requires evidence linking the organisation to its employees and those employees to their universities. The target chunks may lie two or three relations from the named entity and share little surface content with the query.

Extremisation questions: These compare candidates to identify a maximum and may then request another attribute of the selected candidate. Establishing the maximum requires the relevant records of all competing candidates. For example, “What certifications does the most experienced worker at a given company hold?” requires experience records for the comparison and the winner’s certification evidence, which may be semantically unrelated to the comparison criterion.

3.3 Graph-Based Embedding Transformations

We modify the original embeddings offline to incorporate information from related nodes in the graph. The five transformations differ in which nodes influence each modified vector and how their contributions are combined (Figure 1).

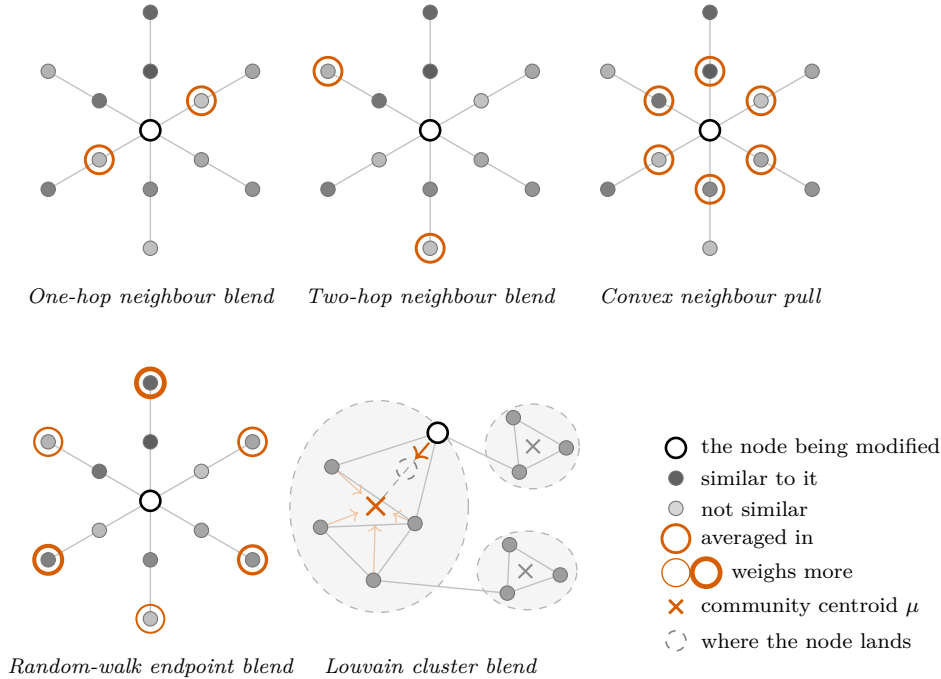


Figure 1: Graph information used by the five transformations. Darker nodes are more similar to the source node; orange outlines mark embeddings included in an average, with thicker outlines indicating greater weight. Dashed regions and crosses show Louvain communities and their centroids.

Original embeddings encode chunk text or, for document and entity nodes, their names. We embed nodes and queries with OpenAI’s `text-embedding-3-small`, using 1,536 dimensions. Each transformation produces one modified vector per node. We denote the original and modified embeddings of node i by $\mathbf{x}_i$ and $\mathbf{z}_i$, respectively. Appendix B lists all evaluated transformations and hyperparameter configurations.

One-hop neighbour blend (hop-1). For each node i , we consider its one-hop neighbours. We define S_i as the 20 % of these nodes with the lowest cosine similarity to the original embedding $\mathbf{x}_i$, selecting at least two whenever available. The modified embedding $\mathbf{z}_i$ combines the original vector with the mean of the original embeddings of the selected neighbours:

$$\mathbf{z}_i = \alpha \mathbf{x}_i + (1 - \alpha) \frac{1}{|S_i|} \sum_{j \in S_i} \mathbf{x}_j, \quad (1)$$

where $\alpha = 0.2$. Selecting the least semantically similar one-hop neighbours is intended to introduce graph-related information that is not already encoded in the source representation.

Two-hop neighbour blend (hop-2). This method uses the same selection and blending procedure, applied to nodes at graph distance exactly 2.

Convex neighbour pull. Let $\mathbf{Z}$ collect the modified node embeddings and E denote the graph’s edge set. We minimise the classical retrofitting objective [3] with uniform anchor and edge weights:

$$J(\mathbf{Z}) = \sum_i \|\mathbf{z}_i - \mathbf{x}_i\|_2^2 + \sum_{(i,j) \in E} \|\mathbf{z}_i - \mathbf{z}_j\|_2^2. \quad (2)$$

The first term anchors each node to its original embedding, while the second pulls adjacent nodes closer together in the modified space. The anchoring term makes the objective strictly convex, ensuring a unique global minimiser to which the iterative optimisation procedure converges.

Random-walk endpoint blend (walk). This transformation launches 100 random walks of exactly two steps from each node. The modified embedding is computed as the frequency-weighted mean of the endpoints’ original embeddings.

Louvain cluster blend. This transformation partitions the unweighted, undirected graph using Louvain community detection at resolution 0.5; we refer to Blondel et al. [11] for details of the community-construction algorithm. Each node i is then moved towards the centroid $\boldsymbol{\mu}_{c(i)}$ of its community $c(i)$:

$$\mathbf{z}_i = \alpha \mathbf{x}_i + (1 - \alpha) \boldsymbol{\mu}_{c(i)}. \quad (3)$$

Each centroid is the mean of its community’s original embeddings and remains fixed. The selected configuration uses $\alpha = 0.5$, assigning equal weight to the node’s original embedding and its community centroid.

3.4 Retrieval Strategies

We compare retrieval strategies that use the original and graph-modified embeddings separately or together. These range from direct searches in either space to combinations of independent rankings and multi-stage searches guided by earlier results. We hold the output at 40 distinct nodes to compare evidence recovery under the same retrieval budget (Figure 2).

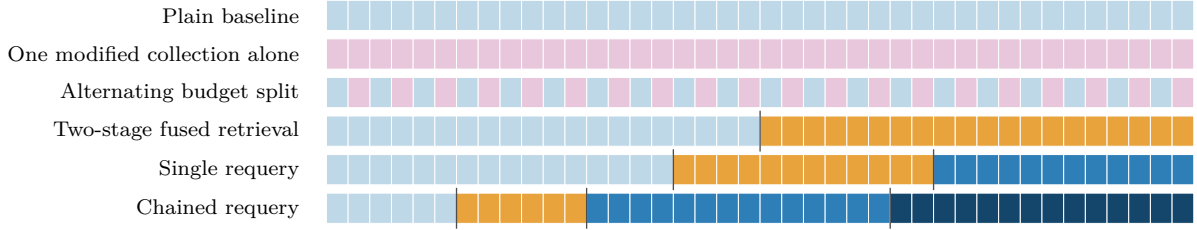


Figure 2: Allocation of the 40-node retrieval budget. Each cell represents one returned node. Light blue and pink denote original- and modified-space searches; orange denotes fused scoring. The two darker blues denote successive searches with shifted queries in the original space. Vertical lines separate stages.

All strategies use cosine similarity over stored embeddings of all node types, without graph access at query time. Later stages exclude previously returned nodes. Appendix C lists all evaluated methods and configurations.

Direct retrieval. The first two rows of Figure 2 each use a single embedding collection. The *plain baseline* returns the nodes whose original embeddings are most similar to the query embedding $\mathbf{q}$. *One modified collection alone* applies the same search to graph-modified embeddings. This comparison tests whether incorporating graph structure into the vectors makes them more useful for direct query matching.

Alternating budget split. This strategy gives both spaces a share of the output. We search them independently using $\mathbf{q}$ and alternate between their rankings, skipping duplicate node identifiers.

Two-stage fused retrieval. Here, the initial results guide the search for further evidence. The first stage retrieves 20 nodes (seeds) from the original space using $\mathbf{q}$ and computes $\mathbf{m}$ as the mean of their modified embeddings. The second stage pools candidates from an original-space search with $\mathbf{q}$ and a modified-space search with $\mathbf{m}$, then ranks each candidate node n by the *fused score*

$$s(n, \alpha) = (1 - \alpha) \cos(\mathbf{q}, \mathbf{x}_n) + \alpha \cos(\mathbf{m}, \mathbf{z}_n). \quad (4)$$

The fusion weight $\alpha = 0.5$ gives equal weight to similarity to the original query and similarity to the seeds in the modified space. The 20 highest-scoring remaining nodes complete the output.

Single requery. This strategy uses the fused-stage results to refine the query for another search in the original space. It retrieves 16 seeds, selects 12 nodes using the fused score in Equation (4), and averages the latter’s original embeddings to obtain $\bar{\mathbf{x}}$. The shifted query

$$\mathbf{q}' = 0.7\mathbf{q} + 0.3\bar{\mathbf{x}} \quad (5)$$

retrieves the final 12 nodes, giving a 16/12/12 allocation.

Chained requery. This strategy adds a second query update, so the first shifted search supplies evidence for the next. It uses 6/6/14/14 nodes for seed retrieval, fused scoring, and two shifted-query searches in the original space. Each update uses the mean original embedding of the preceding stage’s results. Both use Equation (5) with the original query $\mathbf{q}$; only the mean $\bar{\mathbf{x}}$ changes.

Controls without graph conditioning. To separate the contribution of graph conditioning from that of query shifting, we compare each query strategy with two controls that change only its second stage. One sets the fusion weight in Equation (4) to $\alpha = 0$; the other retains $\alpha = 0.5$ but uses original embeddings for both the seed mean $\mathbf{m}$ and the candidates. Section 4.5.2 presents this comparison.

4 Experimental Results

We first test graph-modified embeddings in direct retrieval, then compare strategies that combine original and modified representations. Finally, control experiments assess the contribution of graph conditioning.

For each question, gold-evidence recovery is the fraction of its gold chunks returned; reported percentages average this fraction equally across questions. Document and entity nodes consume retrieval budget but do not count as recovered evidence.

4.1 Direct Retrieval with Graph-Modified Embeddings

We compare the original embeddings with the five graph-modified collections as direct search representations across all 977 validation questions.

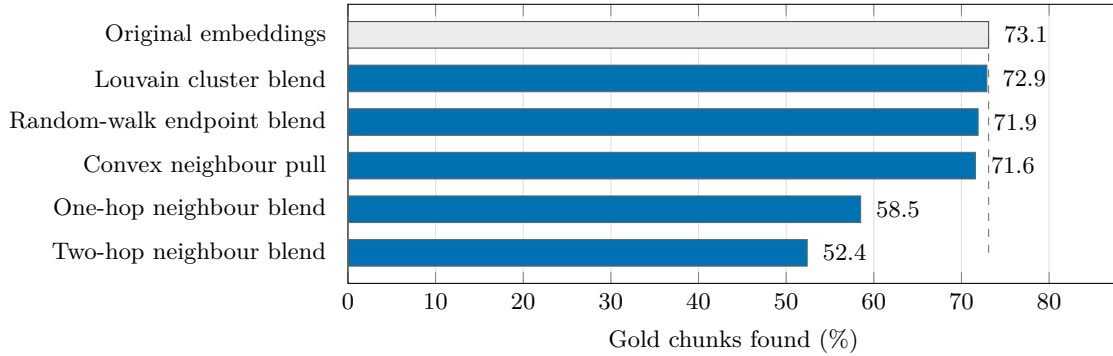


Figure 3: Gold-evidence recovery under direct retrieval across all 977 validation questions. The dashed line denotes plain vector retrieval.

All five transformations recover fewer gold chunks than the original embeddings, with the largest losses observed for the one-hop and two-hop neighbour blends.

4.2 Comparison of Retrieval Strategies

Plain retrieval already recovers 98.9% of the gold evidence on the 450 single-fact questions, leaving little room to distinguish strategies. Unless otherwise stated, the remaining comparisons therefore focus on the 527 difficult questions in the other two groups. Figure 4 compares the five embedding transformations across retrieval strategies.

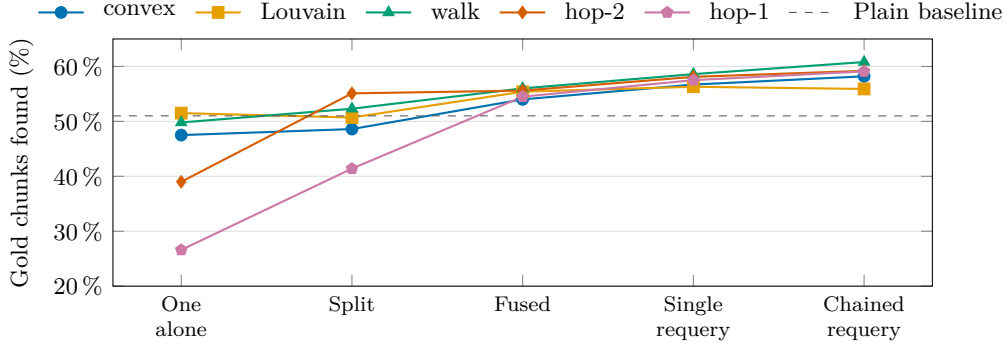
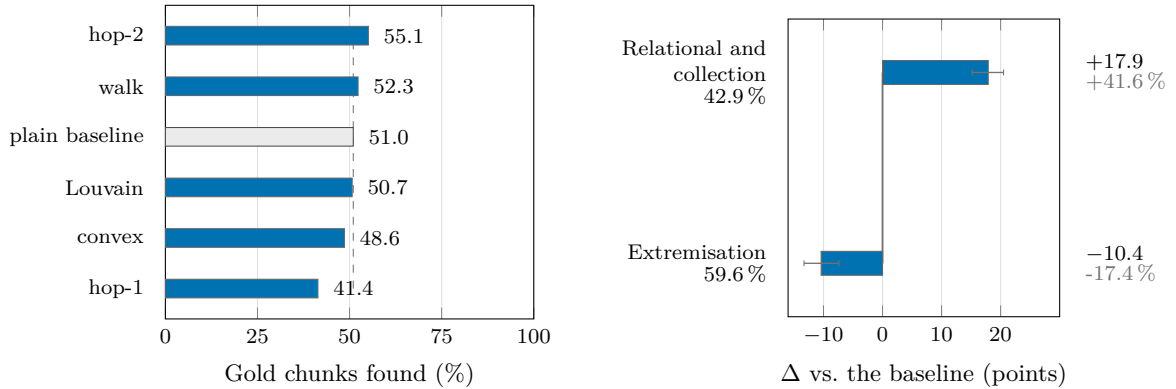


Figure 4: Gold-evidence recovery across retrieval strategies on the 527 difficult questions. The dashed line denotes plain retrieval.

Performance varies most across embedding transformations under direct retrieval and budget splitting. Fused retrieval and requery outperform the plain baseline with all five transformations, with smaller differences between them.

4.3 Alternating Budget Split

The alternating split assigns half of the retrieval budget to each embedding space. In the group-level plots, error bars show 95 % confidence intervals for changes relative to the baseline, estimated from 10,000 paired bootstrap resamples of questions.



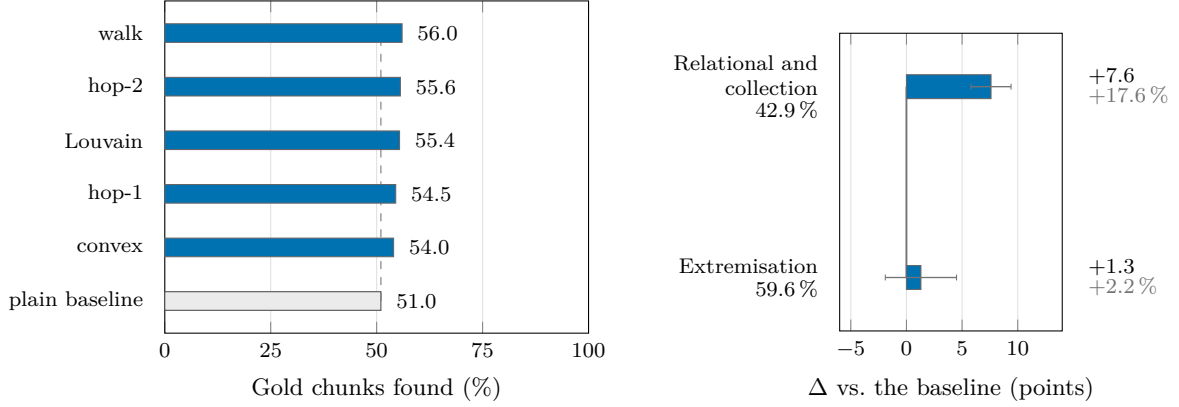
(a) Gold-evidence recovery under the alternating budget split for each graph-modified embedding collection. The dashed line denotes the performance of plain vector retrieval. (b) Effect of the hop-2 alternating budget split across question groups. Baseline recovery is shown below each group label; bar annotations report the corresponding absolute and relative changes.

Figure 5: Alternating budget split on the 527 difficult questions.

The alternating budget split reveals a clear trade-off. Panel (a) shows that only the walk and hop-2 representations outperform plain retrieval, with hop-2 performing best. However, Panel (b) shows that its gain is uneven: relational and collection questions improve by 41.6 %, whereas extremisation questions decline by 17.4 % relative to their baselines. The modified space helps reach semantically distant but graph-connected evidence, while assigning half of the budget to it reduces the broad candidate coverage required for extremisation.

4.4 Two-Stage Fused Retrieval

We next test whether letting the original-space results guide the second stage yields more consistent gains than the independent budget split.



(a) Gold-evidence recovery of the two-stage fused retriever across graph-modified embedding collections. The dashed line denotes plain vector retrieval.

(b) Effect of two-stage fused retrieval with the hop-2 representation across question groups. Baseline recovery is shown below each group label; bar annotations report absolute and relative improvements.

Figure 6: Two-stage fused retrieval on the 527 difficult questions.

Two-stage fused retrieval improves over the baseline with every graph-modified collection, recovering 54.0 % to 56.0 % of the gold evidence against 51.0 % for plain retrieval. The narrow range suggests less sensitivity to the choice of transformation than under budget splitting. The walk representation performs best, gaining 5.0 percentage points (9.8 % relative).

Panel (b) again uses hop-2 to compare directly with the alternating split. Its gain is concentrated on relational and collection questions: 7.6 points (17.6 % relative). The improvement on extremisation questions is small and statistically uncertain.

4.5 Requery Strategies and Graph Conditioning

4.5.1 Performance of Requery Strategies

We examine whether using the fused-stage results to refine the query improves recovery beyond the two-stage schedule.

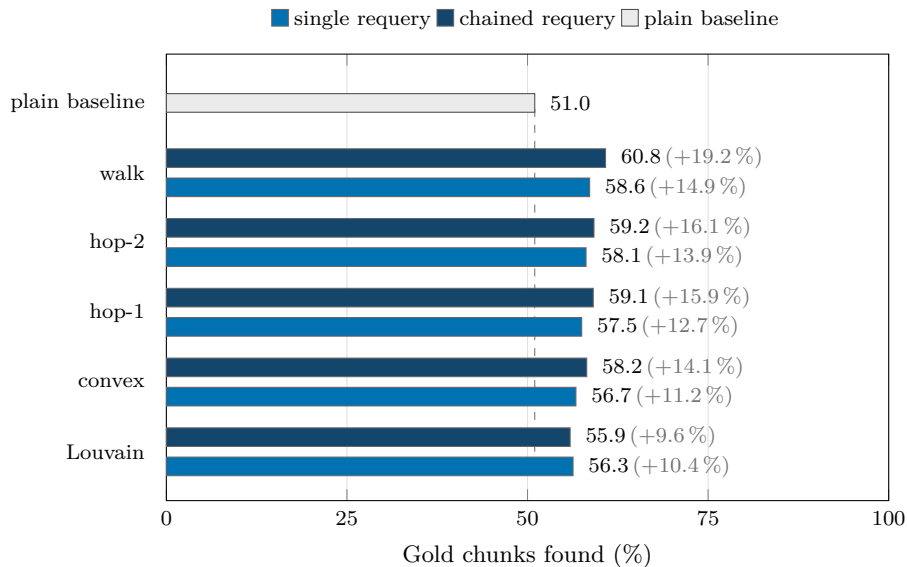


Figure 7: Gold-evidence recovery of the single requery and chained requery schedules across graph-modified embedding collections. The dashed line denotes plain vector retrieval.

Chained requery with the walk representation performs best, recovering 60.8 % of the gold evidence against 51.0 % for plain retrieval, a gain of 9.8 percentage points (19.2 % relative). The walk representation also performs best with single requery.

Unlike the alternating split, this configuration improves both difficult question groups (Figure 8). Recovery increases by 15.4 points on extremisation questions (25.9 % relative) and by 4.3 points on relational and collection questions (10.0 % relative).

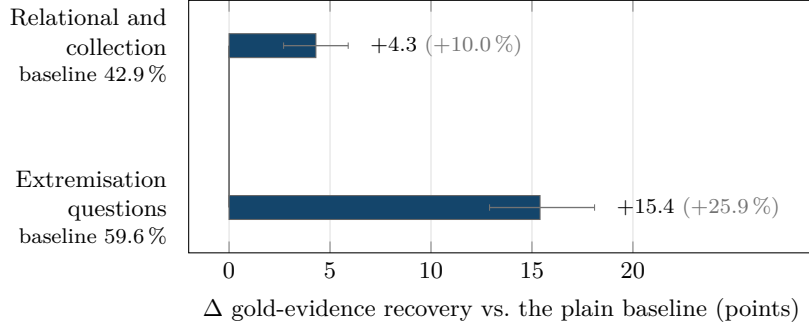


Figure 8: Improvement of chained requery with the walk representation over plain retrieval, reported by question group. Annotations show absolute and relative gains.

4.5.2 Contribution of Graph Conditioning

The fused stage selects the nodes used for subsequent query shifts, so graph conditioning can affect later searches as well as the second-stage results.

We compare chained requery with two controls using the same 6/6/14/14 schedule. The first sets the fusion weight in Equation (4) to $\alpha = 0$, reducing the second-stage score to $\cos(\mathbf{q}, \mathbf{x}_n)$. The second uses original embeddings for both seeds and candidates:

$$0.5 \cdot \cos(\mathbf{q}, \mathbf{x}_n) + 0.5 \cdot \cos(\mathbf{m}, \mathbf{x}_n), \quad (6)$$

where $\mathbf{m}$ is the mean original embedding of the first-stage results. These controls measure the benefit of the multi-stage schedule and query reformulation without graph conditioning.

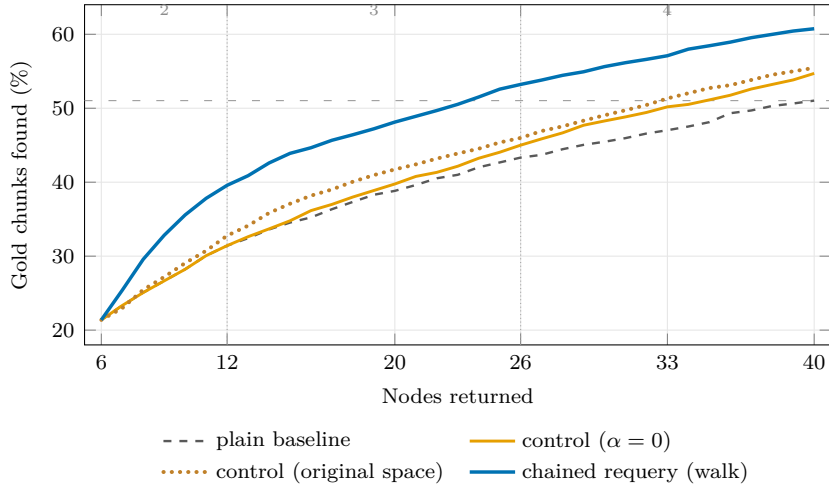


Figure 9: Cumulative gold-evidence recovery over the first k nodes returned in stage order, without global reranking. The full chained requery method uses the walk representation; both controls retain its 6/6/14/14 schedule but remove graph conditioning. Vertical dotted lines mark stage boundaries. The horizontal dashed line shows plain retrieval’s recovery at 40 nodes.

At the full budget, the controls recover 54.7 % and 55.5 % of the gold evidence, compared with 51.0 % for plain retrieval (Figure 9). Original-space fusion provides only a small improvement over query-only scoring in this configuration.

The full method reaches 60.8 %, exceeding the stronger control by 5.3 points. Graph conditioning thus contributes slightly more than half of its 9.8-point gain over plain retrieval. The advantage emerges during the fused stage and persists through later searches. After twelve returned nodes, recovery is 39.6 %, against 31.4 % and 32.8 % for the controls.

The full method also matches plain retrieval’s 40-node recovery after returning only 24 nodes, or 40 % fewer returned nodes.

5 Conclusion

Our results show that graph structure can improve evidence retrieval without graph access at query time. Graph-modified embeddings guide the search for additional evidence, while the original embeddings maintain semantic relevance to the query. This supports developing graph-modified embeddings specifically for multi-stage retrieval and evaluating their contribution to the complete retrieval process.

Retrieval should adapt as evidence accumulates. The question and earlier results should guide when to use graph information, how to allocate the remaining search budget, and when enough evidence has been found to support an answer.

Future evaluation should test the approach on natural corpora, additional embedding models, and graphs with missing or incorrect relations. It should also establish whether smaller retrieved contexts offset the cost of additional searches. The broader goal is to design retrieval with the final answer in mind, using semantic similarity and graph structure together to provide the evidence needed within practical computational limits.

CRedit Author Statement

Federica Santi: Conceptualization, Data Curation, Formal Analysis, Investigation, Methodology, Software, Validation, Visualization, Writing–Original Draft, Writing–Review & Editing.

Vasilje Marković: Conceptualization, Funding Acquisition, Project Administration, Resources, Supervision.

Daniel Scott: Conceptualization, Data Curation, Formal Analysis, Investigation, Methodology, Software, Validation, Visualization, Writing–Original Draft.

Lazar Obradović: Conceptualization, Methodology, Project Administration, Supervision, Writing–Review & Editing.

References

- [1] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-T., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, vol. 33 (pp. 9459-9474). <https://arxiv.org/abs/2005.11401>.
- [2] Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., & Yih, W.-T. (2020). Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing* (pp. 6769-6781). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2020.emnlp-main.550>.
- [3] Faruqui, M., Dodge, J., Jauhar, S. K., Dyer, C., Hovy, E., & Smith, N. A. (2015). Retrofitting word vectors to semantic lexicons. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 1606-1615). Association for Computational Linguistics. <https://doi.org/10.3115/v1/N15-1184>.
- [4] Mrkšić, N., Ó Séaghdha, D., Thomson, B., Gašić, M., Rojas-Barahona, L. M., Su, P.-H., Vandyke, D., Wen, T.-H., & Young, S. (2016). Counter-fitting word vectors to linguistic constraints. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 142-148). Association for Computational Linguistics. <https://doi.org/10.18653/v1/N16-1018>.
- [5] Lengerich, B., Maas, A., & Potts, C. (2018). Retrofitting distributional embeddings to knowledge graphs with functional relations. In *Proceedings of the 27th International Conference on Computational Linguistics* (pp. 2423-2436). Association for Computational Linguistics. <https://aclanthology.org/C18-1205/>.
- [6] Sun, H., Dhingra, B., Zaheer, M., Mazaitis, K., Salakhutdinov, R., & Cohen, W. W. (2018). Open domain question answering using early fusion of knowledge bases and text. In *Proceedings of EMNLP 2018* (pp. 4231-4242). <https://doi.org/10.18653/v1/D18-1455>.
- [7] Sun, H., Bedrax-Weiss, T., & Cohen, W. W. (2019). PullNet: Open domain question answering with iterative retrieval on knowledge bases and text. In *Proceedings of EMNLP-IJCNLP 2019* (pp. 2380-2390). <https://doi.org/10.18653/v1/D19-1242>.
- [8] Li, C., Sun, Y., He, B., Wang, L., Hui, K., Yates, A., Sun, L., & Xu, J. (2018). NPRF: A neural pseudo relevance feedback framework for ad-hoc information retrieval. In *Proceedings of EMNLP 2018* (pp. 4482-4491). <https://doi.org/10.18653/v1/D18-1478>.
- [9] Wang, X., Macdonald, C., Tonellotto, N., & Ounis, I. (2021). Pseudo-relevance feedback for multiple representation dense retrieval. In *Proceedings of ICTIR 2021* (pp. 297-306). <https://doi.org/10.1145/3471158.3472250>.
- [10] Yu, H., Xiong, C., & Callan, J. (2021). Improving query representations for dense retrieval with pseudo-relevance feedback. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM '21)* (pp. 3598-3602). <https://doi.org/10.1145/3459637.3482124>.
- [11] Blondel, V. D., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10), 2008. <https://doi.org/10.1088/1742-5468/2008/10/P10008>.

- [12] Cattaneo, A., Luschi, C., & Justus, D. (2025). Ground-Truth Subgraphs for Better Training and Evaluation of Knowledge Graph Augmented LLMs. arXiv preprint arXiv:2511.04473. <https://doi.org/10.48550/arXiv.2511.04473>.
- [13] Cai, D., Li, X., Ho, J. C.-S., Bing, L., & Lam, W. (2022). Retrofitting multilingual sentence embeddings with Abstract Meaning Representation. In Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (pp. 6456-6472). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2022.emnlp-main.433>.
- [14] Mizuki, S., & Okazaki, N. (2023). Semantic specialization for knowledge-based word sense disambiguation. In Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics (pp. 3457-3470). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2023.eacl-main.251>.
- [15] Xie, R., Liu, Z., Jia, J., Luan, H., & Sun, M. (2016). Representation learning of knowledge graphs with entity descriptions. Proceedings of the AAAI Conference on Artificial Intelligence, 30(1). <https://doi.org/10.1609/aaai.v30i1.10329>.
- [16] Wang, L., Zhao, W., Wei, Z., & Liu, J. (2022). SimKGC: Simple contrastive knowledge graph completion with pre-trained language models. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (pp. 4281-4294). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2022.acl-long.295>.
- [17] Li, Q., Zhong, Y., & Qin, Y. (2024). MoCoKGC: Momentum contrast entity encoding for knowledge graph completion. In Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (pp. 14940-14952). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2024.emnlp-main.832>.
- [18] Qiu, C., Qian, P., Wang, C., Yao, J., Liu, L., Fang, W., & Ng, E.-Y.-K. (2024). Joint pre-encoding representation and structure embedding for efficient and low-resource knowledge graph completion. In Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (pp. 15257-15269). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2024.emnlp-main.851>.
- [19] Papaluca, A., Krefl, D., Lensky, A., & Suominen, H. (2025). Contrastive language-entity pre-training for richer knowledge graph embedding. In C. Wallraven, C.-L. Liu, & A. Ross (Eds.), Pattern Recognition and Artificial Intelligence - ICPRAI 2024, Lecture Notes in Computer Science, vol. 14892 (pp. 233-246). Springer. https://doi.org/10.1007/978-981-97-8702-9_16.
- [20] Perozzi, B., Al-Rfou, R., & Skiena, S. (2014). DeepWalk: Online learning of social representations. In Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 701-710). Association for Computing Machinery. <https://doi.org/10.1145/2623330.2623732>.
- [21] Grover, A., & Leskovec, J. (2016). node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 855-864). Association for Computing Machinery. <https://doi.org/10.1145/2939672.2939754>.
- [22] Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., & Weinberger, K. Q. (2019). Simplifying graph convolutional networks. In Proceedings of the 36th International Conference on Machine Learning, Proceedings of Machine Learning Research, vol. 97 (pp. 6861-6871). PMLR. <https://proceedings.mlr.press/v97/wu19e.html>.

- [23] Xiong, W., Li, X. L., Iyer, S., Du, J., Lewis, P., Wang, W. Y., Mehdad, Y., Yih, W.-T., Riedel, S., Kiela, D., & Oğuz, B. (2021). Answering complex open-domain questions with multi-hop dense retrieval. In International Conference on Learning Representations. <https://arxiv.org/abs/2009.12756>.
- [24] Trivedi, H., Balasubramanian, N., Khot, T., & Sabharwal, A. (2023). 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). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2023.acl-long.557>.
- [25] Shao, Z., Gong, Y., Shen, Y., Huang, M., Duan, N., & Chen, W. (2023). Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. In Findings of the Association for Computational Linguistics: EMNLP 2023 (pp. 9248-9274). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2023.findings-emnlp.620>.
- [26] Rocchio, J. J., Jr. (1971). Relevance feedback in information retrieval. In G. Salton (Ed.), The SMART Retrieval System: Experiments in Automatic Document Processing. Prentice-Hall.
- [27] Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W. W., Salakhutdinov, R., & Manning, C. D. (2018). 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). Association for Computational Linguistics. <https://doi.org/10.18653/v1/D18-1259>.
- [28] Trivedi, H., Balasubramanian, N., Khot, T., & Sabharwal, A. (2022). MuSiQue: Multi-hop questions via single-hop question composition. Transactions of the Association for Computational Linguistics, 10, 539-554. https://doi.org/10.1162/tac1_a_00475.
- [29] Jiménez Gutiérrez, B., Shu, Y., Gu, Y., Yasunaga, M., & Su, Y. (2024). HippoRAG: Neurobiologically inspired long-term memory for large language models. In Advances in Neural Information Processing Systems. <https://arxiv.org/abs/2405.14831>.
- [30] Edge, D., Trinh, H., Cheng, N., Bradley, J., Chao, A., Mody, A., Truitt, S., Metropolitansky, D., Ness, R. O., & Larson, J. (2024). From local to global: A graph RAG approach to query-focused summarization. arXiv preprint arXiv:2404.16130. <https://doi.org/10.48550/arXiv.2404.16130>.

Appendix Overview

Appendix A — Graph and dataset. How the corpus was built: the generation procedure, the graph schema, the rendering of structured facts as text, the determinism guarantees, and the construction of the questions, answers, and gold-evidence sets.

Appendix B — Retrofitting strategies. The formulation of each embedding transformation, its parameter grid, and the implementation choices the main text states only in outline.

Appendix C — Retrieval strategies. The same for the retrievers: the stage schedules, their quotas, and the parameters that distinguish them.

Appendix D — Per-family results. Results for the six harder question families separately, whereas Section 4 reports them pooled into groups.

A Graph Construction and Evaluation Dataset

A.1 Graph Construction

The corpus is a fully synthetic professional network of fictional candidates, organisations, resumes, employer rosters, and textual evidence. Generation is deterministic under a fixed seed and does not involve a language model: every fact is sampled from predefined source pools and then rendered into text by fixed templates. Because the graph is built before any question exists, no answer information can enter its construction.

Candidate identity. A candidate’s name is a first–last pair drawn from two flat pools of 512 and 458 entries. A `used_names` set rejects any repeat and the draw is retried, so every person in the dataset has a distinct full name before the identity-collision checks performed at ingest. With roughly 2.3×10^5 available combinations and a few hundred candidates per graph, rejection is rare. Uniqueness is what allows the human-readable `entity:person:<Name>` identifier to serve as the canonical key: two candidates sharing a name would otherwise collapse into a single node and silently merge their evidence.

What is generated. Each candidate is assigned a primary sector sampled uniformly from eleven, which governs certifications and supplies the default source for employer and role attributes. A candidate receives one or two education records, between one and four jobs, zero to two projects per job, zero to three certifications, and zero to two endorsements. Education degrees are sampled with weights 50 % Bachelor’s, 35 % Master’s, and 15 % PhD, and universities come from a global pool that is not sector-scoped. Project descriptions reuse a subset of the skills attached to their job, so project evidence is correlated with the job rather than sampled independently.

Controlled cross-sector bridges. With probability 0.2 a job’s role attributes—title, skills, responsibilities, tenure, salary band, team size, and project name—are drawn from a different sector, selected through a fixed affinity map, while the employer remains tied to the candidate’s primary sector. This prevents the graph from collapsing into isolated sector-specific clusters and produces known bridges between otherwise sector-oriented regions, such as technology roles inside healthcare organisations.

Dual-perspective evidence. Employment information is represented from two document perspectives. Resume-side chunks describe one person’s education, jobs, projects, endorsements,

and certifications. After all candidate histories are generated, a second pass groups jobs by employer: any employer with at least two employees receives a roster document and a roster chunk listing every employee and title, which mentions the organisation and each employee individually. The same employment fact therefore appears both in a person’s job chunk and in an employer’s roster chunk, so information about one entity is distributed across documents connected only through shared entities. This is what makes the corpus suitable for graph-retrieval experiments rather than reducible to independent flat documents.

Schema. The graph is a plain JSON dictionary with three node types and three edge types. Every node exposes an `embed_field` naming the field embedded at ingest: `name` for entities and documents, `text` for chunks. The graph itself is therefore not embedded during generation.

Category	Type	Semantics
Node	<code>entity</code>	Person or organisation
Node	<code>document</code>	Resume or roster
Node	<code>chunk</code>	Textual evidence
Edge	<code>part_of</code>	Chunk $\rightarrow$ document
Edge	<code>about</code>	Document $\rightarrow$ entity
Edge	<code>mentions</code>	Chunk $\rightarrow$ entity

For each person the generator creates an `entity:person:<Name>` node and a `document:resume:<Name>` node joined by `about`. Every education, job, project, endorsement, and certification record becomes a chunk, linked to the resume by `part_of` and to the person by `mentions`. Education chunks also mention a university entity and job and project chunks an employer entity; organisation nodes are created on first use and reused thereafter. Identifiers are deliberately human-readable—`entity:person:<Name>`, `chunk:job:<Name>:<i>`—and these are the canonical evaluation identifiers, preserved downstream in the vector store and in the result artefacts.

The schema admits exactly three edge patterns and no others, so the graph is tripartite. One hop from a chunk reaches the entities it mentions and the document it belongs to; two hops return to other chunks through a shared entity. Figure 10 draws the schema and one instance from the validation corpus; the recurring chunk–entity–chunk route it shows is what the two-hop neighbour blend exploits.

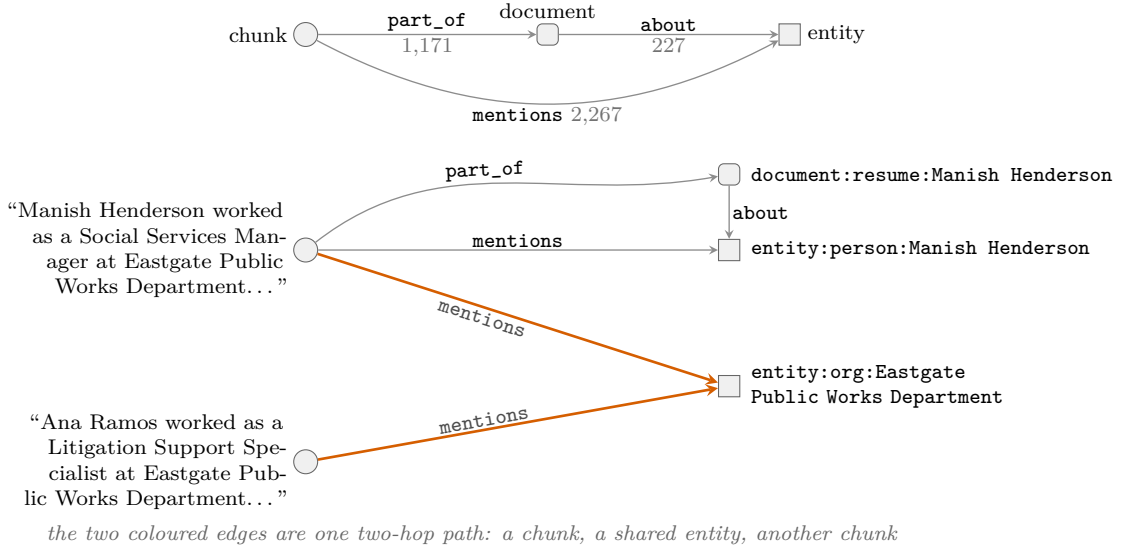


Figure 10: The graph schema and one instance from the validation corpus. Shape marks the node type: a circle is a chunk, a rounded square a document, a square an entity. Counts on the schema edges are those of the validation graph. In the instance, a resume document is **about** a person, its chunks are **part_of** it, and each chunk **mentions** the entities named in its text. Because **mentions** always runs from a chunk to an entity, two hops lead from one chunk to another through an entity they share — here two employees of the same organisation.

Structured facts and gold supervision. Every generated fact is retained in two synchronised forms. Graph edges encode its structural relationships, and a human-readable rendering is stored in the corresponding chunk; the originating structured dictionary is preserved unchanged on that chunk as **source**. Text is therefore the representation exposed to embedding and retrieval, while **source** is the authoritative representation used to compute gold answers and supporting evidence. Question generation reads the structured copy, so exact gold evidence is obtained without an LLM reconstructing facts from prose.

Determinism. The generator seeds Python’s global random module once with `SEED = 42` and proceeds strictly sequentially. The one hash-order-sensitive operation—the union of skills across a candidate’s jobs, used to sample endorsements—is sorted before sampling, so process-dependent string hashing cannot affect the output. Under the same configuration and source pools the resulting `graph.json` is byte-identical across runs and machines, a property checked by `tests/test_generation_determinism.py` under different interpreter hash seeds.

Development, validation, and test graphs. A separate generation mode partitions the source pools themselves into disjoint development, validation, and test vocabularies using `SPLIT_SEED = 7`. Names, attributes, employers, and universities are split before the three graphs are generated, in the order development, validation, test, from a single seeded stream and a single shared `used_names` set. No person name is therefore repeated across the three graphs, and the later graphs are not a mirrored copy of the earlier draws. Hyperparameter selection, generalisation checks, and final reporting do not share the same underlying vocabulary, and the corpora are node-disjoint by construction rather than by filtering.

Validation corpus. The validation graph used throughout this paper contains 1,759 nodes and 3,665 edges.

Component	Count	Composition
Entities	361	133 persons, 228 organisations
Documents	227	133 resumes, 94 rosters
Chunks	1,171	326 project, 319 job, 195 education, 133 endorsement, 104 certification, 94 roster
mentions edges	2,267	chunk $\rightarrow$ entity
part_of edges	1,171	chunk $\rightarrow$ document
about edges	227	document $\rightarrow$ entity

Each node carries an original embedding in $\mathbb{R}^{1536}$, and every modified collection preserves the same node identities, so a retrieved identifier means the same thing in every collection.

A.2 Questions and Gold Answers

Each question is paired with an exact answer and with the evidence required to derive it. Generation is deterministic under a fixed seed and uses no language model: Python functions traverse the graph, read the structured fact preserved in a chunk’s `source` field, instantiate a fixed template, and record the nodes touched along the way. The generated chunk text is never parsed to recover an answer—text is what retrieval sees, while the structured record defines the ground truth.

The stored record. Every generator returns the triple (q, C_q, a_q) , where q is the natural-language question, C_q is the set of graph nodes touched while deriving the answer, and a_q is the answer computed from structured facts. Before persistence C_q is filtered to the evidence-bearing chunks, giving the gold-evidence set G_q , so each stored example is $(q, a_q^{\text{gold}}, G_q^{\text{gold}})$.

Why only chunks count as targets. Filtering keeps only nodes whose type is in `GOLDEN_NODE_TYPES = ("chunk",)`. Entities named in the question are query inputs, not retrieval targets: the retriever already has them. What it must recover is the text that carries the evidence. Rewarding a system for returning an entity the question itself names would measure nothing. Filtering happens at record-writing time only, and changes neither the question text, nor the computed answer, nor the sampling or reasoning depth.

Question families. Eleven families span basic, compositional, and multi-hop reasoning. Basic families retrieve or aggregate local attributes, compare candidates, or select extrema; compositional families chain two basic operations and union their evidence; multi-hop families traverse explicit person–organisation–university relations over one, two, or three hops, alternating between two predefined chains at each level. Because every traversal step is recorded, the benchmark keeps the whole supporting chain and not only the node holding the final answer.

Family	Main operation	Val.
<code>collect_local</code>	Read one attribute for one entity or entity–organisation pair	90
<code>collect_global</code>	Aggregate one attribute across all people linked to an organisation	90
<code>comparison</code>	Compute a set intersection for two people	90
<code>extremize</code>	Select an argmax over job facts, optionally filtered and grouped	77
<code>extremize_collect</code>	Find an extremal person, then retrieve a second attribute	90
<code>collect_extremize</code>	Restrict to an organisation’s people, then apply an extremum	90
<code>collect_global_collect</code>	Traverse through an organisation and retrieve a second attribute for each linked person	90
<code>multi_collect</code>	Retrieve two attributes for one person in one question	90
<code>hop1</code>	One-hop organisation/person or university/person reasoning	90
<code>hop2</code>	Two-hop chains crossing entity types	90
<code>hop3</code>	Three-hop chains with aggregation or peer discovery	90
Validation set		977

One question of each family, taken from the validation set.

Family	Example question
<code>collect_local</code>	“What accreditations does Ethan Ito have?”
<code>collect_global</code>	“What degrees has Union College awarded?”
<code>comparison</code>	“Were Naomi Das and Emily Hong ever at the same company?”
<code>extremize</code>	“Which candidate held a single position for the most years?”
<code>extremize_collect</code>	“Which credentials has the most experienced grant management practitioner earned?”
<code>collect_extremize</code>	“Who worked at Inkerman & Price for the most years?”
<code>collect_global_collect</code>	“What certifications do employees of Juniper Health hold?”
<code>multi_collect</code>	“What was Rajan Cook’s tenure and what skills did they use at Xcel Freight Co?”
<code>hop1</code>	“Who attended University of Kentucky?”
<code>hop2</code>	“Where did people who worked at Jettison Labs study?”
<code>hop3</code>	“Which employer of Yale University graduates has the most total employees?”

Configuration-driven generation. Most basic families are parameterised rather than implemented separately. A configuration names the relevant chunk and entity prefixes, whether an organisation is required, one or more surface templates, and an `answer_fn` that extracts the target value from `source`. A generic routine resolves the linking chunk, applies the extraction function, and fills a template. Question semantics are therefore separated from traversal logic, and a new subtype costs a configuration entry rather than new generation code. Multi-hop generators are the exception: each chain is implemented directly, because successive hops cross different entity types.

Sampling and budget. Candidate tuples come from type-specific samplers—person–organisation, person pair, organisation, extremisation—each relying on seeded shuffling. Families with several subtypes cycle through their configurations so that consecutive questions alternate subtype until the budget is met or the candidate pool is exhausted. The nominal budget is divided evenly across the eleven families, so a request for 1,000 questions asks for 90 of each.

Two families cannot fill that quota, and the reason is structural rather than incidental. `extremize` realises 77 on the validation graph, because an argmax needs a competition to resolve. `collect_global_collect` realises a single question, because some of its subtypes require one

organisation to satisfy more than one semantic role at once, and few organisations in a single graph do. The validation set reported in this paper therefore takes the 888 questions realised on the validation graph and completes the under-filled family with 90 `collect_global_collect` questions generated on a second corpus from the same pipeline, for 977 in total.

Gold-evidence set size. Gold-evidence set size reflects the number of independent facts needed to establish an answer, not where the answer value happens to sit. A local extraction is usually one chunk: `q0000` asks how long a person worked at an organisation and its gold-evidence set is the single job chunk holding that record. Aggregation scales with the organisation: a question about tenure lengths across a company’s staff carries one job chunk per employee. An argmax carries its competitors—`q0180` asks who is most experienced in a given skill and its gold-evidence set is the 15 job chunks that mention it, because the maximum cannot be verified without them. Compositional questions union the two steps’ evidence, and three-hop questions produce the largest sets, up to 15 chunks spanning employer-to-employee, employee-to-university, and university-to-peer links.

Determinism and materialisation. Sampling and template choice draw from Python’s global random generator, seeded once before generation. Each question receives a stable identifier of the form `q0000`, and `questions.jsonl` stores one deterministic JSON record per question: answer sets are serialised as sorted lists and gold-evidence node identifiers are sorted, so downstream evaluation joins on persistent identifiers rather than on text order or container iteration. Question embeddings are held in a separate vector collection, produced by the same embedding engine used for the graph-node collections, so query and document vectors share an embedding model by construction. Questions are stored as a plain model rather than as a graph data point, which keeps them out of any retriever’s default collection scan.

B Retrofitting Strategies: Formulations and Implementation Details

The following sections consolidate the recorded implementation and parameter choices. Counts refer to generated collections, not all combinations with all retrievers. Unless otherwise stated, the blending coefficient retains the original representation with weight α .

B.1 One-Hop Neighbour Blend

`hop1_neighbor_blend` averages a subset of the node’s direct neighbours and mixes that mean with the original vector.

Parameter	Meaning	Values
<code>neighbor_pool_share</code>	Fraction of available direct neighbours used	0.2, 0.5, 1.0
<code>alpha</code>	Weight retained from the original vector	0.0, 0.2, 0.5
<code>neighbor_pick</code>	Select least- or most-similar eligible neighbours	<code>least_similar</code> , <code>most_similar</code>
<code>min_neighbors</code>	Minimum selected count when the fraction is small	2 (<i>fixed</i>)

The share is degree-relative: at 20%, a node with 60 neighbours uses 12, while a node with five uses the minimum of two. A full Cartesian grid would contain 18 combinations. At share 1.0, however, both selection orders include the entire pool and produce byte-identical outputs; only one copy is retained, leaving 15 collections.

B.2 Two-Hop Neighbour Blend

`hop2_only_neighbor_blend` uses nodes at graph distance exactly two, excluding the source and its direct neighbours. It uses the same share, α , selection, and minimum-neighbour grid as the one-hop neighbour blend. The additional parameter `neighbor_vectors` is fixed to `original`: the method always averages the original embeddings of two-hop nodes. Blending already-modified neighbour vectors was deliberately not implemented because its interaction with share-based selection was not specified.

Along chunk–entity–chunk paths, two hops often map a chunk to another chunk through a shared entity. The method therefore captures other texts that discuss the same entities rather than the entities themselves.

B.3 Convex Neighbour Pull

This anchored neighbour-attraction formulation is related to retrofitting [3], applied here to corpus-node embeddings. `convex_neighbor_pull` jointly adjusts node embeddings by balancing fidelity to their original textual representations against agreement between graph neighbours. A useful physical interpretation is a system of springs: each node is attached to its original position and to every adjacent node. The modified embeddings are the equilibrium of this coupled system.

Objective and uniqueness. Let $\mathbf{x}_i$ be the original embedding and $\mathbf{z}_i$ the unknown modified embedding. Let E contain unordered edge pairs, each counted once, and let $N(i)$ be the corresponding neighbour set. The objective is

$$J(\mathbf{Z}) = a \sum_i \|\mathbf{z}_i - \mathbf{x}_i\|_2^2 + b \sum_{(i,j) \in E} \|\mathbf{z}_i - \mathbf{z}_j\|_2^2. \quad (7)$$

Here $a = \text{anchor_weight} > 0$ and $b = \text{edge_weight} \geq 0$. We use a rather than α to distinguish this penalty weight from the blending coefficient used by other transformations. Squared Euclidean distances yield a differentiable quadratic objective with linear stationarity equations. The positive anchor term makes the objective strictly convex, so its minimiser is unique. The implementation rejects nonpositive anchor weights. With $a = 0$ and $b > 0$, any embedding configuration constant within each connected component has zero energy; the minimiser is no longer unique.

Closed-form coordinate update. Setting the derivative with respect to one node to zero gives

$$a(\mathbf{z}_i - \mathbf{x}_i) + b \sum_{j \in N(i)} (\mathbf{z}_i - \mathbf{z}_j) = 0, \quad (8)$$

and hence the weighted-average update

$$\mathbf{z}_i \leftarrow \frac{a\mathbf{x}_i + b \sum_{j \in N(i)} \mathbf{z}_j}{a + b|N(i)|}. \quad (9)$$

The original vector has weight a , while each neighbour has weight b . Thus, total graph influence grows with degree: the denominator is $a + 3b$ for three neighbours and $a + 60b$ for sixty. An isolated node remains at its original vector. Although each coordinate update has a closed form, the global solution requires solving the coupled system.

Gauss–Seidel solver. The solver initialises $\mathbf{Z}$ with the original embeddings and visits nodes in sorted identifier order. Each update is written immediately, so later nodes in the same sweep use the newest available neighbour vectors. This is Gauss–Seidel iteration, rather than a synchronous Jacobi update based entirely on the previous sweep. It requires no learning rate or gradient clipping, although the objective weights and stopping controls remain configuration choices. For this anchored quadratic, convergence is to the same unique minimiser; finite-sweep results and convergence speed can depend on update order. Sorted identifiers ensure a reproducible order.

Stopping rule. During sweep t , the solver records the largest Euclidean displacement over all node updates,

$$\delta_t = \max_i \|\mathbf{z}_i^{\text{after update}} - \mathbf{z}_i^{\text{before update}}\|_2. \quad (10)$$

The displacement is measured before overwriting each vector. Convergence is declared when $\delta_t < 10^{-6}$; the criterion uses the maximum rather than the mean, so every node must satisfy the tolerance. On the reported development graph of 1,811 nodes, convergence took approximately forty sweeps, with a final maximum displacement of 9×10^{-7} . A safety cap of 200 sweeps returns the current configuration even if convergence has not been reached; the manifest records the sweep count and final displacement so that incomplete convergence remains visible. A separate early-stop setting terminates after a prescribed number of sweeps.

Parameter	Meaning	Values
<code>anchor_weight</code>	Strength of the original-vector fidelity term	1.0 (<i>fixed</i>)
<code>edge_weight</code>	Strength of attraction along graph edges	0.5, 1.0
<code>convergence_tol</code>	Stop when vector movement falls below this threshold	10^{-6} (<i>fixed</i>)
<code>max_sweeps</code>	Safety cap on full update sweeps	200 (<i>fixed</i>)
<code>stop_after_sweeps</code>	Optional forced early stop	None, 1, 2, 3

Grid construction. Multiplying both weights by the same positive constant rescales the objective without changing its minimiser or update rule. Only their ratio matters, so $a = 1$ fixes the reference scale. The grid contains two runs to convergence, with $b \in \{0.5, 1.0\}$, and three early-stop probes at $b = 1.0$ after one, two, or three sweeps. These are five configurations, not the full Cartesian product of the listed values. `stop_after_sweeps=None` uses the convergence criterion subject to the safety cap.

Early-stopping probe. The probe tests whether stopping short of the optimum preserves useful structure that full smoothing might remove. In the reported comparison, F1 increases across the tested sweep counts and is highest at convergence:

	1 sweep	2 sweeps	3 sweeps	Converged
F1	0.0933	0.1051	0.1120	0.1178

These measurements provide no evidence of a benefit from the tested early stops. They do not establish monotonic retrieval improvement at every unmeasured iteration.

Retrieval behaviour. The separately reported usage comparison gives F1 of 0.1202 for standalone retrieval and 0.1510 when this representation is used in the single requery schedule. A random-walk endpoint configuration scores 0.1534, a numerical difference of 0.0024, without an iterative convergence phase. This is a comparison of the supplied scores, not a claim of statistical equivalence. It illustrates that minimising the graph-smoothing objective and maximising

retrieval quality are different goals. The reported ranks (491 and 25 in an approximately 600-entry comparison) are omitted because the exact ranking population is not specified relative to the configurations documented here.

B.4 Random-Walk Endpoint Blend

For each node i whose embedding is being modified, `walk_endpoint` launches 100 random walks starting from that same node i . It forms the graph-context vector for i as a frequency-weighted mean of the endpoint embeddings; intermediate paths are discarded.

Parameter	Meaning	Values
<code>walk_steps</code>	Number of graph transitions per walk	2, 3
<code>alpha</code>	Original-vector retention	0.0, 0.5, 0.8
<code>walks_per_node</code>	Walks launched from each node	100 (<i>fixed</i>)
<code>walk_seed</code>	Reproducibility seed	42 (<i>fixed</i>)

Along alternating chunk–entity paths, two transitions can return to a chunk and three can end on an entity. Other node and edge types may change this pattern. Returns to the starting node are possible unless excluded by the transition rule; that rule remains to be verified.

B.5 Louvain Cluster Blend

Community detection uses the Louvain family introduced by Blondel et al. [11]; the subsequent embedding blend is the transformation evaluated here. `louvain_cluster` detects graph communities and moves each node towards the centroid of its community.

Parameter	Meaning	Values
<code>resolution</code>	Controls community granularity; larger values produce smaller communities	0.5, 1.0, 2.0
<code>alpha</code>	Original-vector retention	0.0, 0.5, 0.8
<code>louvain_seed</code>	Reproducible partitioning seed	42 (<i>fixed</i>)

At resolutions 0.5, 1.0, and 2.0, the graph contains respectively 20, 34, and 50 communities, with median sizes 85, 44, and 31. When $\alpha = 0$, every member is mapped to its frozen original-space community centroid, eliminating within-community distinctions.

B.6 Shared-Neighbour Pull

`shared_neighbor_pull` adds an attraction term between nodes that are not directly connected but share graph neighbours.

Parameter	Meaning	Values
<code>shared_pair_weight</code>	Strength of attraction between common-neighbour pairs	1.0, 3.0, 9.0
<code>stop_after_sweeps</code>	Optional forced early stop	None, 1, 2, 3
<code>anchor_weight</code>	Original-vector fidelity term	1.0 (<i>fixed</i>)
<code>edge_weight</code>	Direct-edge attraction term	1.0 (<i>fixed</i>)
<code>convergence_tol</code>	Convergence threshold	10^{-6} (<i>fixed</i>)
<code>max_sweeps</code>	Safety cap	200 (<i>fixed</i>)

The coefficient values are 1, 3, and 9 with direct-edge weight fixed to 1. Their aggregate influence also depends on the number and weighting of eligible shared-neighbour pairs. Three

configurations converge fully; three early-stop probes use the weight-3 setting. Fixing the edge weight to one makes `convex_neighbor_pull__e1p0` an exact zero-shared-pair control under the same solver.

B.7 Gradient-Based Neighbour Pull

`gradient_neighbor_pull` optimises embeddings so that vector similarity follows graph-hop distance.

Parameter	Meaning	Values
<code>hop_target_curve</code>	Desired similarity decay over hop distance	None, <code>linear_steep</code> , <code>hyperbolic_sech</code>
<code>step_size</code>	Magnitude of each optimisation update	0.25, 0.5, 1.0
<code>max_shift</code>	Maximum displacement as a fraction of vector norm	0.5, 1.0, 2.0
<code>steps</code>	Number of optimisation passes	30 (<i>fixed</i>)
<code>patch_hops</code>	Radius of overlapping graph patches	2 (<i>fixed</i>)
<code>hop_target_max_hop</code>	Hop distance beyond which target similarity is zero	4 (<i>fixed</i>)
<code>patch_seed</code>	Reproducibility seed	42 (<i>fixed</i>)

The grid is intentionally sparse: one default configuration uses no explicit curve, step 0.5, and shift 1.0; the two named curves are crossed with two step sizes and two shift limits, producing eight additional configurations. This is the only transformation with a learning-rate-like step and an explicit displacement cap.

B.8 Residual Representations

A residual has no independent hyperparameters. For each of the 22 principal swept collections, it subtracts the normalised original vector from the normalised modified vector. The sole parameter, `modified_collection`, selects the source collection.

C Retrieval Strategies: Formulations and Implementation Details

Every retriever below is evaluated under the invariant final budget $B = 40$, and each is crossed with the derived collections. Two accessory grids re-run part of the shortlist at half the budget and over a chunk-only search domain.

C.1 Graph-Agnostic Embedding Retrieval and One Modified Collection Alone

`MergedSearchRetriever` performs standard nearest-neighbour retrieval over a chosen set of stored collections.

Parameter	Meaning	Values
<code>collections</code>	The index set searched	the three originals jointly, or one derived collection

One run searches the three original collections jointly and defines `plain_baseline`; the others search one derived collection directly.

C.2 Alternating Budget Split

BudgetSplitRetriever searches the original and the modified space independently, then allocates the final positions between the two rankings.

Parameter	Meaning	Values
beta	Share of the final budget given to the modified space	0.5, 0.2

At $\beta = 0.5$, 20 positions go to each space; at $\beta = 0.2$, 8 go to text and 32 to the modified space. Thirty hop-derived collections are crossed with both values, and the fixed-count collection is evaluated once at $\beta = 0.5$.

C.3 Reranking

RerankRetriever retrieves a deep pool from the original space and rescores it with a graph-derived signal, scoring each candidate as

$$s_i = (1 - \text{mix}) \text{sim}(\mathbf{q}, \mathbf{x}_i) + \text{mix} \text{sim}(\mathbf{m}, \mathbf{z}_i). \quad (11)$$

Parameter	Meaning	Values
mix	Weight of the graph-derived score against the original-space score	0.3, 0.5, 0.7
RERANK_OVERSAMPLE	Candidate pool as a multiple of the budget	4 (<i>fixed</i>)

The 22 modified collections are crossed with the three mix values. Mix zero is excluded because it exactly reproduces the text-only ranking. The oversampling factor fixes the pool at $4B = 160$, and the seed count is derived as $B/2 = 20$; neither is treated as a tunable parameter.

C.4 Two-Stage Fused Retrieval and Requery Schedules

MultiStageRetriever fills the budget sequentially, one stage at a time. Seven stage schedules are evaluated:

Quotas	Stage ranks	Interpretation
16/12/12	original, fused, shifted	Single requery: expansion followed by query-shifted text retrieval
6/6/14/14	original, fused, shifted, shifted	Chained requery: a short opening, then two successive shifts
20/20	original, fused	Two-stage fused retrieval: text seeds followed by modified-space expansion
16/8/8/8	original, fused, shifted, shifted	The same chaining from a wide opening
6/12/22	original, fused, shifted	Single requery from a short opening
6/6/14/14	original, fused, fused, fused	The chained quotas with every later stage fused instead of shifted
20/20	original, shifted	Graph-free control: the shift alone, with no fused stage

Parameter	Meaning	Values
mix	Weight of the modified-space score in a fused stage	0.3, 0.5, 0.7
query_shift	Weight of the previous stage’s mean in the shifted query	0.3, 0.5 (bridge only)

The shifted query is

$$\mathbf{q}' = (1 - \text{query_shift})\mathbf{q} + \text{query_shift} \bar{\mathbf{x}}_{\text{previous}}. \quad (12)$$

The coefficient is a nominal weight: the norm of the mean determines its effective contribution, so a coefficient of 0.5 need not produce equal directional influence. The chained schedule carries two parameters the others do not: beyond the mix and query-shift grids it also varies which retrieved nodes a shifted stage anchors on and how far back it looks. The modified-space input ranges over 44 representations: 22 graph-modified collections and their 22 residuals.

C.5 Output Ordering, Overlap, and Computational Scope

Single requery (**bridge**) is formulated in Section 3.4. Its final output concatenates the three stage lists; it is not globally reranked. Stage-level exclusion uses node identity. Candidate-depth and refill rules must ensure that an implementation actually returns 40 distinct nodes, including when independent searches overlap. For the alternating budget split, the quota and overlap-resolution convention must be recovered from the implementation; taking two top lists alone does not guarantee 40 distinct nodes.

The recorded search-call counts are one for plain retrieval, two for budget splitting, one for reranking, three for two-stage fused retrieval, four for single requery, and five for the four-stage chained schedules. They count vector-search operations rather than equal units of work. Reranking evaluates 160 candidates; modified collections require preprocessing and storage. No latency or hardware measurements are available.

D Per-Family Results

Section 4 reports the question families pooled into groups. The table below gives each of the six harder families separately, for the retrieval strategies described in Section 3.4. The five single-fact families are omitted: the baseline already recovers between 96.4 % and 100 % of their evidence, so no strategy can separate itself on them except by degrading them. Each column uses the representation the main text pairs with that schedule: the hop-2 neighbour blend for the alternating budget split and for two-stage fused retrieval, the walk-endpoint blend for both requery schedules. The baseline column is plain vector retrieval on the original collections.

Group	Family	Gold chunks found (%)				
		Base	Split hop-2	Fused	Single	Chained walk
Extremisation	collect_extremize	72.4	71.6	86.6	87.5	88.1
	extremize	61.9	42.5	54.5	68.4	75.0
	extremize_collect	44.7	32.4	40.6	59.7	61.7
Relational and collection	collect_global_collect	66.4	85.7	80.3	70.0	71.6
	hop2	42.3	64.7	48.5	46.7	50.0
	hop3	20.1	31.9	22.7	20.6	20.2

Two patterns are visible here that the group-level reporting averages away.

The first is that the two leading strategies are complementary rather than ranked. On the three extremisation families the chained requery is best everywhere, by 13.2 to 17.0 points over the baseline; on the three relational and collection families the order reverses and the split is best on all three, by 11.8 to 22.4 points. Which strategy helps therefore depends on the operation a

question requires — covering a comparison set, or reaching along a relational chain — and not on a single ordering of methods.

The two failures are not symmetric, and the difference matters for anyone who must choose one schedule for a mixed workload. Outside its own group the split is actively harmful: it falls below the baseline on all three extremisation families, by 0.8, 12.3 and 19.3 points. The chained requery outside its own group is merely modest: it stays above the baseline on all three relational and collection families, by 5.2, 7.6 and 0.1 points, the last of which is indistinguishable from zero. Choosing the split therefore costs real evidence on the questions it does not suit, whereas choosing the chained schedule costs almost nothing.

The second is that **hop3** resists every strategy. Its baseline is the lowest in the benchmark at 20.1 % and the best column improves it by 11.8 points, while both requery schedules leave it where it was. The limit is not the budget: 40 retrieved nodes are ample for a median gold-evidence set of twelve chunks, and doubling the budget from 20 to 40 nodes gains the baseline only 1.6 points, because the extra nodes repeat content the ranking has already exhausted.