

Self-improving data synthesis can scale embedding models across domains and tasks

Zucan Lyu^{1,2†}, Rui Tang^{1,2†}, Jianlyu Chen^{1,2†}, Peisheng Jin^{1,2},
Zheng Liu^{1,2}, Defu Lian^{1,2*}

¹University of Science and Technology of China, Hefei, China.

²State Key Laboratory of Cognitive Intelligence, Hefei, China.

*Corresponding author(s). E-mail(s): liandefu@ustc.edu.cn;

Contributing authors: zucanlv@mail.ustc.edu.cn;

[†]These authors contributed equally: Zucan Lyu, Rui Tang, Jianlyu Chen.

Abstract

Adapting embedding models to new tasks requires training data that reflect task-specific information needs and relevance criteria. However, manual data design is costly and difficult to scale. Here we introduce SYNTRA, a novel retrieval data synthesis method whose self-improvement loop iteratively refines reusable synthesis instructions using feedback on trial outputs. This process requires only an unlabelled corpus and a few relevant query–document pairs. Experiments across 24 retrieval datasets spanning 15 domains demonstrate the effectiveness of the synthesized data. These data also improve widely used, strong embedding models on tasks with and without existing labelled training sets. Self-improvement enhances retrieval on all six evaluated tasks, outperforming human calibration on four. Moreover, retrieval performance continues to improve on MS MARCO as we scale the training dataset to 32 times its initial size. Overall, with minimal human intervention, SYNTRA enables scalable adaptation of embedding models across domains and tasks.

Keywords: self-improving, data synthesis, text embeddings, information retrieval, large language models

Artificial intelligence (AI) agents increasingly rely on information retrieved from external resources to guide reasoning and action. Cursor combines a specialised embedding model with conventional search tools to locate relevant code (<https://cursor.com/blo>

[g/semsearch](#)). OpenScholar retrieves and integrates evidence from scientific literature to answer research questions [1]. Across domains and tasks, agents’ use of external knowledge depends on retrieving information that meets each task’s needs (Fig. 1a).

New domains and applications continually create demand for retrieval models tailored to new corpora and information needs. During the COVID-19 pandemic, rapidly expanding scientific literature and evolving search needs motivated the development of TREC-COVID [2]. More recently, OpenScholar adapted its retriever to scientific literature [1], while Cursor trained a specialised embedding model for code retrieval. Such adaptation must account for differences in domain knowledge and task-specific relevance criteria. Even within the same corpus, different user objectives can change which documents are relevant [3] (Fig. 1b). Embedding models must therefore be adapted rapidly and efficiently as domains and information needs change.

A key bottleneck in this adaptation is constructing training data that reflect both domain knowledge and task-specific relevance criteria. Suitable training data are often scarce, even when unlabelled corpora are available [4, 5]. To learn task-relevant distinctions, embedding models need queries paired with relevant documents and informative negatives [6, 7]. Constructing these examples requires deciding what users might ask, which documents satisfy their needs and which plausible alternatives should be rejected. These decisions depend on the domain and task, making data construction a recurring challenge as retrieval is adapted to new settings.

Generative AI offers a way to turn unlabelled corpora into retrieval supervision [8, 9]. Large language models (LLMs) can generate queries [10–18] and assess document relevance [19–21]. These capabilities have enabled the construction of effective training datasets for general-purpose retrieval [19, 22–26], code retrieval [27] and reasoning-intensive document retrieval [28, 29]. Yet constructing such data can still depend on task-specific human preparation and design. This human work includes defining retrieval tasks and relevance criteria, screening proposed tasks, and refining generation and annotation instructions [11, 14, 17, 27, 29]. Repeating these activities across retrieval tasks makes human effort a bottleneck to scaling retrieval adaptation. The central question is: Can an AI system infer task requirements and adapt data synthesis with minimal human intervention, enabling retrieval adaptation to scale across domains and tasks?

Here we introduce SYNTRA, a self-improving agentic framework for retrieval data synthesis. With minimal human intervention, SYNTRA enables scalable adaptation of embedding models across domains and tasks. Given an unlabelled corpus and few-shot examples pairing queries with positive documents, SYNTRA infers task requirements and develops initial synthesis instructions. Through few-shot-guided self-improvement, SYNTRA evaluates trial outputs against these requirements and uses the resulting feedback to refine the shared instructions. This process aligns data synthesis with task-specific information needs and relevance criteria without requiring manual refinement of synthesis instructions for each task. The refined instructions are then reused for large-scale data production (Fig. 1c). By turning few-shot guidance into reusable synthesis instructions, SYNTRA makes adaptation of embedding models scalable across domains and tasks.

Experiments covering 24 retrieval datasets spanning 15 domains show that the same SYNTRA workflow produces synthetic training data that improve the retrieval performance of embedding models. Ablations across six tasks show that automatic

self-improvement enhances the training value of the generated data and provides an effective alternative to human calibration of synthesis instructions. The generated data also improve widely used, strong embedding models, supplying task-specific supervision where labelled training sets are unavailable and complementing existing supervision where they are available. On MS MARCO, retrieval performance continues to improve as we scale up SYNTRA-generated training data to 32 times the initial size.

SYNTRA

Here we briefly introduce the conceptual framework of SYNTRA, with formal definitions and implementation details deferred to Methods, and complete prompt templates for the principal synthesis workflow provided in Supplementary Note 1.

Framework

SYNTRA uses few-shot-guided self-improvement to adapt the instructions governing a shared data synthesis workflow for retrieval (Figs. 2 and 3). Adaptation takes place in *test mode*, where feedback on trial outputs guides revisions to shared instructions. In *production mode*, these instructions are reused to construct training data across the corpus.

The user supplies an unlabelled corpus and a small set of few-shot examples pairing queries with relevant documents (Fig. 2a). Corpus preprocessing selects diverse source documents for query generation and builds a full-corpus index for candidate retrieval. SYNTRA uses the examples to infer a task definition specifying the intended query type, document type and relevance criteria. It checks and refines this definition against the examples, establishing common task requirements for the synthesis modules. The resulting definition guides the initial instructions for query synthesis and relevance-label synthesis.

In *test mode*, SYNTRA evaluates whether trial outputs follow the task definition and the patterns conveyed by the few-shot examples (Fig. 2b). Diagnostic feedback guides revisions to the shared instructions, including how queries express information needs and how documents are judged for relevance. The modules are tested again on small samples with the revised instructions. Because self-improvement targets shared synthesis rules, revisions made during testing guide later outputs across documents. Once the test loops finish, the final instructions remain fixed throughout production.

In *production mode*, three synthesis modules apply these task-adapted instructions to the selected documents. Identifier extraction and attribute construction identify document-grounded entities or relations and establish dimensions for varying the information need and its expression (Fig. 3a). Query synthesis combines an identifier with attribute options to plan an information need, then expresses it as a query (Fig. 3b). Relevance-label synthesis retrieves candidates from the full corpus and assesses them against the task’s relevance criteria, producing positives, hard negatives and easy negatives (Fig. 3c). The source document receives the same relevance assessment if it is retrieved. The resulting (query, positive, negative) records provide the supervision used to train embedding models after synthesis.

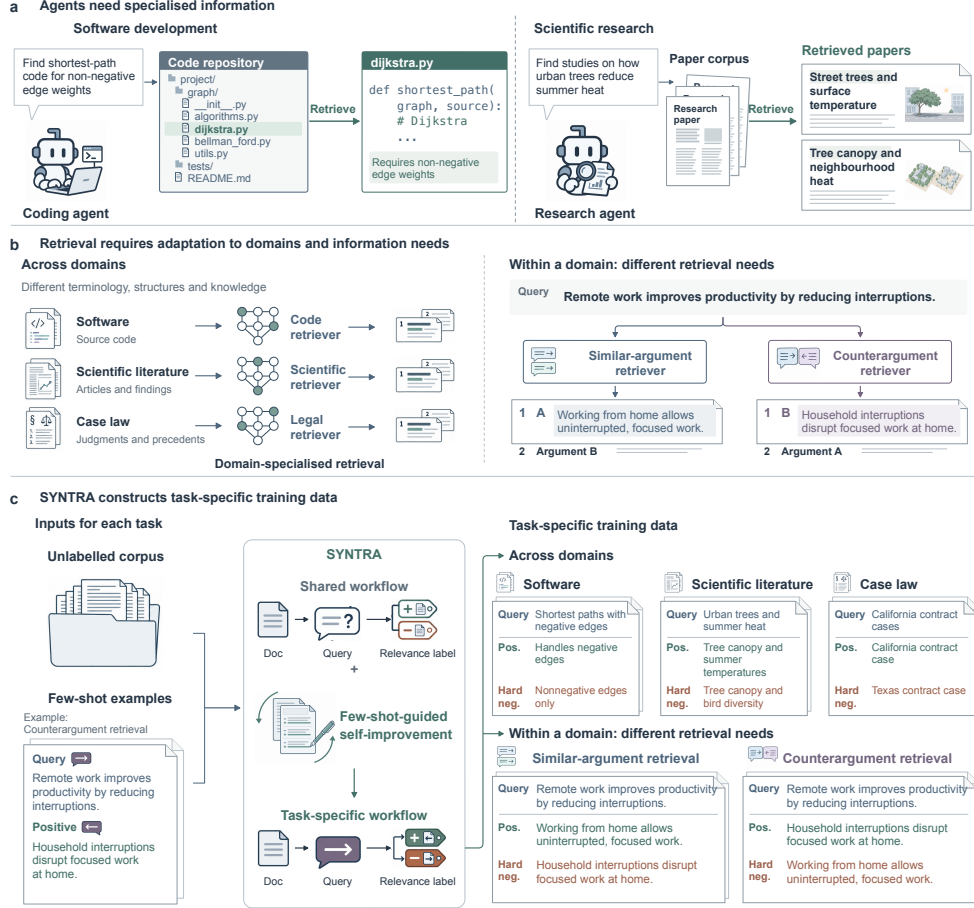


Fig. 1 Constructing task-specific training data for specialised retrieval. **a**, A coding agent searches a repository and retrieves the implementation that satisfies a stated constraint. A research agent retrieves supporting studies from a paper corpus. **b**, Documents differ across domains in terminology, structure and knowledge. Within a single domain, the user’s objective changes which documents are relevant. For the same query, similar-argument and counterargument retrieval rank the same two arguments in opposite orders. **c**, SYNTRA receives an unlabelled corpus and a small set of few-shot examples, each pairing a query with a positive document. Few-shot-guided self-improvement turns a shared synthesis workflow into a task-specific one. This workflow then constructs (query, positive, negative) training records at scale. SYNTRA is run separately for each retrieval task using the same framework. The examples shown span software, scientific literature and case law. Within a domain, the same two arguments exchange positive and hard-negative roles under different retrieval needs. Pos., positive document; Hard neg., hard-negative document. All examples are schematic and abbreviated, not generated samples or measured retrieval results.

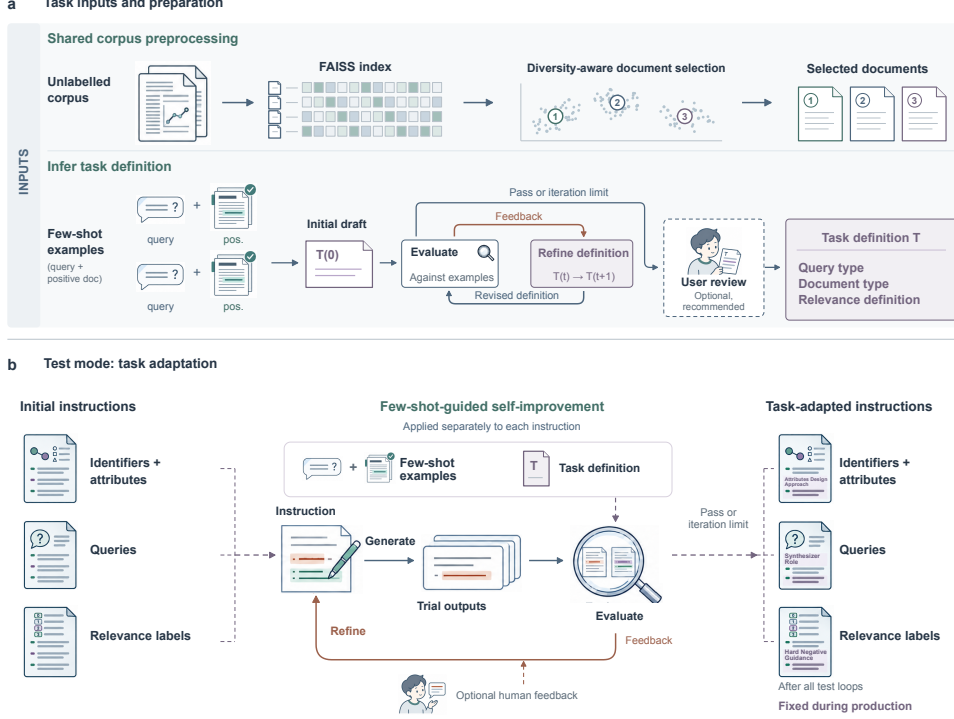


Fig. 2 Task preparation and adaptation in SYNTRA. **a**, SYNTRA receives an unlabelled corpus and a small set of few-shot examples pairing queries with positive documents. Corpus pre-processing builds a full-corpus index and selects diverse source documents. The task definition is inferred from the few-shot examples; optional user review is recommended. Point groups schematically illustrate semantic diversity, not clustering outputs. **b**, Small-sample generate-evaluate-refine cycles establish reusable task-adapted instructions before production. Each test loop ends when verification passes or its iteration limit is reached. The shared instructions then remain fixed throughout production.

1 Results

1.1 One synthesis workflow improves embeddings across domains and tasks

Our experiments covered 24 retrieval datasets spanning 15 domains (Fig. 4a). SYNTRA generated task-specific training data using the same synthesis workflow. Few-shot-guided self-improvement in test mode adapted the shared synthesis instructions, with human involvement limited to task setup (Section 3.2). In the main cross-task evaluation, we compared models trained on SYNTRA-generated data with an MS MARCO-trained baseline [30], using the same embedding backbone with LoRA adaptation. We selected MS MARCO as a common reference because it is widely used for supervised training of dense retrievers [31–33]. We report retrieval performance as normalised discounted cumulative gain at rank 10 (nDCG@10), on a 0–100 scale.

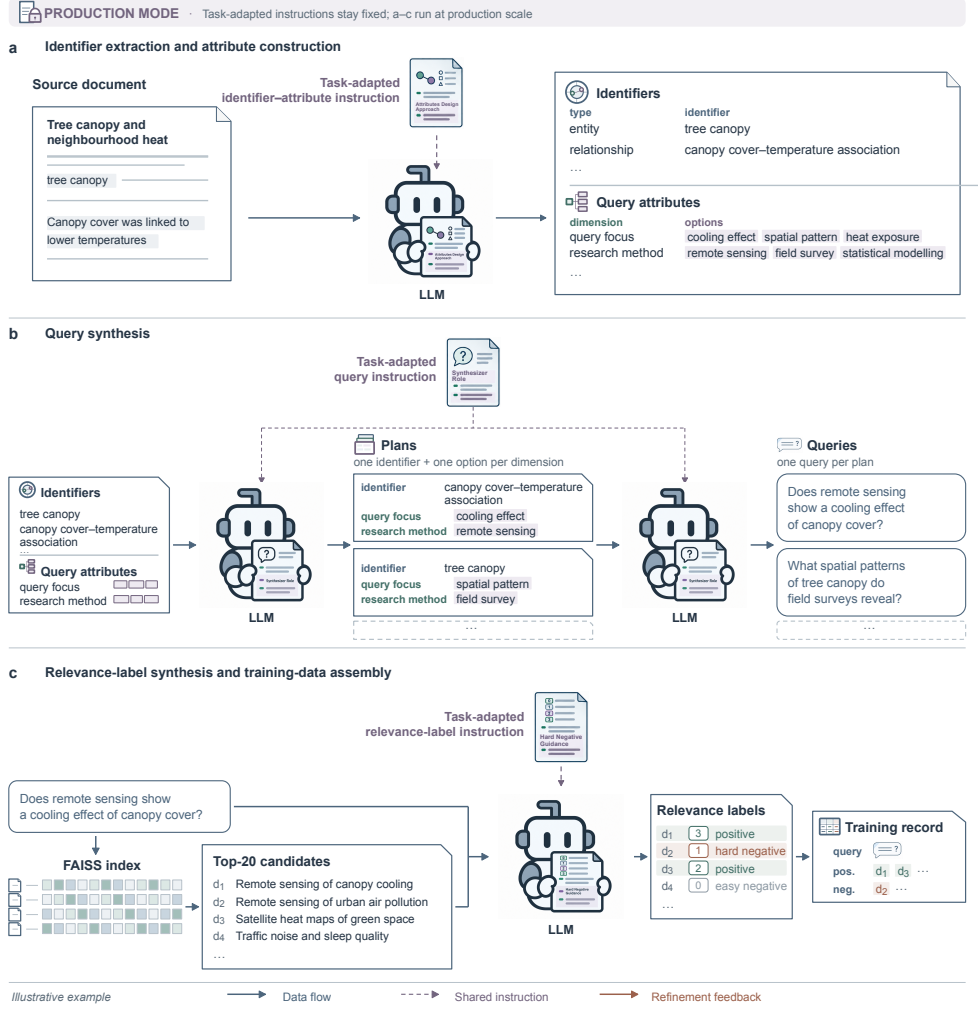


Fig. 3 Production-scale synthesis in SYNTRA. Task-adapted instructions established in test mode remain fixed during production. **a**, For each selected document, an LLM jointly extracts semantic identifiers and constructs query attributes with dimensions and options. **b**, Query synthesis forms plans that pair one semantic identifier with one option from each query-attribute dimension. It then writes one query per plan using the same task-adapted instruction for both steps. **c**, Relevance-label synthesis retrieves candidates from the full corpus and labels them with the task-adapted relevance-label instruction. The labelled candidates are assembled into (query, positive, negative) training records. Document content, plans, queries, candidates and labels are illustrative.

Training on SYNTRA-generated data improved retrieval performance over the MS MARCO-trained baseline in every task-specific comparison shown in Fig. 4a. These gains spanned legal, financial, scientific, biomedical, medical, coding, mathematical, debate and entity retrieval, as well as spatial and commonsense reasoning. These results support using a shared synthesis workflow to construct effective task-specific

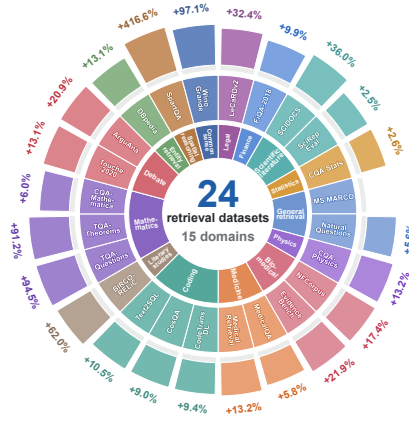
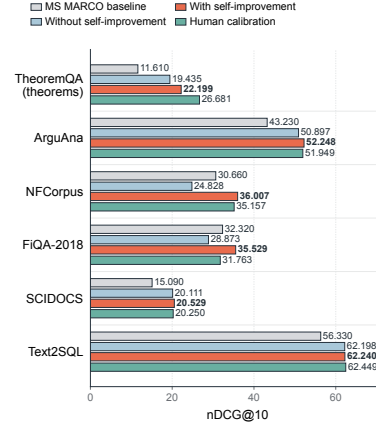
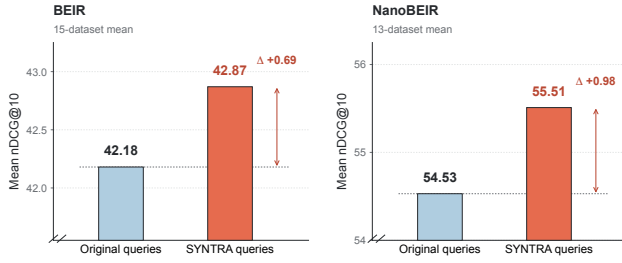
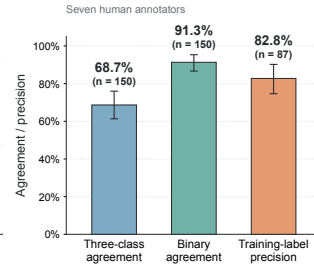
a Cross-task applicability**b Self-improvement across tasks****c Query quality****d Label reliability**

Fig. 4 Cross-task applicability, self-improvement and quality of SYNTRA-generated supervision. **a**, Coverage of 24 datasets spanning 15 domains, with relative retrieval gains for the 23 task-specific comparisons against the MS MARCO-trained baseline. Numeric labels report percentage gains; radial extensions use a nonlinear scale. The CodeTrans-DL gain averages three reported variants. MS MARCO is included as the common training corpus without a task-adaptation gain. **b**, Retrieval performance of the MS MARCO-trained baseline and models trained on data generated without self-improvement, with automatic self-improvement or with human calibration. The six tasks were TheoremQA (theorems), ArguAna, NFCorpus, FiQA-2018 (FiQA), SCIDOCS and Text2SQL. **c**, Mean nDCG@10 on BEIR and NanoBEIR for models trained with original or SYNTRA-generated queries under matched conditions. **d**, Agreement between Qwen3-30B-A3B relevance labels and reference labels derived from votes by seven human annotators. Binary and three-class agreement were evaluated on 150 query–document pairs, with 50 pairs from each of three tasks. Binary evaluation distinguishes relevant from non-relevant documents; three-class evaluation distinguishes positive, hard-negative and easy-negative labels. Training-label precision reports exact-category matches among the 87 evaluation pairs labelled positive or hard negative by the LLM. Error bars indicate 95% confidence intervals from 10,000 bootstrap resamples stratified by dataset. The 95% confidence intervals were 61.3–76.0% for three-class agreement, 86.7–95.3% for binary agreement and 74.7–90.2% for training-label precision. Four pairs with tied highest three-class vote counts were assigned hard-negative reference labels.

supervision across diverse retrieval settings without manually redesigning synthesis instructions for each task.

In the main cross-task evaluation, Qwen3-30B-A3B [34] deployed on four NVIDIA GeForce RTX 5090 GPUs produced the training data. A Qwen3-0.6B backbone was adapted separately for each task using low-rank adaptation (LoRA) [35]. Human involvement was limited to task setup, typically requiring no more than 10 min per task based on retrospective estimates. Selected task-level production times are reported in Extended Data Table 1. Details of human involvement, computational resources and training procedures are provided in Methods (Sections 3.2 and 3.4).

1.2 Self-improvement enhances the training value of synthetic data

Automatic self-improvement increased the training value of SYNTRA-generated data across all six retrieval tasks we tested (Fig. 4b). Comparing synthesis using initial and refined instructions revealed the largest gains on NFCorpus and FiQA. Their nDCG@10 scores increased from 24.828 to 36.007 and from 28.873 to 35.529, respectively. On both tasks, training on the initial synthetic data yielded retrieval scores below the MS MARCO-trained baseline. Training on data generated with the revised instructions exceeded that baseline. Smaller gains were observed on TheoremQA theorems, ArguAna, SCIDOCs and Text2SQL.

Automatic self-improvement provided an effective alternative to human calibration of synthesis instructions. It yielded higher retrieval scores on four of the six tasks. Performance on Text2SQL was close to that obtained with human calibration (62.240 versus 62.449). Human calibration yielded a higher nDCG@10 score on TheoremQA theorems than automatic self-improvement (26.681 versus 22.199).

A FiQA case study illustrates how feedback sharpened the task requirements expressed in shared synthesis instructions (Fig. 5). For query generation, refinement explicitly directed the system to express a user’s financial information need rather than summarize the source document (Fig. 5a). For relevance assessment, a known-positive example exposed an overly narrow interpretation of helpfulness: the annotator rejected an answer explaining why a proposed mortgage-payment plan might fail because it did not help implement the plan. Feedback prompted a revision clarifying that an answer can be relevant by resolving the user’s question, including by correcting its premise (Fig. 5b). The same example was then labelled positive. In both cases, feedback was incorporated into shared instructions for subsequent data synthesis. Complete instructions and example inputs for this case study are provided in Supplementary Note 1.

1.3 Synthetic queries outperform human-generated queries in retrieval training

To assess the effectiveness of synthetic queries for embedding learning, we constructed training data from the same pool of 10,000 MS MARCO source documents, using either original human-generated queries from Bing search logs or SYNTRA-generated

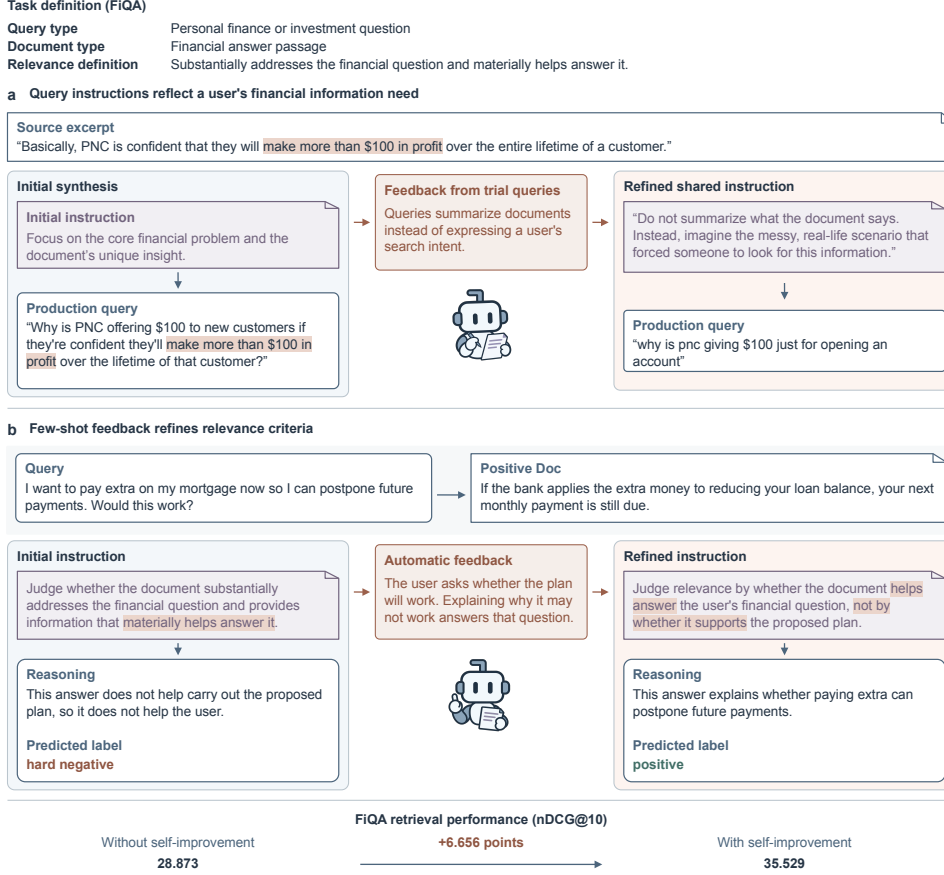


Fig. 5 Few-shot-guided self-improvement on FiQA. **a**, Refinement of query-generation instructions. The source excerpt and queries are verbatim production examples generated in separate runs with instructions corresponding to the first and third refinement rounds. Attribute plans were not held fixed between these runs. Feedback summarizes the original refinement history. **b**, Refinement of relevance-annotation instructions using a known-positive few-shot example. Feedback clarifies that explaining why a plan may fail can help answer the user's question. Text in this panel provides an illustrative summary of the recorded calibration episode. Bottom, FiQA retrieval performance without and with self-improvement, reflecting joint refinement of query-generation and relevance-annotation instructions.

queries. Both query sources underwent the same relevance annotation and negative-mining procedures, and the resulting data were used to train the same embedding backbone with the same training recipe.

Models trained with SYNTRA-generated queries achieved higher mean nDCG@10 on both evaluation suites: 42.87 versus 42.18 on BEIR and 55.51 versus 54.53 on NanoBEIR (Fig. 4c). These results show that synthetic queries can provide more effective supervision for embedding learning than human-generated queries under matched training conditions.

To assess the effectiveness of automatically adapted query synthesis, we compared SYNTRA-generated queries with task-specific query sources for constructing training data. These sources included queries released by ReasonEmbed for TheoremQA questions and theorems [29], and official queries for CosQA [36] and Synthetic Text2SQL. Models trained with SYNTRA-generated queries achieved competitive retrieval performance across all four tasks (Fig. 6d), with higher scores on TheoremQA questions and CosQA and scores close to the references on TheoremQA theorems and Synthetic Text2SQL.

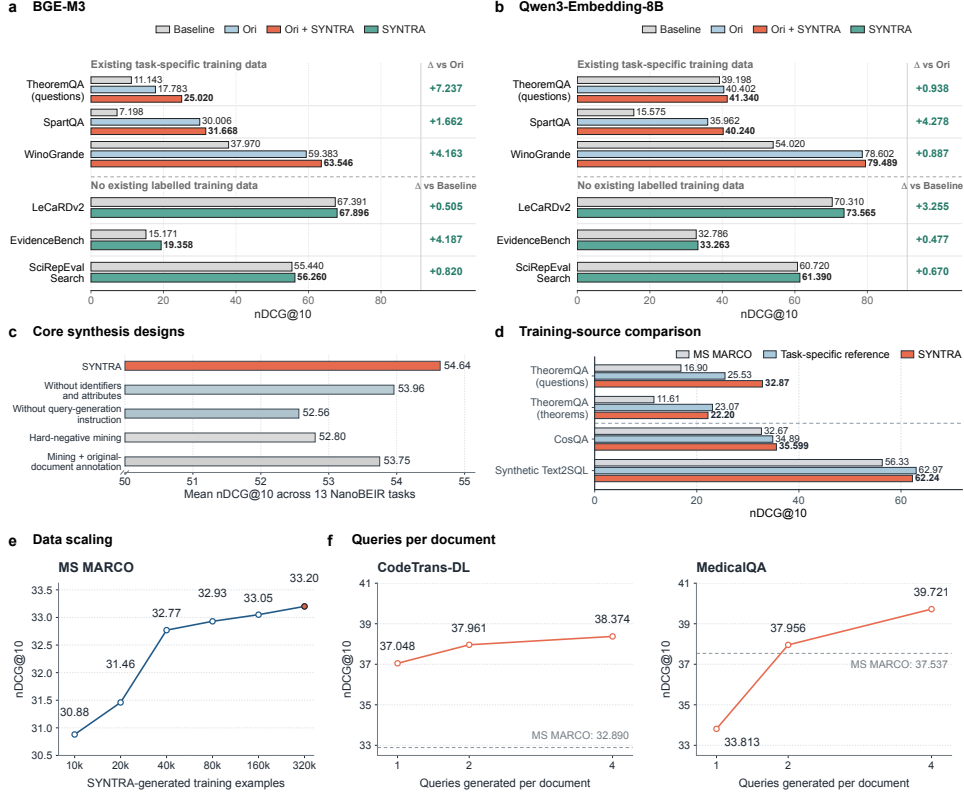


Fig. 6 Model adaptation and ablations of SYNTRA. a,b, BGE-M3 (a) and Qwen3-Embedding-8B (b) results with and without existing task-specific labelled training data. Ori denotes models fine-tuned on existing task-specific labelled data. Dashed lines separate the two settings; right-hand values report gains over Ori or the corresponding pretrained baseline. c, NanoBEIR ablations of the core synthesis components. d, Comparison of the shared MS MARCO baseline, a task-specific reference and SYNTRA. Reference training data were constructed using ReasonEmbed queries for the two TheoremQA tasks and official queries for CosQA and Synthetic Text2SQL. e, MS MARCO nDCG@10 as SYNTRA-generated training data increase from 10,000 to 320,000 examples. f, Retrieval performance as the number of queries generated per document increases on CodeTrans-DL and MedicalQA. Dashed lines show the MS MARCO-trained reference scores.

1.4 Generated labels agree with human relevance judgements

We evaluated Qwen3-30B-A3B relevance labels against reference labels obtained by voting across seven independent human annotators. The evaluation included 150 query–document pairs, with 50 pairs from each of MS MARCO, Text2SQL and TheoremQA (theorems). Dataset-level agreement and class-wise performance are reported in Extended Data Tables 2 and 3.

Binary agreement was 91.3% when distinguishing relevant from non-relevant documents (137/150; Cohen’s $\kappa = 0.719$). Three-class agreement was 68.7% when distinguishing positives, hard negatives and easy negatives (103/150; $\kappa = 0.510$). Most three-class disagreements concerned the distinction between hard and easy negatives (34/47; Extended Data Table 3).

We further assessed the positive and hard-negative labels used for training. Among the 87 evaluation pairs assigned either category by the LLM, 72 matched the human reference category, yielding a pooled label precision of 82.8% (Fig. 4d; Extended Data Table 3).

These results support using SYNTRA-generated relevance labels as supervision for retrieval training. The labels closely matched human binary relevance judgements, while most three-class disagreements concerned the distinction between hard and easy negatives.

1.5 Gains extend to widely used, strong embedding models

The cross-task experiments used a controlled backbone, leaving open whether generated data still help models already trained on large curated corpora. Two situations matter in practice: a new task for which no labelled training set exists, and an established task for which one does. We adapted two widely used, strong embedding models, BGE-M3 [37] and Qwen3-Embedding-8B [38], in both situations, using three retrieval tasks each (Fig. 6a,b).

For tasks without existing labelled training sets, SYNTRA used a small set of few-shot examples to guide task adaptation and generated the data used for model training. Both embedding models improved on all three tasks. On LeCaRDv2, EvidenceBench and SciRepEval Search Tasks, BGE-M3 gained 0.505, 4.187 and 0.820 nDCG@10 points, respectively, while Qwen3-Embedding-8B gained 3.255, 0.477 and 0.670 points. These improvements were measured relative to the corresponding pretrained models without task-specific fine-tuning.

For tasks with existing labelled training sets, we compared fine-tuning on those sets alone with fine-tuning on the same sets augmented by SYNTRA-generated data. Augmentation improved both models on all three tasks. On TheoremQA questions, SpartQA and WinoGrande, BGE-M3 gained 7.237, 1.662 and 4.163 nDCG@10 points, respectively, while Qwen3-Embedding-8B gained 0.938, 4.278 and 0.887 points. These improvements were measured relative to fine-tuning on the existing task-specific training sets alone. Synthetic data can therefore both supply task-specific supervision where labelled training sets are unavailable and complement existing supervision when adapting widely used, strong embedding models.

1.6 Scaling synthetic data yields continued retrieval gains

To evaluate the effect of scaling synthetic training data, we trained a Qwen3-8B backbone on 10,000 to 320,000 SYNTRA-generated examples from MS MARCO passages. We evaluated retrieval performance on MS MARCO with a fixed training recipe and evaluation protocol.

Retrieval performance improved at each of the six evaluated scales. nDCG@10 increased from 30.88 to 33.20 as we scaled up the training data (Fig. 6e), an overall gain of 2.32 points. These results demonstrate the value of scaling synthetic training data, with larger datasets yielding continued improvements in embedding performance.

1.7 Core design choices improve retrieval performance

To identify contributions from SYNTRA’s synthesis components, we evaluated task-adapted query instructions, planning with semantic identifiers and query attributes, relevance annotation, and multiple queries per document. For the component ablations, we report mean nDCG@10 across 13 NanoBEIR tasks. Query-count comparisons are reported separately for each evaluated task (Fig. 6c,f).

Removing semantic identifiers and query attributes reduced mean nDCG@10 on NanoBEIR from 54.64 to 53.96. Removing the task-adapted query-generation instruction reduced the score to 52.56. For relevance-label synthesis, replacing relevance annotation with hard-negative mining reduced the score to 52.80. Adding relevance annotation for the source document raised it to 53.75, below the full workflow’s 54.64. These comparisons support the contributions of semantic identifiers and query attributes, task-adapted query instructions and relevance annotation to the effectiveness of the generated training data.

Generating multiple queries per document also improved retrieval on both evaluated tasks (Fig. 6f). Different combinations of semantic identifiers and query attributes allow the same document to support queries expressing different information needs. Increasing the number of queries from one to two and four raised nDCG@10 from 37.048 to 37.961 and 38.374 on CodeTrans-DL. On MedicalQA, the corresponding scores increased from 33.813 to 37.956 and 39.721. These results support generating several queries per document as a way to provide additional useful supervision on these tasks.

2 Discussion

Adapting retrieval models to new domains and tasks requires training data that reflect their information needs and relevance criteria. SYNTRA uses few-shot-guided self-improvement to adapt shared synthesis instructions to different retrieval requirements. Few-shot examples convey what users seek and which documents satisfy their needs. Feedback on trial outputs translates that understanding into reusable guidance for constructing supervision. The synthesis system can therefore take on part of the task-specific design work needed to adapt specialised embedding models. Once refined, the instructions are reused for large-scale production with a locally deployed

model, while commercial-model calls are concentrated in small-scale calibration. This supports economical data synthesis.

The self-improvement results show why adapting synthesis instructions matters for downstream learning. On NFCorpus and FiQA, training on data generated with refined instructions exceeded the MS MARCO-trained baseline. Training on data generated with the initial instructions fell below that baseline. Automatic self-improvement was competitive with human calibration on most tested tasks, although human calibration remained advantageous on TheoremQA (theorems). SYNTRA-generated data also benefited widely used, strong embedding models, supplying supervision where labelled training sets were unavailable and complementing existing supervision. On MS MARCO, retrieval gains from scaling up the training data indicate that the additional synthetic data remained useful across the evaluated scales. These findings support few-shot-guided adaptation of a shared synthesis workflow within the retrieval settings examined here.

A natural extension is to apply self-improvement to training-data synthesis for diverse LLM capabilities. Such systems could infer what a target capability requires and adapt how training examples are generated and assessed, with minimal human involvement and low production costs. Recent studies have automated question-answer construction with assessment criteria and task construction with teacher demonstrations, providing foundations for this direction [39, 40]. Feedback for self-improvement would need to show whether generated examples meet the intended training requirements. Answer checks and code execution offer scalable ways to assess correctness in mathematical and coding tasks [41]. Within a task, these checks may need to be combined with assessments of relevance, completeness and contextual requirements using task-specific rubrics [42]. Such feedback could guide revisions to the synthesis rules for the target capability.

A further direction is to use feedback from model evaluation, task execution and user interactions to identify previously unmet training needs. These signals could also reveal requirements that emerge during use and guide revisions to generated examples and their assessment criteria. For retrieval and LLM training alike, this could extend self-improvement towards identifying what models need to learn next and adapting data synthesis accordingly.

3 Methods

We describe corpus preprocessing and task preparation, followed by self-improvement in test mode and data synthesis in production mode. We then explain how the generated labels are converted into training data and describe the embedding models and evaluation procedures.

3.1 Corpus preprocessing and task preparation

SYNTRA receives an unlabelled corpus and a small set of few-shot examples, each pairing a query with a positive document (Fig. 2a). Corpus preprocessing supplies the documents and retrieval index used in test and production modes. Document titles and bodies are concatenated where both are available, encoded as L2-normalised

Algorithm 1 Corpus indexing and diversity-aware document selection

Require: Corpus $\mathcal{C}$, encoder f , target count M , similarity threshold τ

Ensure: Selected documents S and full-corpus retrieval index J

```
1: Encode each document  $d$  as  $\mathbf{v}_d = f(d)/\|f(d)\|_2$ 
2: Build inner-product index  $J$  over all document vectors
3: Build an auxiliary IVF index  $J_{\text{sel}}$  for neighbour search
4:  $L \leftarrow \text{SHUFFLE}(\mathcal{C})$ ;  $A \leftarrow \mathcal{C}$ ;  $S \leftarrow \emptyset$ 
5: for each document  $d$  in  $L$  do
6:   if  $|S| = M$  or  $A = \emptyset$  then
7:     break
8:   end if
9:   if  $d \in A$  then
10:     $S \leftarrow S \cup \{d\}$ ;  $A \leftarrow A \setminus \{d\}$ 
11:     $N_d \leftarrow \text{AVAILABLENEIGHBOURS}(J_{\text{sel}}, \mathbf{v}_d, A)$ 
12:     $A \leftarrow A \setminus \{d' \in N_d : \mathbf{v}_d^\top \mathbf{v}_{d'} > \tau\}$ 
13:   end if
14: end for
15: return  $S, J$ 
```

vectors and stored in a FAISS inner-product index [43]. Diversity-aware document selection uses an auxiliary inverted-file (IVF) index to reduce semantic redundancy among the documents used for generation (Algorithm 1). It selects documents in shuffled order and excludes neighbours whose similarity exceeds the selection threshold. Selection continues until the target count is reached or the available pool is exhausted. Where feasible, we targeted a selection of 10,000 documents for data generation. Test mode uses small samples of the selected documents, whereas production processes the selected set with the final instructions. Selection leaves the full-corpus index intact for candidate retrieval in both modes.

Task preparation uses the few-shot examples to infer a task definition comprising the query type, document type and relevance criteria. A generate-evaluate-refine procedure compares the proposed definition with the examples and revises inaccurate or insufficiently specific requirements. Alternatively, users can supply a task definition directly for the synthesis workflow. This step establishes the task requirements used to construct and refine the synthesis instructions in test mode (Section 3.2).

3.2 Task adaptation through self-improvement

Test mode implements self-improvement by refining shared synthesis instructions through generate-evaluate-refine cycles on small samples (Fig. 2b). Each synthesis module generates trial outputs and evaluates them against the task definition and few-shot examples. Diagnostic feedback guides revisions to its instruction or guidance before the module runs again on the same trial inputs. These revisions update the shared synthesis rules that guide subsequent data generation across documents.

Each loop terminates when verification passes or its iteration limit is reached. All three loops allow up to three attempts by default, including the initial trial.

This limit is configurable. For the identifier–attribute and query loops, LLM verifiers return scores from 0 to 3. Scores of at least 2 are treated as a pass in these loops. Relevance-label verification uses the programmatic consistency check on few-shot positive pairs described below. On termination, SYNTRA stores the last evaluated instruction, including when the iteration limit is reached without a pass. All loops finish before production begins, and their final shared instructions remain fixed throughout production.

Identifier extraction and attribute construction share one refinement loop on sampled documents. The initial trial uses a shared generation template conditioned on the task definition, with no additional refinement guidance. An LLM verifier assesses document grounding, redundancy and whether the attribute dimensions and options support useful variation within the task. Its feedback is used to develop reusable guidance for subsequent extraction and construction. After this loop finishes, SYNTRA regenerates identifiers and attributes with the final guidance to provide fixed inputs for query-instruction refinement.

Query-instruction refinement adapts information-need planning and query writing to the task definition and few-shot examples. The initial instruction is induced from these inputs and specifies the intended query producer and information need. It also specifies how document evidence should be selected or abstracted, together with the linguistic characteristics and constraints of the target queries. Trial synthesis applies this instruction to both planning and query writing. An LLM verifier evaluates sampled document–query pairs for style, task alignment and naturalness. It also assesses whether queries plausibly reflect their intended producer and whether the instruction is effective. Its feedback guides revisions to the instruction used in both stages, while the upstream identifiers and attributes remain fixed during this loop.

Relevance-label refinement adapts the task-specific criteria for positives, hard negatives and easy negatives. The initial instruction is generated from the task definition and the required annotation format. It is tested on the known (query, positive document) pairs supplied as few-shot examples. A program checks whether every pair is classified as positive, treating missing predictions as mismatches. When the check fails, the LLM analyses the errors and revises the relevance criteria. It also infers and refines criteria distinguishing apparently relevant documents that fail a required condition from documents clearly unrelated to the information need. These criteria become part of the shared annotation instruction used during production. The examples directly verify positive classifications, whereas negative criteria are inferred from the task requirements. The framework also supports calibration against human-confirmed labels during test mode.

Human involvement in the principal experiments was confined to task setup. The operator reviewed and occasionally revised the inferred task definition before data production. The operator also sometimes adjusted the document-selection threshold to obtain the intended document count. Retrospective estimates indicated that setup typically required no more than 10 min per task.

The self-improvement ablation compared three conditions across six retrieval tasks (Fig. 4b). Synthesis without self-improvement used the initial task-specific instructions for data production. Automatic self-improvement used instructions revised through

feedback during test mode. In the human-calibration condition, human input was used to calibrate module-level synthesis instructions during test mode. Data generated under each condition were used to train task-specific embedding models. Their retrieval performance is compared in Results (Section 1.2).

3.3 Production-mode data synthesis

In production mode, three synthesis modules reuse the final task-adapted instructions across the selected documents (Fig. 3). Their outputs comprise generated queries and sets of labelled candidate documents. These outputs are converted into training records as described in Section 3.4.

3.3.1 Identifier extraction and attribute construction

For each selected document, a single LLM call jointly extracts semantic identifiers and constructs query attributes. This call uses the instruction and guidance established during test mode. Semantic identifiers are document-grounded entities or relations that provide possible targets for an information need. Query attributes define dimensions along which the information need or its expression can vary, such as scope, reasoning requirements or specificity. Both the dimensions and their task-appropriate options are constructed for each document. The resulting identifiers and attributes provide the components for planning distinct queries.

3.3.2 Query synthesis

Query synthesis uses two LLM stages to plan an information need and express it as a query. The planner receives the source document, task definition, few-shot examples and task-adapted query instruction. Each proposed plan combines one semantic identifier with one option from every query-attribute dimension. These plans specify the information need and its intended expression for query writing. The query-writing stage converts each plan into a query using the same task-adapted instruction and source document.

3.3.3 Relevance-label synthesis

Each generated query retrieves its top- K candidate documents from the full corpus using the indexing encoder and cosine similarity. If the source document is retrieved, it receives the same relevance assessment as the other candidates. Candidates are annotated in chunks using the task-adapted relevance-label instruction, with each query-document pair judged independently on a four-point scale. A score of 3 indicates strong satisfaction of the relevance criteria, whereas 2 indicates a weaker but valid positive. A score of 1 indicates a plausible candidate that fails the criteria, and 0 indicates a clearly unrelated candidate. Scores of 3 and 2 map to positive, 1 to hard negative and 0 to easy negative. Candidate annotations are merged into positive, hard-negative and easy-negative sets for training-data construction. Parsing retries and structural checks are also applied to outputs during production.

3.4 Training data construction and model configurations

Training records.

Training records retain queries with at least one positive and one usable negative document. Each record contains the query and variable-length lists of positive and negative document texts. The hard-negative-only export uses labelled hard negatives as its negative examples. An alternative export supplements hard negatives with labelled easy negatives when fewer than the requested number of hard negatives are available. For Qwen3 training, the loader samples one positive and a configurable number of negatives from each record. It repeats available negative indices when necessary to supply the requested number. Records also contain a retrieval instruction that can be prepended to the query during training.

Embedding models.

Embedding-model training and retrieval evaluation followed completion of data synthesis. The main cross-task evaluation used a Qwen3-0.6B backbone with LoRA adaptation, with MS MARCO-trained models providing the comparison baselines. SYNTRA generated data separately for each task, and models trained on those data were evaluated on the corresponding retrieval tasks. Key experimental settings are provided in Supplementary Note 2 (Supplementary Tables 1–2).

Experiments with BGE-M3 and Qwen3-Embedding-8B evaluated synthetic data in two settings (Fig. 6a,b). For tasks without existing labelled training sets (LeCaRDv2, EvidenceBench and SciRepEval Search [44]), models fine-tuned on SYNTRA-generated data were compared with their pretrained counterparts. For tasks with existing labelled training sets (TheoremQA questions [45], SpartQA [46] and WinoGrande [47]), we compared two fine-tuning conditions. These used the existing sets alone or the same sets augmented by SYNTRA-generated data.

Query-source comparisons.

The MS MARCO query-source experiment compared original human-generated queries from Bing search logs [30] with SYNTRA-generated queries using the same pool of 10,000 source documents. Both conditions used the same relevance-annotation and negative-mining procedures and the same embedding-model training recipe. Valid training-record counts are provided in Extended Data Table 1. Retrieval performance was evaluated on both the BEIR and NanoBEIR suites.

We also compared SYNTRA-generated queries with task-specific query sources on TheoremQA questions and theorems, CosQA and Synthetic Text2SQL. Within each task, both query sources underwent the same candidate-retrieval and relevance-annotation procedures and were used to train the same embedding backbone.

For the TheoremQA comparisons, we used the ReasonEmbed-Data-0928 release [29] from the Hugging Face repository `hanhainebola/reason-embed-data` (configuration `v0928`, revision `8dada2e649a2`). The files `theoremqa_questions-formatted.jsonl` and `theoremqa_theorems-formatted.jsonl` contained 7,184 and 8,861 queries, respectively. We used the `query` field and paired each query with one of its

released positive documents as the source document for candidate retrieval and relevance annotation. All usable released queries were retained, without matching their counts to SYNTRA.

For CosQA and Synthetic Text2SQL, reference queries came from the `train` splits of `CoIR-Retrieval/cosqa` (revision 0846fa3b963a) and `CoIR-Retrieval/synthetic-text2sql` (revision 16c5efcfbda2), respectively. CosQA used all 9,020 positively labelled training pairs. For Synthetic Text2SQL, we sampled 10,000 official training pairs from the 13,621 whose document identifiers intersected the selected source-document pool, using seed 20260627. SYNTRA generated queries from those same 10,000 documents.

The SYNTRA-trained CosQA model used seed 42 and a maximum passage length of 1,024 tokens. The official-query reference used seed 42 and 512 tokens. Other training settings followed the same recipe.

For these four tasks, training records required at least one annotated positive and one hard negative. Two positive-selection variants were supplied together to the training loader: one retained all annotated positives, whereas the other retained only the source document when it was among those positives and otherwise kept the original positive list. The variants used the same negatives and were concatenated without deduplication. Query counts, valid-record counts and the four tasks’ absolute retrieval scores are reported in Extended Data Table 4.

Data-scaling experiment.

The MS MARCO data-scaling experiment used a Qwen3-8B backbone with 10,000, 20,000, 40,000, 80,000, 160,000 and 320,000 SYNTRA-generated training examples. The training recipe and evaluation protocol were held fixed across data scales. We performed one training run at each data scale using the settings in Supplementary Table 2.

Computational resources and production times.

Data production used Qwen3-30B-A3B on four NVIDIA GeForce RTX 5090 GPUs within a six-GPU system. The other two GPUs in the system were used for embedding-model operations. Extended Data Table 1 reports selected production batches across 24 retrieval tasks used in the experiments. The reported times cover production stages and exclude calibration, prebuilt indexing and embedding-model training and evaluation. The table caption specifies the batch-selection procedure and the stages included in the timings.

3.5 Human evaluation of relevance labels

Seven annotators independently rated the 150 query–document pairs using the relevance scale and label mapping described above (Section 3.3.3). Human reference labels were determined by plurality voting over the three categories. Four pairs with tied highest vote counts were assigned hard negative. For binary evaluation, each annotator’s two negative categories were merged before majority voting. We measured agreement between LLM and human reference labels using exact matches and Cohen’s κ . Human inter-annotator agreement was measured using Fleiss’ κ on the mapped

labels. Training-label precision was the fraction of LLM-labelled positives and hard negatives whose exact category matched the human reference. We estimated 95% percentile confidence intervals using 10,000 bootstrap resamples, each comprising 50 query–document pairs sampled with replacement from each dataset. Human reference labels were held fixed, and training-label precision was recalculated for each resample.

Human inter-annotator agreement, measured by Fleiss’ κ , was 0.483 for three-class labels and 0.606 for binary labels. To assess whether the four tied pairs influenced LLM–human agreement, we repeated the three-class comparison after excluding those pairs. Agreement was 69.2% (101/146; Cohen’s $\kappa = 0.520$), compared with 68.7% (103/150) when all pairs were included.

3.6 Consistency between NanoBEIR and BEIR

Evaluating every model variant on the complete BEIR suite [31] requires substantial computation and time. We therefore examined whether NanoBEIR could serve as a lower-cost proxy for repeated model screening and ablation studies during iterative development. We evaluated 18 publicly available embedding models on both BEIR and NanoBEIR and computed paired suite-level mean nDCG@10 scores (Extended Data Table 5). These comprised nine encoder-only models and nine decoder-only models. Only the models listed in the paired-score table were included in the correlation analysis.

NanoBEIR evaluation used the MTEB implementation [48] and comprised 13 retrieval tasks: ArguAna, Climate-FEVER, DBpedia, FEVER, FiQA2018, HotpotQA, MS MARCO, NFCorpus, Natural Questions, Quora, SCIDOCS, SciFact and Touche2020. We evaluated the split named `train` in the corresponding `zeta-alpha-ai/Nano*` dataset releases. Each task contained 50 evaluation queries except Touche2020, which contained 49, giving 649 queries in total. For each task, nDCG@10 was averaged over its evaluation queries. The suite score was the unweighted arithmetic mean of the 13 task scores, multiplied by 100; tasks therefore contributed equally rather than in proportion to their query counts.

We evaluated component ablations on NanoBEIR and the MS MARCO query-source comparison on both BEIR and NanoBEIR. NanoBEIR runs for these experiments used MTEB version 1.39.7. The 18-model consistency analysis used versions 1.39.7 and 2.12.30. The dataset identifiers and pinned dataset revisions were identical across these evaluations and across the two software versions.

The decoder-only models generally occupied a higher score range than the encoder-only models. Pooling the two families could therefore yield a high correlation partly driven by between-family differences. We computed Spearman’s rank correlation separately for each family to assess whether NanoBEIR preserved the ordering of models within each group. NanoBEIR closely reproduced the BEIR ordering for encoder-only models ($\rho = 0.933$, two-sided $P = 2.0 \times 10^{-4}$). Decoder-only models also showed a strong positive association ($\rho = 0.817$, two-sided $P = 7.0 \times 10^{-3}$; Extended Data Table 6). The median absolute rank change was one position in both groups. Eight of nine encoder-only models and six of nine decoder-only models remained within one position of their BEIR rank.

These results indicate that NanoBEIR preserves the broad relative ordering of models within both architecture families. They support using NanoBEIR as a lower-cost proxy for rapid model screening, ablation studies and hyperparameter selection during iterative experimentation. The absolute nDCG@10 scores on the two suites are not directly interchangeable, and several local rank reversals remain.

Data availability. SYNTRA retrieval-training data exports, including synthetic-query data, original-query controls and archived experimental variants, are publicly available at <https://huggingface.co/datasets/ZucanLyu/SYNTRA>. The repository provides file manifests and describes the audited exports. Source benchmark data are available from their original providers, as cited in this Article, and remain subject to their respective access conditions and terms.

Code availability. The SYNTRA source code is publicly available at <https://github.com/zucanly/SYNTRA>. The repository includes the data synthesis pipeline, training and retrieval evaluation scripts, code for human relevance evaluation, configuration files, and installation and workflow documentation.

References

- [1] Asai, A. *et al.* Synthesizing scientific literature with retrieval-augmented language models. *Nature* **650**, 857–863 (2026). URL <https://www.nature.com/articles/s41586-025-10072-4>.
- [2] Roberts, K. *et al.* TREC-COVID: Rationale and structure of an information retrieval shared task for COVID-19. *Journal of the American Medical Informatics Association* **27**, 1431–1436 (2020). URL <https://doi.org/10.1093/jamia/ocaa091>.
- [3] Song, T., Gan, G., Shang, M. & Zhao, Y. *IFIR: A comprehensive benchmark for evaluating instruction-following in expert-domain information retrieval. Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, 10186–10204 (2025). URL <https://aclanthology.org/2025.naacl-long.511/>.
- [4] Wang, K., Thakur, N., Reimers, N. & Gurevych, I. *GPL: Generative pseudo labeling for unsupervised domain adaptation of dense retrieval. Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2345–2360 (Association for Computational Linguistics, 2022). URL <https://aclanthology.org/2022.naacl-main.168/>.
- [5] Li, M. & Gaussier, E. *Domain adaptation for dense retrieval and conversational dense retrieval through self-supervision by meticulous pseudo-relevance labeling. Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, 5247–5259 (ELRA and ICCL, 2024). URL <https://aclanthology.org/2024.lrec-main.467/>.

- [6] Karpukhin, V. *et al.* *Dense passage retrieval for open-domain question answering. Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, 6769–6781 (Association for Computational Linguistics, 2020). URL <https://aclanthology.org/2020.emnlp-main.550/>.
- [7] Zhou, K. *et al.* *SimANS: Simple ambiguous negatives sampling for dense text retrieval. Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*, 548–559 (Association for Computational Linguistics, 2022). URL <https://aclanthology.org/2022.emnlp-industry.56/>.
- [8] Ma, J., Korotkov, I., Yang, Y., Hall, K. & McDonald, R. *Zero-shot neural passage retrieval via domain-targeted synthetic question generation. Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, 1075–1088 (Association for Computational Linguistics, 2021). URL <https://aclanthology.org/2021.eacl-main.92/>.
- [9] Jeronymo, V. *et al.* InPars-v2: Large language models as efficient dataset generators for information retrieval. *arXiv preprint arXiv:2301.01820* (2023). URL <https://arxiv.org/abs/2301.01820>.
- [10] Bonifacio, L., Abonizio, H., Fadaee, M. & Nogueira, R. *InPars: Unsupervised dataset generation for information retrieval. Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2387–2392 (Association for Computing Machinery, 2022). URL <https://doi.org/10.1145/3477495.3531863>.
- [11] Dai, Z. *et al.* *Promptagator: Few-shot dense retrieval from 8 examples. International Conference on Learning Representations* (2023). URL <https://openreview.net/forum?id=gmL46Ympu2J>.
- [12] Xian, J. *et al.* Prompts as auto-optimized training hyperparameters: Training best-in-class IR models from scratch with 10 gold labels. *arXiv preprint arXiv:2406.11706* (2024). URL <https://arxiv.org/abs/2406.11706>.
- [13] Saad-Falcon, J. *et al.* *UDAPDR: Unsupervised domain adaptation via LLM prompting and distillation of rerankers. Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 11265–11279 (Association for Computational Linguistics, 2023). URL <https://aclanthology.org/2023.emnlp-main.693/>.
- [14] Mayfield, J. *et al.* Synthetic cross-language information retrieval training data. *arXiv preprint arXiv:2305.00331* (2023). URL <https://arxiv.org/abs/2305.00331>.
- [15] Abonizio, H. *et al.* InPars Toolkit: A unified and reproducible synthetic data generation pipeline for neural information retrieval. *arXiv preprint arXiv:2307.04601* (2023). URL <https://arxiv.org/abs/2307.04601>.

- [16] Fang, Y. *et al.* *Scaling laws for dense retrieval. Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1339–1349 (Association for Computing Machinery, 2024). URL <https://doi.org/10.1145/3626772.3657743>.
- [17] Wen, H., Guo, J., Zhang, Y., Jiang, J. & Wang, Z. *On synthetic data strategies for domain-specific generative retrieval. Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 7961–7976 (Association for Computational Linguistics, 2025). URL <https://aclanthology.org/2025.acl-long.392/>.
- [18] Chaudhary, A., Raman, K. & Bendersky, M. *It’s all relative! – a synthetic query generation approach for improving zero-shot relevance prediction. Findings of the Association for Computational Linguistics: NAACL 2024*, 1645–1664 (Association for Computational Linguistics, 2024). URL <https://aclanthology.org/2024.findings-naacl.107/>.
- [19] Lee, J. *et al.* *Gecko: Versatile text embeddings distilled from large language models. arXiv preprint arXiv:2403.20327* (2024). URL <https://arxiv.org/abs/2403.20327>.
- [20] Thomas, P., Spielman, S., Craswell, N. & Mitra, B. *Large language models can accurately predict searcher preferences. Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Association for Computing Machinery, 2024). URL <https://www.microsoft.com/en-us/research/publication/large-language-models-can-accurately-predict-searcher-preferences/>.
- [21] Sun, W. *et al.* *Is ChatGPT good at search? investigating large language models as re-ranking agents. Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 14918–14937 (Association for Computational Linguistics, 2023). URL <https://aclanthology.org/2023.emnlp-main.923/>.
- [22] Wang, L. *et al.* *Improving text embeddings with large language models. Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 11897–11916 (Association for Computational Linguistics, 2024). URL <https://aclanthology.org/2024.acl-long.642/>.
- [23] Almeida, T. & Matos, S. *Exploring efficient zero-shot synthetic dataset generation for information retrieval. Findings of the Association for Computational Linguistics: EACL 2024*, 1214–1231 (Association for Computational Linguistics, 2024). URL <https://aclanthology.org/2024.findings-eacl.81/>.

- [24] Chen, H. *et al.* *Little giants: Synthesizing high-quality embedding data at scale. Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, 1392–1411 (Association for Computational Linguistics, 2025). URL <https://aclanthology.org/2025.naacl-long.64/>.
- [25] Krastev, M., Hamar, M., Toapanta, D., Brouwers, J. & Lei, Y. InPars+: Supercharging synthetic data generation for information retrieval systems. *arXiv preprint arXiv:2508.13930* (2025). URL <https://arxiv.org/abs/2508.13930>.
- [26] Springer, J. M., Adlakha, V., Reddy, S., Raghunathan, A. & Mosbach, M. *Understanding the influence of synthetic data for text embedders. Findings of the Association for Computational Linguistics: ACL 2025*, 22551–22567 (Association for Computational Linguistics, 2025). URL <https://aclanthology.org/2025.findings-acl.1160/>.
- [27] Li, C., Chen, J., Shao, Y., Lian, D. & Liu, Z. *Towards a generalist code embedding model based on massive data synthesis. Advances in Neural Information Processing Systems*, Vol. 38 (2025). URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/da5b2141ae6563ddfb813d928377c2d9-Abstract-Dataset_s_and_Benchmarks_Track.html.
- [28] Shao, R. *et al.* *ReasonIR: Training retrievers for reasoning tasks. Second Conference on Language Modeling* (2025). URL <https://openreview.net/forum?id=kkBCNLMbGj>.
- [29] Chen, J., Lan, J., Li, C., Lian, D. & Liu, Z. *ReasonEmbed: Enhanced text embeddings for reasoning-intensive document retrieval. Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 1203–1221 (Association for Computational Linguistics, 2026). URL <https://aclanthology.org/2026.acl-long.54/>.
- [30] Nguyen, T. *et al.* *MS MARCO: A human generated machine reading comprehension dataset. Proceedings of the Workshop on Cognitive Computation: Integrating neural and symbolic approaches 2016*, Vol. 1773 of *CEUR Workshop Proceedings* (2016). URL https://ceur-ws.org/Vol-1773/CoCoNIPS_2016_paper9.pdf.
- [31] Thakur, N., Reimers, N., Rücklé, A., Srivastava, A. & Gurevych, I. *BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, Vol. 1 (2021). URL <https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/65b9eea6e1cc6bb9f0cd2a47751a186f-Abstract-round2.html>.
- [32] Izacard, G. *et al.* *Unsupervised dense information retrieval with contrastive learning. Transactions on Machine Learning Research* (2022). URL <https://openreview.net/forum?id=jKN1pXi7b0>.

- [33] Wang, L. *et al.* Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533* (2022). URL <https://arxiv.org/abs/2212.03533>.
- [34] Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388* (2025). URL <https://arxiv.org/abs/2505.09388>.
- [35] Hu, E. J. *et al.* LoRA: Low-Rank Adaptation of Large Language Models. *International Conference on Learning Representations* (2022). URL <https://openreview.net/forum?id=nZeVKeeFYf9>.
- [36] Huang, J. *et al.* CoSQA: 20,000+ Web Queries for Code Search and Question Answering. *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 5690–5700 (Association for Computational Linguistics, 2021). URL <https://aclanthology.org/2021.acl-long.442/>.
- [37] Chen, J. *et al.* M3-Embedding: Multi-Linguality, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation. *Findings of the Association for Computational Linguistics: ACL 2024*, 2318–2335 (Association for Computational Linguistics, 2024). URL <https://aclanthology.org/2024.findings-acl.137/>.
- [38] Zhang, Y. *et al.* Qwen3 Embedding: Advancing Text Embedding and Reranking Through Foundation Models. *arXiv preprint arXiv:2506.05176* (2025). URL <https://arxiv.org/abs/2506.05176>.
- [39] Li, X. *et al.* ARES: Automated rubric synthesis for scalable LLM reinforcement learning (2026). URL <https://arxiv.org/abs/2605.23454>. [arXiv:2605.23454](https://arxiv.org/abs/2605.23454).
- [40] Zhao, J. *et al.* NexForge: Scaling agent capabilities through requirement-driven task synthesis for LLMs (2026). URL <https://arxiv.org/abs/2607.14186>. [arXiv:2607.14186](https://arxiv.org/abs/2607.14186).
- [41] Guo, D., Yang, D., Zhang, H. *et al.* DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature* **645**, 633–638 (2025). URL <https://www.nature.com/articles/s41586-025-09422-z>.
- [42] Gunjal, A. *et al.* Rubrics as rewards: Reinforcement learning beyond verifiable domains. (eds Vondrick, C. *et al.*) *International Conference on Learning Representations*, Vol. 2026, 127924–127945 (2026). URL https://proceedings.iclr.cc/paper_files/paper/2026/file/cfd7bee7a651ee9af525098ef67a9e45-Paper-Conference.pdf.
- [43] Douze, M. *et al.* The Faiss library. *arXiv preprint arXiv:2401.08281* (2024). URL <https://arxiv.org/abs/2401.08281>.

- [44] Singh, A., D’Arcy, M., Cohan, A., Downey, D. & Feldman, S. *SciRepEval: A Multi-Format Benchmark for Scientific Document Representations*. *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 5548–5566 (Association for Computational Linguistics, 2023). URL <https://aclanthology.org/2023.emnlp-main.338/>.
- [45] Su, H. *et al.* *BRIGHT: A Realistic and Challenging Benchmark for Reasoning-Intensive Retrieval*. *International Conference on Learning Representations* (2025). URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/7a0f8055c838df8e62329a76c7c6403d-Abstract-Conference.html.
- [46] Mirzaee, R., Rajaby Faghihi, H., Ning, Q. & Kordjamshidi, P. *SPARTQA: A Textual Question Answering Benchmark for Spatial Reasoning*. *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 4582–4598 (Association for Computational Linguistics, 2021). URL <https://aclanthology.org/2021.naacl-main.364/>.
- [47] Sakaguchi, K., Le Bras, R., Bhagavatula, C. & Choi, Y. *WinoGrande: An Adversarial Winograd Schema Challenge at Scale*. *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34, 8732–8740 (2020). URL <https://ojs.aaai.org/index.php/AAAI/article/view/6399>.
- [48] Muennighoff, N., Tazi, N., Magne, L. & Reimers, N. *MTEB: Massive text embedding benchmark*. *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, 2014–2037 (Association for Computational Linguistics, 2023). URL <https://aclanthology.org/2023.eacl-main.148/>.
- [49] Li, H. *et al.* *LeCaRDv2: A Large-Scale Chinese Legal Case Retrieval Dataset*. *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2251–2260 (Association for Computing Machinery, 2024). URL <https://doi.org/10.1145/3626772.3657887>.

Acknowledgements. We thank the annotators for their contributions to the human relevance annotation study.

Author contributions. Zucan Lyu, J.C., Zheng Liu and D.L. conceived the study and designed the methodology. Zucan Lyu developed the software, prepared the corpora and retrieval tasks, and produced the synthetic data. Zucan Lyu and J.C. designed the experiments, and Zucan Lyu and R.T. performed them. Zucan Lyu, R.T., J.C. and P.J. conducted the human calibration and annotation studies. Zucan Lyu and R.T. analysed the results and prepared the figures and tables. Zucan Lyu, R.T., J.C. and D.L. wrote and revised the manuscript. J.C., Zheng Liu and D.L. supervised the research and provided resources.

Competing interests. The authors declare no competing interests.

Supplementary information. Supplementary Information contains complete principal-workflow prompts, retrieval-task definitions and complete initial and refined

instructions for the FiQA case study (Supplementary Note 1), and key experimental settings (Supplementary Note 2; Supplementary Tables 1–2).

Extended Data

Extended Data Table 1 Production-stage runtimes and finalised training-data sizes across 24 retrieval tasks.

Production timings are reported for one batch per listed task and query-source condition; when multiple batches were considered, the shorter-running batch was retained. Training records are valid entries in a finalised JSONL (JSON Lines) training-data file. Each batch has two equally sized training-file variants. Counts are reported per variant, without summing the variants or multiplying by training epochs. Timings include identifier extraction and attribute construction, query generation, candidate retrieval and final relevance annotation. Calibration, prebuilt indexing, failed attempts, waiting between resumed runs, final training-format conversion, and model training and evaluation are excluded. For resumed runs, elapsed times for the relevant completed stages are summed. For MS MARCO original queries, the reported runtime covers retrieval and annotation only. The measurements describe selected batches rather than task-average runtimes or total experimental costs. Data production used a system equipped with six NVIDIA GeForce RTX 5090 GPUs. Four hosted the generative model, and the other two were used only during embedding-model operations. Times are elapsed production-stage times, rather than GPU-hours or sums of per-GPU runtimes.

Task	Training records	Production time (h)
MS MARCO (original queries)	9,546	4.444
MS MARCO (SYNTRA queries)	9,486	5.789
LeCaRDv2 [49]	8,278	63.786
FiQA	9,360	6.489
SCIDOCS	9,742	7.197
SciRepEval Search	9,302	7.040
CQADupStack-Stats	9,620	10.625
Natural Questions	3,886	2.429
CQADupStack-Physics	9,165	4.157
NFCorpus	2,897	2.250
EvidenceBench	9,817	7.223
MedicalQA (1 query/document)	1,919	2.498
MedicalRetrieval	9,754	3.561
CodeTrans-DL (1 query/document)	615	4.214
CosQA	5,140	3.761
Text2SQL	8,419	5.300
BIRCO-relic	4,909	4.614
TheoremQA (questions)	9,006	8.529
TheoremQA (theorems)	9,517	8.950
CQADupStack-Mathematica	9,337	8.245
Touche2020	9,511	8.214
ArguAna	8,528	9.702
DBpedia-entity	9,537	6.662
SpartQA	1,062	5.381
WinoGrande	2,672	4.117

Extended Data Table 2 Agreement between LLM-generated relevance labels and human reference labels overall and by dataset. Human reference labels were derived from seven annotators as described in Methods. Agreement is the fraction of exact matches, and κ denotes Cohen’s κ . Three-class evaluation distinguishes positives, hard negatives and easy negatives; binary evaluation merges the two negative classes. Macro-P, macro-R and macro-F1 denote class-averaged precision, recall and F1, respectively.

Scope	Label space	n	Matches	Agreement	κ	Macro-P	Macro-R	Macro-F1
All samples	Three class	150	103	0.687	0.510	0.713	0.749	0.699
MS MARCO	Three class	50	37	0.740	0.600	0.756	0.764	0.746
Text2SQL	Three class	50	32	0.640	0.444	0.742	0.824	0.704
TheoremQA (theorems)	Three class	50	34	0.680	0.484	0.670	0.725	0.675
All samples	Binary	150	137	0.913	0.719	0.875	0.846	0.859
MS MARCO	Binary	50	44	0.880	0.705	0.893	0.829	0.851
Text2SQL	Binary	50	49	0.980	0.898	0.917	0.989	0.949
TheoremQA (theorems)	Binary	50	44	0.880	0.593	0.797	0.797	0.797

Extended Data Table 3 Confusion matrix and class-wise performance of LLM-generated relevance labels. Human reference labels were derived from seven annotators as described in Methods. **a**, Three-class confusion matrix. Rows denote human reference categories and columns denote LLM-assigned categories. **b**, Class-wise performance. Human n and LLM n give the number of evaluation pairs assigned to each category. Among the 87 evaluation pairs labelled positive or hard negative by the LLM, 72 matched the human reference category, yielding a pooled precision of 82.8%. The pooled precision combines the counts from both categories rather than averaging their precisions. These are the categories retained in the hard-negative-only training export.

a. Three-class confusion matrix

Human reference	LLM positive	LLM hard negative	LLM easy negative
Positive	22	8	0
Hard negative	5	50	32
Easy negative	0	2	31

b. Class-wise performance

Class	Human n	LLM n	Precision	Recall	F1
Positive	30	27	0.815	0.733	0.772
Hard negative	87	60	0.833	0.575	0.680
Easy negative	33	63	0.492	0.939	0.646

Extended Data Table 4 Data sizes and retrieval performance for the task-specific query-source comparisons. **a**, Nonempty query entries submitted for relevance annotation, valid training records per positive-selection variant, and combined input rows supplied to training. Query entries include repeated query strings. Each valid record contains at least one positive and one hard negative. The two equally sized variants differ in positive-document selection as described in Methods and are concatenated without deduplication; combined input rows therefore equal twice the per-variant count. Input rows count supplied records before batching, rather than distinct queries or examples consumed over training epochs. **b**, nDCG@10 on a 0–100 scale, corresponding to Fig. 6d. The MS MARCO-trained model provides the shared baseline. Reference queries are from ReasonEmbed for the two TheoremQA tasks and official training pairs for CosQA and Synthetic Text2SQL. All task-specific query sources undergo SYNTRA relevance annotation and the shared training procedure.

a. Query and training-data sizes

Task	Query source	Query entries	Records per variant	Combined input rows
TheoremQA (questions)	ReasonEmbed	7,184	5,873	11,746
	SYNTRA	9,852	9,006	18,012
TheoremQA (theorems)	ReasonEmbed	8,861	6,947	13,894
	SYNTRA	9,880	9,517	19,034
CosQA	Official	9,020	6,599	13,198
	SYNTRA	5,938	5,140	10,280
Synthetic Text2SQL	Official	10,000	8,245	16,490
	SYNTRA	9,956	8,419	16,838

b. Retrieval performance

Task	MS MARCO baseline	Reference queries	SYNTRA queries
TheoremQA (questions)	16.90	25.53	32.868
TheoremQA (theorems)	11.61	23.07	22.199
CosQA	32.67	34.89	35.599
Synthetic Text2SQL	56.33	62.97	62.24

Extended Data Table 5 Paired BEIR and NanoBEIR results for the 18 models used in the consistency analysis. Values are suite-level mean nDCG@10 scores on a 0–100 scale. Ranks are computed separately within each model family; rank 1 denotes the highest score.

Model family	Model	BEIR	NanoBEIR	BEIR rank	NanoBEIR rank
Encoder-only	snowflake-arctic-embed-l	55.980	65.360	1	2
	snowflake-arctic-embed-m	54.900	64.420	2	3
	snowflake-arctic-embed-xs	50.150	61.170	8	7
	bge-large-en-v1.5	54.340	65.910	3	1
	bge-base-en-v1.5	53.230	63.760	4	4
	bge-small-en-v1.5	51.660	62.650	5	5
	e5-large-v2	50.560	61.550	6	6
	e5-base-v2	50.300	61.010	7	8
	e5-small-v2	49.040	60.960	9	9
Decoder-only	NV-Embed-v2	63.211	71.480	1	2
	e5-mistral-7b	57.072	66.250	8	7
	gte-Qwen2-7B-instruct	58.860	71.390	5	3
	Qwen3-Embedding-8B	62.758	71.600	2	1
	Qwen3-Embedding-4B	61.578	69.910	3	5
	Qwen3-Embedding-0.6B	55.523	65.620	9	8
	stella_en_1.5B_v5	59.311	70.920	4	4
	jina-embeddings-v5-text-small	57.379	67.190	7	6
	gte-Qwen2-1.5B-instruct	58.289	65.020	6	9

Extended Data Table 6 Rank consistency between NanoBEIR and BEIR within embedding-model families. Spearman’s ρ was computed from paired suite-level mean nDCG@10 scores. Ranks were assigned separately within each family, with rank 1 denoting the highest score. P values are two-sided.

Model family	n	Spearman’s ρ	P value	Within one rank
Encoder-only	9	0.933	2.0×10^{-4}	8/9
Decoder-only	9	0.817	7.0×10^{-3}	6/9

Supplementary Information

Self-improving data synthesis can scale embedding models across domains and tasks

Zucan Lyu, Rui Tang, Jianlyu Chen, Peisheng Jin, Zheng Liu and Defu Lian

This Supplementary Information presents the complete prompts for the principal synthesis workflow, retrieval-task definitions, complete initial and refined instructions for the FiQA case study, and experimental configurations. Shared prompt text is displayed once. Annotation prompts use the candidate-list format with document identifiers used in the formal experiments. Supplementary Note 2 summarizes key experimental settings in Supplementary Tables 1 and 2.

Contents

Supplementary Note 1: Prompts and task-specific instructions	S2
Source and input conventions	S2
Task definition	S2
Identifier and attribute guidance	S5
Query-generation instructions	S9
Query synthesis	S12
Relevance-annotation instructions	S15
Production relevance annotation	S17
Optional branches in the accompanying code	S18
Task descriptors and registered examples	S18
FiQA case study: complete initial and refined instructions	S21
Supplementary Note 2: Experimental configurations	S27

Supplementary Note 1: Prompts and task-specific instructions

Source and input conventions

Prompts are user messages unless a system role is specified. Braced lower-case fields denote runtime inputs; upper-case fields denote shared display blocks that must be expanded before a prompt is sent. Each block contains the complete text of its branch, rather than an excerpt. The repository at <https://github.com/zucanlv/SYNTRA> contains the prompt builders and input formatters. The accompanying `docs/supplementary/` files provide fully expanded templates, optional branches, the complete selected instruction records and registry descriptors.

Prompt wording and branch-specific schemas are preserved. Line wrapping and blank-line spacing are typographic. The accompanying machine-readable files provide fully expanded templates; the source builders specify exact formatting. Pictographs, arrows and non-breaking hyphens use reversible Unicode escape notation; decode these to recover the original symbols. Python JSON expressions denote the serialized runtime value, using `ensure_ascii=False` and `indent=2`, rather than literal code passed to the model.

The principal flow infers task requirements, adapts identifier/attribute and query-generation guidance, synthesizes queries, and refines a relevance-annotation instruction. The refined annotation instruction is subsequently reused as the system message for production annotation. Generating several queries from one source document and annotating several candidate documents for one query are separate operations.

Input fields. `query_type`, `doc_type` and `relevance_def` describe the task. `doc` supplies a source document. `task_instruction` is the task-specific query instruction. `annotation_instruction` or `evaluation_instruction` is the relevance-annotation instruction. `feedback` supplies evaluation feedback; `samples_json` and `samples_brief` supply trial outputs. `refinement_guidance` and `current_guidance` hold the identifier/attribute addendum. `identifiers_brief`, `attributes`, `plans` and `already_selected` supply planning inputs. `num_plans` specifies the requested number of new plans; query generation produces one query for each supplied plan.

Few-shot and demonstration inputs. `target_examples_text` and `examples_text` format labelled target-task examples. `few_shot_section` contains query-only or query-positive examples according to the caller. Labelled examples include a hard-negative field only when available. `demo_blocks` pairs the MS MARCO and ArguAna reference examples with their task definitions; `anchor_block` contains their definitions alone. References equal to the target task are omitted. The task-definition caller uses up to five examples from each reference task in stored order. The complete fixed inputs and formatting functions are available in `Main_Pipeline/Few_Shot_Example.py`, `Few_Shot_Formatter.py` and `HighLevel_Generator.py`.

Trial and candidate inputs. Identifier/attribute validation serializes document identifiers, document text, identifiers and attributes; refinement omits document text. Query-instruction validation and refinement sample up to five and eight trial query-document pairs, respectively. Annotation candidates retain their order. Multi-document calls use zero-based local `doc_id` values, which are mapped back to original identifiers after parsing.

Task definition

Prompt 1: Task-definition generation

Infer the query type, document type and relevance criterion from examples. Replace `CONTRASTIVE_ANALYSIS` with one of the following complete lines, according to whether the target-task examples include hard negatives.

Source: `Main_Pipeline/HighLevel_Generator.py`.

You are an expert Information Retrieval task analyst. Your job is to precisely characterise an IR task from its labeled examples.

Two-Shot Demonstrations

Study these examples carefully — each shows how few-shot examples map to an abstract high-level task definition:

{demo_blocks}

Your Task: Extract the High-Level Task Definition for the target corpus

Few-Shot Examples to Analyse

{target_examples_text}

Step-by-Step Analysis Instructions

Work through the following three steps in your reasoning. Be specific and cite evidence from the examples above.

Step 1 — Query Analysis

- What is the average length of the queries (short/medium/long)?
- What is the vocabulary style (natural language, keyword-style, formal, argumentative, domain-specific)?
- What is the user intent (fact-seeking, opinion-seeking, argument-seeking, research-oriented, ...)?
- What is the domain (general web, biomedical, legal, debate, ...)?
- Summarise into a precise 2-5 word phrase for `query\_type`.

Step 2 — Document Analysis

- What type/structure do the Positive documents have (news articles, scientific abstracts, debate arguments, forum answers, ...)?
- What domain/register are they in?
- How long are they on average?
- Summarise into a precise 2-5 word phrase for `doc\_type`.

{CONTRASTIVE_ANALYSIS}

Step 3 — Relevance Definition Synthesis

Combine the above analysis into ONE precise sentence following this exact template:

"Given a query (<query_type>) and a document (<doc_type>), the document is relevant if <specific criterion based on the contrastive analysis>."

The sentence must:

- Fill in <query_type> and <doc_type> from Steps 1 & 2.
- State the exact criterion (not just "it answers the query") derived from the contrastive analysis.
- Be specific enough that a language model reading only this sentence would produce queries and documents matching the style in the examples above.

Output Format (MANDATORY — valid JSON only, no Markdown fences)

After your step-by-step reasoning, output a single JSON object:

```
{
  "query_type": "2-5 word phrase",
  "doc_type": "2-5 word phrase",
  "relevance_definition": "Given a query (...) and a document (...), the document is relevant if ..."
}
```

Output the JSON as the LAST thing in your response. Do not include any text after the closing brace.

Target examples with hard negatives:

- **Contrastive Analysis**: Compare each Positive document against its Hard Negative. Hard Negatives are topically related but NOT relevant — identify the precise criterion that distinguishes a relevant document from a hard-but-wrong one. This is the single most important signal for writing a precise relevance_definition.

Target examples containing positives only:

- **Contrastive Analysis**: No Hard Negative examples are provided. Focus on what specific aspect of the Positive documents makes them the ideal match for the queries.

Prompt 2: Task-definition validation

Evaluate the inferred definition against the examples.

Source: Main_Pipeline/HighLevel_Generator.py.

```
You are a senior quality evaluator for an information retrieval data synthesis pipeline.

## Your Task
Evaluate whether the high-level task definition below correctly and precisely characterises the IR task
evidenced by the few-shot examples for the target corpus.

## Few-Shot Examples
{examples_text}

## Proposed High-Level Task Definition
- query_type: "{query_type}"
- doc_type: "{doc_type}"
- relevance_definition: "{relevance_def}"

## Evaluation Criteria (check each rigorously)
1. **Query Type Accuracy**: Does `query_type` precisely describe the style, intent, and domain of the
queries in the examples? It should not be too generic (e.g., "query") nor too narrow.
2. **Document Type Accuracy**: Does `doc_type` precisely describe the Positive documents' type,
structure, and domain? It should distinguish these documents from documents in other IR tasks.
3. **Relevance Definition Precision**: Does `relevance_definition` correctly explain why each Positive
document is relevant to its query? Test it mentally: if you applied this definition as a filter, would it accept
all Positive documents and reject all Hard Negatives?
4. **Task Distinctiveness**: Could this definition be confused with another well-known IR task? If so, the
definition is too vague.
5. **Actionability**: Is the definition specific enough that an LLM reading only this sentence would know
what kind of queries to generate and what kind of documents to retrieve?

## Step-by-step Evaluation
Go through each example one by one. For each:
- Does the query fit `query_type`?
- Does the Positive document fit `doc_type` and satisfy `relevance_definition`?
- (If Hard Negative present) Does the Hard Negative correctly fail `relevance_definition`?

Record any per-example issues, then summarise.

## Output Format (MANDATORY — valid JSON only, no Markdown)
{
  "passed": true | false,
  "overall_score": 1-4,
  "feedback": "Concise summary of the main issues, citing specific examples. Or confirm quality if none.",
  "query_type_issues": "Specific issues with query_type, or 'none'.",
  "doc_type_issues": "Specific issues with doc_type, or 'none'.",
  "relevance_def_issues": "Specific issues with relevance_definition (false accepts, false rejects, vagueness),
or 'none'.",
  "suggestions": "Concrete, actionable rewrites or additions to fix the identified issues."
}

IMPORTANT: A "passed" verdict (overall_score >= 3) means the definition is production-ready. Any
systematic mismatch with the examples must set passed=false.
```

Prompt 3: Task-definition refinement

Revise the definition using evaluation feedback.

Source: Main_Pipeline/HighLevel_Generator.py.

```
You are a prompt engineering expert specialising in information retrieval task characterisation.

## Context
We have a high-level task definition for the target corpus that was evaluated and found to have issues.

## Few-Shot Examples (ground truth)
{examples_text}
```

```

## Current (Flawed) Definition
- query_type: "{query_type}"
- doc_type: "{doc_type}"
- relevance_definition: "{relevance_def}"

## Evaluation Feedback
{feedback}

## Quality Anchors (reference definitions for other corpora)
{anchor_block}

## Your Task
Rewrite the high-level task definition for the target corpus to fix ALL issues identified in the feedback.

## Requirements
1. Fix every issue: Address each problem explicitly mentioned in the feedback.
2. Stay grounded: Every claim in the definition must be directly supported by at least one example.
3. Maintain specificity: query_type and doc_type must be 2-5 word phrases. relevance_definition must be one precise sentence.
4. Follow the template: "Given a query (<query_type>) and a document (<doc_type>), the document is relevant if <specific criterion>."

## Output Format (MANDATORY — valid JSON only, no Markdown fences)
{
  "query_type": "2-5 word phrase",
  "doc_type": "2-5 word phrase",
  "relevance_definition": "Given a query (...) and a document (...), the document is relevant if ..."
}

```

Identifier and attribute guidance

Prompt 4: Identifier/attribute generation

Generate document-grounded identifiers and independent query-attribute dimensions. **GUIDANCE_BLOCK** is empty before refinement; subsequently insert the following block at the displayed position.

Source: `Main_Pipeline/IdAttr_Generator.py`.

```

You are a professional data synthesis and NLP expert. Your task is to analyze the document and produce two outputs for downstream diverse and high-quality query generation:
1) Document Identifiers (information anchors)
2) Query Attributes (style/format/constraint dimensions with options)

## Task Context (use this to guide your analysis)
- This document is a: {doc_type}
- Queries that will target this document are: {query_type}
- Relevance criteria: {relevance_def}
- Tailor both identifiers and attribute dimensions so they are maximally useful for generating diverse queries of the above type.

## Core Definitions (must follow strictly)
- Identifier (Information Anchor): An irreplaceable core entity/relationship extracted directly from the document. It serves as the anchor that diverse queries should target. Do NOT include redundant/irrelevant details. Do NOT use external knowledge.
  - Entity-type: concrete objects in the document (e.g., people, places, events, records/results).
  - Relationship-type: unique relationships/logic between entities stated or implied by the document (e.g., relations between people, places, events, records/results).
- Query Attribute: a non-core feature dimension controlling how queries are asked. Attributes should NOT introduce new core facts; they should steer query diversity while keeping identifiers as the information anchors.
  - Example dimensions (for inspiration only; you may adopt, adapt, or omit): reasoning depth, answer form, topic specificity. These are common dimensions, but not all documents or query types need every one.
  - Brainstorm step: Besides considering the above examples, brainstorm 5-7 more independent dimensions by carefully analyzing THIS document (domain, topic, structure, content type, and what kinds of queries would realistically target it). Each dimension must be independent (no overlap with others), practical for query generation, and have 2-4 mutually exclusive options in mind.
  - Selection step (reason step-by-step): First, analyze the Task Context and the Input Document. Then strictly filter the full pool (example-inspired + brainstormed, typically 8-10 candidate dimensions) using the Scientific Approach below, and select exactly 3-6 dimensions that are most useful for diversifying queries over THIS document. Output only these 3-6 selected dimensions in the final JSON.

```

\U0001f52c Critical: Scientific Approach to Attribute Design

Before finalizing your Identifiers and Attributes, you MUST perform a "Task Alignment Analysis". Answer these questions:

1. **WHAT** makes this task unique?
 - Identify specific patterns, challenges, or domain characteristics of {query_type}. (Do not just repeat the task context; explain what it means in practice).
2. **WHY** is each selected attribute suitable?
 - Explain the MATCH between the attribute and the task characteristics. How does it help generate high-quality queries?
3. **MULTIPLE OPTIONS TEST** (Crucial):
 - Does the attribute genuinely have multiple valid, mutually exclusive options for THIS specific task?
 - **CRITICAL RULE:** If the task context restricts an attribute to only ONE valid option, you MUST NOT select it! (e.g., If it's a reasoning-intensive task, "reasoning depth" should always be "high", so "reasoning depth" is NOT a valid diversifying attribute).

\U26a0\ufe0f This is not a template to fill - this is how experts think.

Blindly applying standard methods without understanding WHY is not acceptable.

High-Quality Query Attributes Criteria

1. Strictly align with the characteristics of {query_type}.
2. Each selected dimension must have 2-4 mutually exclusive, actionable options applicable to THIS document.

{GUIDANCE_BLOCK}

Input Document

doc: {doc}

Output Format Requirements (MANDATORY, machine-parsable, no deviation)

- Output MUST be a single valid JSON object (RFC 8259).
- Output MUST contain ONLY the JSON object: no Markdown, no code fences, no commentary, no extra keys, no trailing text.
- Use double quotes for all strings and keys. No trailing commas.
- The JSON must start with a single left brace and end with a single right brace.
- Note: In this prompt template you may see doubled braces (e.g., `{` / `}`) due to Python string escaping; your OUTPUT JSON must use normal single braces.

Hard constraints

- identifiers: 3-10 items; no duplicates/overlap; each must be a standalone anchor.
- attributes: 3-6 dimensions total. Produce them by (1) using the example dimensions only as inspiration, (2) brainstorming from THIS document, then (3) filtering via the Scientific Approach. Each dimension unique; each must have 2-4 options.
- options must be mutually exclusive, actionable, and applicable to THIS document's domain/topic for practical query generation.

JSON schema (fixed keys; output MUST follow exactly)

```
{
  "identifiers": [
    {
      "type": "entity" | "relationship",
      "identifier": "string (copied/derived from document only; no external info)",
      "label": "string (3-8 words; short unique label)"
    }
  ],
  "attributes": [
    {
      "dimension": "string (clear independent dimension name)",
      "options": ["string", "string", "string", "string (optional)"]
    }
  ]
}
```

Refined Guidance (from prompt calibration — follow strictly)

{refinement_guidance.strip()}

Prompt 5: Identifier/attribute validation

Evaluate trial outputs for task alignment, grounding and usable diversity.

Source: Main_Pipeline/IdAttr_Generator.py.

```
You are a senior quality evaluator for an information retrieval data synthesis pipeline.

## Your Task
Evaluate whether the generated Identifiers and Attributes across the sample documents below are well-suited for producing diverse, high-quality {query_type} queries over {doc_type} documents.

## Step-by-step guideline:
1. Grasp the task. Understand the context and what good {query_type} queries look like for this task. What characterizes {query_type} queries here?
2. Learn from examples. Use the few-shot examples to infer the STYLE, DEPTH, and FOCUS of real {query_type} queries. The identifiers and attributes you evaluate should be able to yield {query_type} queries that match these in character.
3. Evaluate the samples one by one. There are multiple sample documents; evaluate each sample's identifiers and attributes individually and record issues as you go:
    i. Compare to the task. For each sample in turn, check whether its identifiers and attributes align with what good {query_type} queries need: diversity, grounding in the document, and actionable dimensions. Note any mismatch for that specific sample.
    ii. Apply the criteria. For each sample, evaluate its identifiers and attributes against the evaluation criteria (Identifier Quality, Attribute-Task Alignment, Multiple Options Test, Coverage, Actionability) and decide whether they are fit for the task. If there are issues, record them clearly for that sample (e.g., "Sample 2 (doc_id X): attribute dimension Y should not have multiple options in this task—it has a fixed characteristic and must be flagged.").
    Keep a clear record of per-sample issues so you can aggregate them in Step 4.
4. Summarize. Based on the per-sample issues you noted in Step 3, write a concise summary of the main issues—or confirm quality if none. This summary will go to the prompt engineer for prompt refinement.
5. Format. Respond strictly in the required JSON output format.

## Task Context
- Query Type: {query_type}
- Document Type: {doc_type}
- Relevance Definition: {relevance_def}

## Few-Shot Examples (gold-standard query-document pairs for this task)
{few_shot_section}

## Evaluation Criteria (check each rigorously)
1. Identifier Quality: Are identifiers diverse, non-redundant, and grounded in the document? Do they cover the key information anchors that {query_type} would realistically target?
2. Attribute-Task Alignment: Do the attribute dimensions genuinely help diversify {query_type}? Would each dimension produce meaningfully different queries when its options are varied?
3. Multiple Options Test: Does every attribute dimension have 2-4 mutually exclusive options that are ALL valid for this task? If the task context effectively restricts a dimension to a single option, it MUST be flagged.
4. Coverage: Across all samples, is there a healthy variety of attribute dimensions, or do the samples all use near-identical dimensions?
5. Actionability: Can a query generator actually use these identifiers + attributes to write concrete, natural {query_type} queries?

## Sample IdAttr Outputs (generated by the current prompt)
{samples_json}

## Scoring Rubric (assign exactly one integer score 0–3)
> Important: The parenthetical examples (e.g., ...) in each score level are purely illustrative of the type of issue at that severity. Do not treat them as a checklist — base your score on the holistic pattern across all five evaluation criteria above, not just the dimensions that happen to be mentioned in the examples.

- Score 3 — Production-Ready: Identifiers are diverse, specific, and fully grounded in each document's content. All attribute dimensions are genuinely task-aligned with 2–4 mutually exclusive options that are ALL valid for this query type (Multiple Options Test passes for every dimension). Coverage across samples shows healthy variety — samples do not all share near-identical dimension sets. A query generator can immediately produce diverse, natural {query_type} queries without any revision.
- Score 2 — Acceptable: Mostly sound, with only minor isolated issues — e.g., 1–2 identifiers that are slightly generic, or one attribute option that is marginally weak. Crucially, no Multiple Options Test violations (this is a hard disqualifier for Score 2+), and no systemic dimension repetition across samples. A query generator can still produce useful, diverse queries; only small prompt tweaks are needed.
```

- ****Score 1 — Needs Revision****: Systematic problems undermine query diversity or groundedness. This includes: any Multiple Options Test violation (one or more attribute dimensions effectively restricted to a single valid option), identifiers that are consistently ungrounded or generic across samples (not anchored to document-specific content), or near-identical dimension sets repeating across samples with little variety. A query generator would produce repetitive, low-quality queries.

- ****Score 0 — Unusable****: Fundamentally broken. Identifiers are absent, fabricated, or entirely off-topic relative to the documents. Attribute dimensions are structurally invalid, nonsensical, or all collapse to a single valid option. The output cannot be used for query generation at all.

Output Format (MANDATORY — valid JSON only, no Markdown)

Output a single JSON object:

```
{
  "overall_score": 0-3,
  "passed": true if overall_score >= 2 else false,
  "feedback": "Concise summary of the main issues (or confirmation of quality). Be specific: cite which dimensions/identifiers are problematic and WHY. The summary will be sent to the prompt engineer to refine the prompt.",
  "identifier_issues": "Specific issues with identifiers, or 'none'.",
  "attribute_issues": "Specific issues with attribute dimensions/options, or 'none'.",
  "suggestions": "Concrete, actionable suggestions for improving the generation prompt. Focus on what guidance to ADD, not what to remove."
}
```

IMPORTANT: Be constructively critical. A "passed" verdict (overall_score >= 2) means the outputs are ready for production use. If you see ANY Multiple Options Test violation or systematic identifier/coverage issue, assign score <= 1 and set passed=false.

Prompt 6: Identifier/attribute refinement

Produce a reusable task-level addendum. For an empty cache, PREVIOUS_GUIDANCE is the sentence below; otherwise it is {current_guidance.strip()}.

Source: Main_Pipeline/IdAttr_Generator.py.

You are a prompt engineering expert specializing in information retrieval data synthesis.

Context

We have a prompt that instructs an LLM to generate ****Identifiers**** (information anchors) and ****Attributes**** (query diversity dimensions) for documents. The generated IdAttr will be used downstream to produce diverse ****{query_type}**** queries over ****{doc_type}**** documents.

- Relevance Definition: {relevance_def}

Problem

The current prompt was evaluated on sample documents and received the following feedback. Note: if this feedback contains ****specific examples**** (e.g., "Identifier X for doc Y is wrong because ..."), you must ****abstract**** from these examples into ****task-level**** lessons.

Evaluation Feedback

{feedback}

Sample Outputs (from the current prompt)

{samples_brief}

Previous Guidance (the last addendum that was tried)

{PREVIOUS_GUIDANCE}

Your Task (two mental steps — output only the final guidance)

****Step 1 — Summarize and abstract (do this mentally, do not output it):****

- Read the feedback and identify ****what went wrong**** in the examples.

- ****Generalize****: For each type of issue, extract the ****underlying principle or pattern**** that would help the generator avoid that class of error on ****any**** document, not just the ones in the sample. Ask: "What rule or criterion would prevent this kind of mistake in general?"

****Step 2 — Write Refined Guidance:****

Produce a ****Refined Guidance**** addendum that encodes those ****task-level, query-agnostic**** principles. This guidance will be inserted into the generation prompt and must apply to ****all**** future IdAttr generation, not to specific docs or queries.

Requirements for the Refined Guidance

1. ****Query-agnostic and task-level only****: Do NOT include doc-specific or query-specific fixes (e.g., do not reference particular doc_id, query text, or "for the query about X do Y"). Write only ****general rules**** that improve IdAttr generation for the whole task.

2. **Derived from feedback**: Every recurring issue type in the feedback should be reflected as a **general principle** in the guidance.
3. **Preserve what works**: If parts of the previous guidance are effective, keep them.
4. **Specific and actionable**: Say exactly WHAT to prefer/avoid and WHY for this task type.
5. **Concise**: 100-300 words. Dense, high-signal, imperative tone (e.g., "Prioritize ...", "Avoid ...", "For {query_type}, ensure ...").
6. **Only guidance about identifiers and attributes generation**: No meta-commentary, no JSON, no Markdown fences.

Output Format

Output **ONLY** the refined guidance text (plain text, no JSON wrapping, no Markdown fences). This text will be inserted verbatim into the generation prompt.

There was no previous guidance that was tried before.

Query-generation instructions

Prompt 7: Instruction construction

Construct a task-specific query-generation instruction from few-shot query-positive document pairs.

Source: Main_Pipeline/QueryInstr_Generator.py.

You are an expert in information retrieval and synthetic query synthesis.

Task Context

- Query Type: {query_type}
- Document Type: {doc_type}
- Relevance Definition: {relevance_def}

Few-Shot Examples — (Query \u2194 Positive Document relevance pairs)

Each example shows a gold-standard {query_type} and a real {doc_type} that is relevant to it.

Use these pairs to infer how a real query author formulates the query-side expression that makes the document relevant.

{few_shot_section}

Your Task — Two phases

Phase 1 — Per-example analysis (think step-by-step for EACH example):

For each (document, query) pair above, reason through:

1. Who is the real-world entity producing this type of query? What is their role, context, and goal?
2. What query-side purpose or expression would make the positive document relevant, and how is it formulated by the query author?
3. Which document details are authentic to include in this task's queries, and which details should be included, omitted, or transformed according to the query-document relationship demonstrated by the few-shot examples?
4. Does the query reflect the authentic voice, purpose, and framing of the query author, according to the query-document relationship demonstrated by the few-shot examples?

Phase 2 — Cross-example generalisation (the actual output):

After analysing all examples, output **exactly two sections** in the following order:

Synthesizer Role

Write 1-2 sentences describing the role an LLM should *adopt* when synthesising {query_type} queries for this task.

The role must:

- Name who the producer is (e.g. "an everyday internet user", "a competitive debater", "a biomedical researcher")
- Describe what they are doing and what their goal is (e.g. "typing a query into a search engine to find an answer", "formulating a claim to argue in a debate round")
- Be grounded in the patterns observed across ALL few-shot examples — do NOT invent a role not supported by the data

Query Extraction Process

Write a short, bulleted list of concrete steps the LLM should follow when reading a new {doc_type} and synthesising a {query_type} from it.

Each step must:

- Be stated as a general rule applicable to `*any* {doc_type}` in this corpus — do NOT reference specific entities, dates, or facts from the example documents
- Guide the LLM to infer a plausible query-side purpose or expression that would make the document relevant from the `*producer's perspective*` (as defined in Synthesizer Role)
- Specify which document-derived signals should shape the query, which details should be included/omitted/generalized, and how to frame the query according to the query-document relationship demonstrated by the few-shot examples

Hard constraints for BOTH sections

- Every sentence must apply to any new `{doc_type}` encountered in this task, not just the examples above.
- Do NOT quote, paraphrase, or reference specific content from the example documents.
- The two sections together must be immediately usable as part of a system prompt without further editing.

Output Format

Output ONLY the two sections above, in order, with their exact headings (`#### Synthesizer Role` and `#### Query Extraction Process``).

No JSON, no preamble, no per-example commentary, no additional text.

Prompt 8: Style guidance

Extract the query style and register required by the task.

Source: `Main_Pipeline/QueryInstr_Generator.py`.

Role

You are a Senior Query Synthesist and Style Analyst. Your expertise lies in analyzing text patterns to create precise, actionable instructions for generating high-quality synthetic queries.

Task Context:

- Target Output (Query Type): `{query_type}`
- Source Input (Document Type): `{doc_type}`
- Relevance Criteria: `{relevance_def}`

Few-Shot Example Queries:

`{few_shot_section}`

Instruction:

Based on the task context and the patterns observed in the few-shot examples, synthesize a "Task-Specific Style Guide". This guide will serve as the "gold standard" for an LLM to generate high-quality `{query_type}` queries for `{doc_type}` documents.

Output Requirements:

You must actively analyze and distill features from both the Task Context and the Few-Shot Example Queries above. You must observe, abstract, and summarize what makes these queries fit the task, so the resulting guide can steer high-quality query generation. Address the following:

1. Linguistic DNA: From the examples, infer and state the tone (e.g. inquisitive, keyword-heavy), structural syntax, typical length, and any other recurring linguistic traits.
2. Relevance Heuristics: From the task context and examples, identify which information patterns in a `{query_type}` should be emphasised to satisfy the relevance criteria (`{relevance_def}`).
3. Negative Constraints: List "anti-patterns"—phrasing, styles, or excessive detail that would harm data purity and should be avoided.

Output Format:

Produce a compact, bulleted instruction set suitable for embedding in a system prompt for synthetic query generation. Organise your output under the three subheadings above: Linguistic DNA, Relevance Heuristics, and Negative Constraints (with bullet points listed under each).

Output only these subheadings and bullet points in plain text; no other text.

Prompt 9: Instruction validation

Evaluate whether generated queries satisfy task requirements and match the example style.

Source: `Main_Pipeline/QueryInstr_Generator.py`.

You are a senior quality evaluator for an information retrieval data synthesis pipeline.

Your Task

Evaluate whether the `**generated queries**` below match the expected style, tone, and characteristics of real `**{query_type}**` queries for this task.

Step-by-step guideline:

1. **Understand the task.** Be clear on what good {query_type} queries look like in this setting and what distinguishes them.
2. **Evaluate the (document, query) pairs one by one.** For each pair, evaluate the generated query against ALL five criteria below and record issues as you go.
 - i. **Compare to the reference set.** Does the query match the golden queries in style, tone, length, and the level of abstraction/specificity?
 - ii. **Apply the criteria.** Record any issue clearly per pair (e.g., "Pair 3: over-specifies document details, too long").
3. **Summarize.** Provide a concise summary of the main issues — or confirm quality is acceptable. If citing a problematic pair, use it only as evidence; the feedback itself must generalize the underlying task-level issue rather than preserve sample-specific entities, dates, facts, or wording. This summary will be used to refine a reusable query synthesis instruction.
4. **Output.** Respond strictly in the required JSON output format.

Task Context

- Query Type: {query_type}
- Document Type: {doc_type}
- Relevance Definition: {relevance_def}

Few-Shot Golden Queries (style + abstraction benchmark — compare generated queries to these)
{few_shot_queries_section}

Current Query Synthesis Instruction (used to generate the queries below)
{query_instruction}

Generated (Positive Document \u2194 Query) Pairs (sample)
{pairs_section}

Evaluation Criteria (check each rigorously)

1. **Style Match:** Do the generated queries resemble the few-shot example queries in tone, length, vocabulary, and structure?
2. **Task Alignment:** Are the queries the kind of {query_type} that would realistically be issued against {doc_type} documents?
3. **Naturalness:** Do the queries read like something a real producer of this query type would write, or do they sound artificial/templated?
4. **User Authenticity:** Does each query feel like it was genuinely produced by the role described in the Synthesizer Role section of the instruction — i.e., someone acting in that role *without* having access to the source document? The key question is whether they are *authentic to the producer's perspective and intent in this task*. Penalise queries that merely describe or echo document content rather than reflecting genuine producer intent.
5. **Instruction Effectiveness:** Does the current instruction successfully guide the LLM on all four criteria above, including the Synthesizer Role and Query Extraction Process sections?

Scoring Rubric (assign exactly one integer score 0–3)

> **Important:** Base your score on the holistic pattern across **all five criteria** above.

- **Score 3 — Production-Ready:** Generated queries closely match the golden queries in style, authenticity, naturalness, and the query-document relationship demonstrated by the examples. The Synthesizer Role and Query Extraction Process sections guide the LLM to produce queries from the genuine producer's perspective.
- **Score 2 — Acceptable:** Queries are broadly on-style and authentic, with only minor isolated deviations from the golden examples' producer perspective or query-document relationship. Only small targeted adjustments are needed.
- **Score 1 — Needs Revision:** Systematic issues across most queries — e.g., consistently using the wrong producer role, wrong level of specificity, wrong register, wrong relationship to the positive document, or otherwise missing the authentic producer perspective demonstrated in the golden examples. The Synthesizer Role or Query Extraction Process section is missing, vague, or not guiding correctly.
- **Score 0 — Unusable:** Queries are fundamentally off-task, incoherent, or incompatible with the task's expected query type and query-document relationship.

Output Format (MANDATORY — valid JSON only, no Markdown)

Output a single JSON object:

```
{
  "overall_score": 0-3,
  "passed": true if overall_score >= 2 else false,
  "feedback": "Concise summary of the main issues. Be specific: cite example pairs that are problematic and explain WHY.",
  "style_issues": "Specific style mismatches vs few-shot examples, or 'none'.",
  "authenticity_issues": "Whether queries fail to reflect the genuine producer perspective (e.g. just echo document content, wrong role, wrong intent), or 'none'.",
  "instruction_issues": "What the query instruction gets wrong or misses, or 'none'.",
  "suggestions": "Concrete, actionable suggestions for improving the query instruction."
```

```
}
IMPORTANT: A "passed" verdict (overall_score >= 2) means the queries are production-ready. If there is
ANY systematic authenticity or style mismatch across multiple pairs, assign score <= 1 and set
passed=false.
```

Prompt 10: Instruction refinement

Revise the reusable query-generation instruction using feedback.

Source: `Main_Pipeline/QueryInstr_Generator.py`.

```
You are a prompt engineering expert specializing in information retrieval query synthesis.

## Context
We have a "Query Synthesis Instruction" that is embedded as a system prompt to steer an LLM toward
producing {query_type} queries for {doc_type} documents.
- Relevance Definition: {relevance_def}

## Few-Shot Golden Queries (the gold standard — queries MUST match this style and abstraction level)
{few_shot_queries_section}

## Problem
The current instruction was evaluated and received the following feedback:

### Evaluation Feedback
{feedback}

### Sample (Positive Document \u2194 Generated Query) Pairs (produced by the current instruction)
{pairs_section}

### Current Query Synthesis Instruction
{query_instruction}

## Your Task
Rewrite the Query Synthesis Instruction to fix all issues identified in the feedback. The rewritten
instruction must produce queries that closely match the few-shot examples in style, tone, length, and
domain conventions.

## Requirements
1. Preserve what works: If parts of the current instruction are effective, keep them.
2. Fix what's broken: Address every issue in the feedback — including authenticity mismatches where
generated queries fail to reflect the genuine producer perspective (e.g., they just echo document content,
adopt the wrong register, or lack the intent of the role described).
3. Be specific: Instead of "use appropriate tone", specify exactly what tone (e.g., Tone: Analytical,
assertive, and occasionally contentious to reflect a strong stance.").
4. Update the Synthesizer Role section if needed: If the feedback indicates the LLM is not adopting the
right producer identity, sharpen the role description to better convey who is writing the query and why.
5. Update the Query Extraction Process section if needed: If the feedback indicates queries are not
reflecting the authentic producer perspective, revise the extraction steps to more explicitly guide the LLM
to think from the producer's viewpoint.
6. Format: Output as a compact, bulleted query style guidelines set directly embeddable as a system
prompt. No JSON wrapping. Keep the same style and structure as the current query style guidelines above
(e.g. same level of detail, sectioning, and bullet conventions).

## Output Format
Output ONLY the rewritten instruction text (plain text with bullet points and section headings). No JSON,
no Markdown fences, no commentary.
```

Query synthesis

The following blocks are shared by blueprint planning and query realization. Expand each named block exactly at its displayed position. Planning selects information needs; realization writes one query per supplied blueprint.

QUERY_INPUTS:

```
You will be given:
1) A {doc_type} (the ONLY source of truth)
2) A set of identifiers (information anchors extracted from the document)
3) A set of query attribute dimensions (each has multiple options)
```

TASK_CONTEXT:

```
**Task Context:**
- Target Output (Query Type): {query_type}
- Source Input (Document Type): {doc_type}
- Relevance Criteria: {relevance_def}
**Few-Shot Examples:**
{few_shot_section}
```

STYLE_GUIDANCE:

```
**Task-Specific Style Guidance:**
{task_instruction}
```

SOURCE_DOCUMENT:

```
**Input document:**
{doc}
```

JSON_ONLY:

- Output MUST be a single valid JSON object (RFC 8259).
- Output MUST contain ONLY the JSON object: no Markdown, no code fences, no commentary, no extra keys, no trailing text.

JSON_SYNTAX:

- Use double quotes for all keys and strings. No trailing commas.
- JSON must start with '{' and end with '}'.

BLEUPRINT_FIELDS:

```
{
  "used_identifier": ["identifier string"],
  "used_attributes": {
    "dimension name": "one option",
    "dimension name": "one option"
  }
}
```

Prompt 11: Blueprint planning

Use the shared blocks above. The branch-specific instructions and output schema below are complete.

Source: Main_Pipeline/DiverseQuery_Generator.py.

```
You are planning diverse search query "blueprints" for information retrieval.
{QUERY_INPUTS}
{TASK_CONTEXT}
**Your task:** generate exactly {num_plans} diverse BLUEPRINTS. Each blueprint contains:
- exactly 1 chosen identifier (copy EXACTLY from the provided identifiers list)
- ONE attribute group: choose exactly ONE option for EACH attribute dimension
**Important:** In this stage, DO NOT write any search queries. Only output the selections (identifiers +
attributes).
{STYLE_GUIDANCE}
Use this guidance to inform your blueprint planning decisions.
```

```

**Hard constraints:**
- Each blueprint must contain exactly 1 identifier — no more, no less.
- Do NOT invent identifiers or attribute options.
- Avoid redundancy: do not repeat the same identifier+attribute-group combination.
**Diversity planning (across these {num_plans} blueprints, and considering any already selected blueprints):**
- Maximize coverage of different identifiers.
- For each attribute dimension, try to cover as many different options as possible.
**Already selected blueprints (DO NOT repeat these combinations):**
{json.dumps(already_selected, ensure_ascii=False, indent=2)}
{SOURCE_DOCUMENT}
**Identifiers (choose from these; copy identifiers exactly):**
{json.dumps(identifiers_brief, ensure_ascii=False, indent=2)}
**Attribute dimensions (choose exactly one option per dimension per blueprint):**
{json.dumps(attributes, ensure_ascii=False, indent=2)}
**Output format requirements (MANDATORY):**
{JSON_ONLY}
- Do NOT include any "already_selected" plans in your output (do not repeat them); output ONLY NEW plans in the "plans" object.
{JSON_SYNTAX}
**JSON schema (fixed keys; output MUST follow exactly):**
{
  "plans": [
    {
      {BLUEPRINT_FIELDS}
    }
  ]
}

```

Prompt 12: Query realization from blueprints

Use the shared blocks above. The branch-specific instructions and output schema below are complete.

Source: `Main_Pipeline/DiverseQuery_Generator.py`.

```

You are a world-class synthetic query writer for information retrieval.
{QUERY_INPUTS}
4) A list of BLUEPRINTS (each blueprint already specifies exactly 1 identifier and exactly one option per attribute dimension)
{TASK_CONTEXT}
**Your task:** Generate a list of {len(plans)} distinct queries, strictly adhering to the provided blueprints.
{STYLE_GUIDANCE}
**Task Execution:**
1. Iterate through the provided Blueprints one by one.
2. Apply the Task-Specific Style Guidance to determine the appropriate linguistic register and domain-specific terminology (e.g., professional, academic, or casual).
3. For each blueprint, craft a high-quality query that implicitly but clearly reflects all assigned Attributes and targets the specified Identifiers.
4. Ensure the query is a perfect marriage of the blueprint's rigid requirements and the domain's stylistic nuances.
**NO reselection rule (STRICT):**
- You MUST NOT change the blueprint selections.
- For each query item in the output, you MUST include the blueprint's used_identifier and used_attributes EXACTLY as separate JSON fields (for reference). Do NOT embed them inside the "query" text.
**Attribute expressibility rule (STRICT):**
- The query text must clearly reflect the chosen options for each dimension (observable from wording).
- Even if a blueprint presents a logical challenge, you must provide the most plausible grounded query without altering the constraints.
{SOURCE_DOCUMENT}
**Identifiers reference:**
{json.dumps(identifiers_brief, ensure_ascii=False, indent=2)}
**Attribute dimensions reference:**
{json.dumps(attributes, ensure_ascii=False, indent=2)}

```



```

**Blueprints (write one query per blueprint, in the same order):**
{json.dumps(plans, ensure_ascii=False, indent=2)}
**Output format requirements (MANDATORY):**
{JSON_ONLY}
{JSON_SYNTAX}
**JSON schema (fixed keys; output MUST follow exactly):**
{
  "queries": [
    {
      "query": "string (one query matching the task-specified query type)",
      BLUEPRINT_FIELDS
    }
  ]
}

```

Relevance-annotation instructions

Prompt 13: Annotation-instruction construction

Construct the task-specific annotation instruction for a query and a list of candidate documents carrying `doc_id` values.

Source: `Main_Pipeline/AnnoInstr_Generator.py`.

You are an expert LLM annotator for information retrieval evaluation. Create a concise system instruction for another LLM annotator to evaluate documents and classify them into three categories: positive, hard negative, or easy negative.

****Critical thinking guideline:**** The instruction should push the annotator to reason from the specific query-document relationship in this task: identify what would truly satisfy the query, what only looks relevant, and why the candidate crosses or fails that boundary. Avoid generic relevance advice.

The system instruction should specify following parts clearly:

****1. Task Context:****

- Query Type: {query_type}
- Document Type: {doc_type}
- Relevance Definition: {relevance_def}

****2. Scoring & Classification System:****

The instruction must define a task-specific 0-3 absolute scoring rubric grounded in the Query Type, Document Type, and Relevance Definition above. Guide the another annotator to reason about whether each document satisfies the task's stated relevance criteria, and to distinguish strong satisfaction, valid-but-less-complete positive cases, related-but-not-relevant cases, and clearly irrelevant cases.

Keep this label mapping fixed:

- ****Score 3 -> positive****: Strongly satisfies the task-specific relevance definition.
- ****Score 2 -> positive****: Satisfies the task-specific relevance definition well enough to be a valid positive, but is not enough to be a score 3 document.
- ****Score 1 -> hard_negative****: The document looks plausible for the query in this task, but fails the task-specific relevance definition.
- ****Score 0 -> easy_negative****: Clearly unrelated or outside the task-specific relevance criteria.

****3. Hard Negative Guidance:**** The instruction should explain hard negatives with respect to the task's actual relevance criteria: they may appear plausible for this task, but they fail the definition of relevance.

****4. Input Format:****

- One query string
- List of candidate documents with identifiers (`doc_id`)

****5. Annotation Protocol:****

1. Process sequentially: Annotate each document exactly once in the order provided. Ensure no `doc_id` is duplicated or omitted in the final output.
2. Independent evaluation: Evaluate each document in isolation based strictly on its relationship with the query. Do not allow the quality or content of other documents in the list to influence the current score (avoid relative ranking; use absolute scoring).

****6. Output Format (valid JSON only):****

```

{"query": "original query",
 "annotations": [{"doc_id": "id", "score": 0-3, "classification": "positive|hard negative|easy negative",
 "reasoning": "brief"}],
 "positive_doc_ids": [...], "hard_negative_doc_ids": [...], "easy_negative_doc_ids": [...]}

```

The instruction should be concise, emphasize hard negative identification, and require valid JSON output. Output **ONLY** the system instruction text. ****DO NOT**** wrap the instruction in a JSON object!

Prompt 14: Calibration-feedback generation

After calibration labels are compared with the reference labels, convert the confirmed mismatches into actionable feedback for instruction refinement. The label comparison supplies `compare_details`; `mismatches_json` serializes its mismatches with `ensure_ascii=False` and `indent=2`.

Source: `Main_Pipeline/AnnoCalibration_Generator.py`.

You are a senior prompt evaluator for an information retrieval annotation pipeline.

Your Task

The calibration labels are already confirmed, and pass/fail has already been decided by exact label comparison. Do not re-grade the calibration set and do not decide whether the run passed.

Your job is to turn the confirmed misclassifications below into constructive feedback for rewriting the annotation instruction. The feedback must explain what the current instruction is causing the annotator to misunderstand and how the next instruction should prevent the same mistakes.

Step-by-step guideline

1. **Understand the task boundary.** Read the query type, document type, and relevance definition. Identify what a document must actually do to count as positive for this task.
2. **Analyze each mismatch.** For every mismatch, compare the expected label with the predicted label. Use the query, document text, expected reasoning, and predicted reasoning to infer the likely failure mode.
3. **Group repeated failure patterns.** Infer the recurring failure modes from the mismatches themselves instead of forcing them into predefined error categories.
4. **Diagnose instruction gaps.** Explain which missing or weak parts of the current annotation instruction likely allowed these errors.
5. **Write actionable refinement guidance.** Suggest concrete wording-level improvements for the next instruction. Focus on task-specific relevance criteria and hard-negative boundaries, not generic advice like "be more careful".

Task Context

- Query Type: {query_type}
- Document Type: {doc_type}
- Relevance Definition: {relevance_def}

Current Annotation Instruction

{annotation_instruction}

Deterministic Calibration Result

- Total labels: {compare_details.get('total', 0)}
- Correct labels: {compare_details.get('correct', 0)}
- Mismatches: {len(mismatches)}

Confirmed Misclassifications

Each item contains the query/document pair, the confirmed expected label, the annotator's predicted label, and any available expected/predicted reasoning.

{mismatches_json}

Feedback Goal

Write structured feedback that guides the instruction refiner's thinking. The fields below are not separate checkboxes; together they should form a coherent diagnosis of the global failure pattern.

Do not merely enumerate individual errors. Use specific mismatches as evidence, but convert them into generalizable guidance that can prevent the same class of mistakes on future query-document pairs. Avoid generic advice such as "be more careful" or "think harder".

Output Format (MANDATORY — valid JSON only, no Markdown)

Output a single JSON object:

```
{
  "feedback": "Concise global summary of the main failure pattern. Cite representative mismatches only as evidence for the broader diagnosis.",
  "positive_issues": "Global issues involving positive classifications, especially cases where plausibility caused false positives; use 'none' if not relevant.",
  "negative_issues": "Global issues involving hard/easy negative distinctions or missed hard-negative boundaries; use 'none' if not relevant.",
  "instruction_gaps": "The missing, weak, or ambiguous instruction guidance that likely caused the observed error pattern.",
  "suggestions": "Concrete task-specific changes the next instruction should make to prevent this class of errors."
}
```

Prompt 15: Annotation-instruction refinement

Revise the instruction using calibration feedback while preserving the candidate-list annotation contract. `annotation_prompt_mode` specifies the contract with document identifiers.

Source: `Main_Pipeline/AnnoInstr_Generator.py`.

```
You are a prompt engineering expert specializing in information retrieval evaluation.

## Context
We have an annotation instruction that guides an LLM to evaluate one or more candidate documents for a
query and classify each document as positive, hard_negative, or easy_negative.
- Query Type: {query_type}
- Document Type: {doc_type}
- Relevance Definition: {relevance_def}
- Annotation Prompt Contract: {annotation_prompt_mode}

## Problem
The current instruction was evaluated and received the following feedback:

### Evaluation Feedback
{feedback}

### Current Instruction
{annotation_instruction}

## Your Task
Rewrite the annotation instruction to fix all issues identified in the feedback. The rewritten instruction
must produce more accurate and consistent classifications for the multi-document annotation setting.

## Requirements
1. **Preserve the scoring system**: Keep the 0-3 score and three-class classification
(positive/hard_negative/easy_negative).
2. **Preserve the multi-doc output format**: Keep the JSON schema with annotations, positive_doc_ids,
hard_negative_doc_ids, and easy_negative_doc_ids, and require exact doc_id echoing for each candidate.
3. **Preserve what works**: If parts of the current instruction are effective, keep them.
4. **Fix what's broken**: Address every issue in the feedback, especially misclassification patterns and
weak relevance-boundary guidance.
5. **Add task-specific guidance**: Include concrete criteria specific to {query_type}/{doc_type} that help
distinguish true positives from hard negatives.
6. **Be precise about boundaries**: Clearly define where positive ends and hard_negative begins for this
task.
7. **Preserve the instruction style**: Keep the same style and structure as the current annotation
instruction above (e.g. same level of detail, sectioning, and bullet conventions).

## Output Format
Output ONLY the rewritten annotation instruction (plain text). No JSON wrapping, no Markdown fences.
This will be used as a system prompt for the annotator LLM.
```

Production relevance annotation

Prompt 16: Annotation system and user messages

Source: `Main_Pipeline/Batch_Candidate_Annotator.py`. The refined task-specific annotation instruction becomes the system message.

System message:

```
{evaluation_instruction}
```

User message:

```
Query: {query}
Candidate documents (each with doc_id and doc):
{candidate_document_blocks}
```

Each `candidate_document_blocks` entry is `[doc_id: {doc_id}]`, followed by a newline, the document text and two newlines. Local identifiers are zero-based positions in the candidate chunk.

Parser-retry user message: the original conversation is retained and the following repair request is added.

Your output was not a valid JSON or was missing required fields. Please output **ONLY** the strictly valid JSON object with annotations, `positive_doc_ids`, `hard_negative_doc_ids`, and `easy_negative_doc_ids` as specified in the system instructions.

Optional branches in the accompanying code

The complete expanded templates in `docs/supplementary/shared_prompt_templates.json` also include the query-generation ablation without identifiers (`DiverseQuery_Generator.py`), the alternative annotation-instruction LLM validator (`AnnoInstr_Generator.py`), and document-type filtering with or without positive prototypes (`DocType_Filter.py`). These have distinct functions or output contracts and are retained as separate templates. Input-rendering code and fixed demonstrations are supplied in the source files identified above.

Task descriptors and registered examples

The registry spans 24 retrieval tasks and contains 121 examples: five per task, except six for DBpedia-entity. Each entry below gives its query type, document type and complete relevance definition. These definitions are reproduced from the released registry. Task-definition prompts infer and refine requirements from examples supplied at runtime. Example-source metadata and available identifiers accompany the code in `docs/supplementary/task_registry_descriptors.json`.

MS MARCO

Query type: Web query. *Document type:* Web passage.

Relevance definition: Given a query (web query) and a document (web passage), the document is relevant to the query if the critical concepts or theories discussed in the document can provide references for domain experts to draft an answer to the query.

LeCaRDv2

Query type: Chinese Criminal Case Description. *Document type:* Chinese Criminal Judgment.

Relevance definition: Given a query (a Chinese criminal case description) and a document (a candidate Chinese criminal judgment), the document is relevant if legal experts consider it legally comparable to the query, taking into account the criminal charge and similarities in critical facts, constitutive elements of the offense, sentencing circumstances, procedural controversies, and overall legal relevance.

FiQA

Query type: Personal finance or investment question. *Document type:* Financial answer passage.

Relevance definition: Given a query (personal finance or investment question) and a document (financial answer passage), the document is relevant if it substantially addresses the specific financial scenario, decision, or concept raised by the query and provides information that would materially help answer the user's financial information need.

SCIDOCs

Query type: Scientific paper title. *Document type:* Scholarly article passage.

Relevance definition: Given a query (scientific paper title) and a document (scholarly article passage), the document is relevant if it describes a scientific work or foundational source that the query paper cites or would reasonably cite, including closely related prior work, methods, datasets, benchmark systems, theoretical background, or seminal work that directly supports, motivates, contextualizes, or is compared with the query paper.

SciRepEval Search

Query type: Scientific literature search query. *Document type:* Scientific paper title, abstract, venue, and publication year.

Relevance definition: Given a query (a scientific literature search query) and a document (a scientific paper represented by its title, abstract, venue, and publication year), the document is relevant if it directly satisfies the scientific information need expressed by the query, such as identifying a requested work or addressing the specified research topic, method, phenomenon, dataset, population, intervention, or outcome. Venue and year may help resolve publication-specific constraints, but do not by themselves make an otherwise topically unrelated paper relevant. A document that merely shares broad terminology while studying a different question or failing the query's specific constraints is insufficient.

CQADupStack-Stats

Query type: Statistics Q&A question. *Document type:* Statistics Q&A forum post.

Relevance definition: Given a query (Statistics Q&A question) and a document (Statistics Q&A forum post), the document is relevant if it asks or addresses the same underlying statistical question or problem as the query. It should express an equivalent or near-duplicate information need, even if the wording, notation, dataset, or concrete example differs. Merely mentioning the same statistical method, analytical tool, or broad topic is not sufficient unless the document helps answer the query’s specific concern.

Natural Questions

Query type: Natural-language factoid question. *Document type:* Wikipedia passage.

Relevance definition: Given a query (natural-language factoid question) and a document (Wikipedia passage), the document is relevant if it contains factual evidence that directly answers the question or provides the specific information needed to derive the answer. Merely discussing the same entity or broad topic without resolving the question is insufficient.

CQADupStack-Physics

Query type: Physics Q&A question. *Document type:* Physics Q&A forum post.

Relevance definition: Given a query (Physics Q&A question) and a document (Physics Q&A forum post), the document is relevant if it asks or addresses the same underlying physics question or problem as the query. It should express an equivalent or near-duplicate information need, even if the wording, notation, physical system, or concrete example differs. Merely mentioning the same physical concept, theory, or formula is not sufficient unless the document helps answer the query’s specific concern.

NFCorpus

Query type: Non-technical health or nutrition query. *Document type:* Scientific medical article or biomedical research abstract.

Relevance definition: Given a query (non-technical health or nutrition query) and a document (scientific medical article or biomedical research abstract), the document is relevant if it provides scientific evidence, clinical or epidemiological findings, experimental results, or biomedical explanations that directly address or substantially inform the health, nutrition, disease, or dietary intervention topic raised by the query.

EvidenceBench

Query type: Biomedical scientific hypothesis. *Document type:* Sentence from a biomedical research paper.

Relevance definition: Given a query (a biomedical scientific hypothesis) and a document (a sentence from a biomedical research paper), the document is relevant if it reports a finding, observation, analysis, or conclusion that directly supports, contradicts, qualifies, or otherwise provides evidence about the relationship or outcome asserted by the hypothesis. Merely mentioning the same biomedical entities or broad topic is insufficient.

MedicalQA

Query type: English-language medical question. *Document type:* English-language medical answer passage.

Relevance definition: Given a query (an English-language medical question) and a document (an English-language medical answer passage), the document is relevant if it directly provides the specific medical information needed to answer the question, such as information about a disease, condition, symptom, drug, medical test, procedure, or treatment. Merely mentioning the same medical topic or entity without addressing the specific aspect asked about in the query is insufficient.

MedicalRetrieval

Query type: Chinese-language medical or health question. *Document type:* Chinese-language medical answer passage.

Relevance definition: Given a query (Chinese-language medical or health question) and a document (Chinese-language medical answer passage), the document is relevant if it provides a direct and medically pertinent answer to the question or otherwise helps resolve the health concern expressed in the query.

CodeTrans-DL

Query type: TensorFlow Code Snippet. *Document type:* PaddlePaddle Code Snippet.

Relevance definition: Given a query (TensorFlow Code Snippet) and a document (PaddlePaddle Code Snippet), the document is relevant if it implements the same deep learning functionality as the query using PaddlePaddle APIs and conventions, making the two snippets functionally equivalent cross-framework counterparts.

CosQA

Query type: Natural language web query for code search. *Document type:* Python function implementation with optional documentation/comments.

Relevance definition: Given a query (Natural language web query for code search) and a document (Python function implementation with optional documentation/comments), the document is relevant if the code im-

plementation and its internal documentation (docstrings) provide a direct, functional solution to the specific programming task or logic described in the query.

Text2SQL

Query type: Natural Language Database Query. *Document type:* SQL Query.

Relevance definition: Given a query (Natural Language Database Query) and a document (SQL Query), the document is relevant if it accurately and completely implements the database operation described in the query.

BIRCO-relic

Query type: Literary analysis excerpt with a masked quotation. *Document type:* Candidate passage from a literary work.

Relevance definition: Given a query (an excerpt of literary analysis containing a quotation replaced by '[masked sentence(s)]') and a document (a candidate passage from the analyzed literary work), the document is relevant if it can be naturally inserted at the masked position and directly supports at least one of the literary claims made in the surrounding context. Sharing the same work, characters, themes, or vocabulary without supporting the surrounding interpretation, or merely repeating content already present in the query, is insufficient.

TheoremQA questions

Query type: Theorem-based STEM word problem. *Document type:* Solved STEM problem.

Relevance definition: Given a query (theorem-based STEM word problem) and a document (solved STEM problem), the document is relevant if it uses the same or a closely related theorem, principle, formula, or reasoning pattern required to solve the query, providing a worked example that can guide the solution.

TheoremQA theorems

Query type: Mathematical word problem. *Document type:* Mathematical theorem, definition, or proof.

Relevance definition: Given a query (mathematical word problem) and a document (mathematical theorem, definition, or proof), the document is relevant if it states, defines, or proves a theorem or mathematical principle that is used to derive the solution to the query.

CQADupStack-Mathematica

Query type: Mathematica Q&A question. *Document type:* Mathematica Q&A forum post.

Relevance definition: Given a query (Mathematica Q&A question) and a document (Mathematica Q&A forum post), the document is relevant if it asks or addresses the same underlying Mathematica or Wolfram Language question or problem as the query. It should express an equivalent or near-duplicate information need even if the wording, code, data, or concrete example differs. Merely mentioning the same function, symbol, or broad topic is insufficient unless the document helps resolve the query's specific concern.

Touche2020

Query type: Controversial Argumentative Question. *Document type:* Argumentative passage with reasoning.

Relevance definition: Given a query (Controversial Argumentative Question) and a document (Argumentative passage with reasoning), the document is relevant if it provides a comprehensive, logically structured argument or rebuttal that takes a clear stance on the topic, supporting its position with specific contentions, evidence, or reasoning.

ArguAna

Query type: Argumentative Claim. *Document type:* Counter-argument Passage.

Relevance definition: Given a query (argumentative claim) and a document (counter-argument passage), the document is relevant if it provides a valid, logically structured counter-argument that directly challenges, refutes, or offers a conflicting perspective to the central premise of the query.

DBpedia-entity

Query type: Entity-centric search query. *Document type:* Encyclopedic entity description.

Relevance definition: Given a query (entity-centric search query) and a document (encyclopedic entity description), the document is relevant if it provides a factual summary of a specific entity, instance, or concept that directly satisfies the categorical constraints and attributes specified in the query.

SpartQA

Query type: Textual spatial scene followed by a multiple-choice question. *Document type:* Candidate spatial answer phrase.

Relevance definition: Given a query (a textual spatial scene followed by a multiple-choice question) and a document (a candidate spatial answer phrase), the document is relevant if it denotes an answer supported by the scene's stated spatial relations and valid inferences across objects and containers. If both choices satisfy the queried relation, each choice and 'both of them' are relevant; if neither does, 'none of them' is relevant. Mere object mention or lexical overlap is insufficient.

WinoGrande

Query type: Commonsense fill-in-the-blank sentence. *Document type:* Candidate answer phrase.

Relevance definition: Given a query (a sentence containing one blank marked by an underscore) and a document (a candidate answer phrase), the document is relevant if inserting it into the blank yields the intended completion under the causal, physical, social, or temporal commonsense constraints of the full sentence. Grammatical compatibility or lexical association alone is insufficient.

FiQA case study: complete initial and refined instructions

This section provides the complete instruction pairs and original examples underlying the FiQA case study in main-text Fig. 5. Query-instruction refinement directed generation towards a user’s financial information need, while annotation-instruction refinement clarified that correcting a query’s premise can be relevant. The instruction texts below are reproduced in full from the saved records.

Query-generation example and instructions

The initial and refined instructions correspond to the first and third recorded attempts. The source document and queries below are verbatim production examples generated in separate runs using these instructions. Attribute plans differed between the runs.

Source document (doc_id 310844):

There's no catch. Banks need to acquire customers just like any other business. One common way to acquire new customers is by advertising on the radio, TV, print, etc. Another common way to acquire new customers is by offering incentives like the one you linked to. Basically, PNC is confident that they will make more than \$100 in profit over the entire lifetime of a customer. This is a very reasonable assumption, considering that:

Query generated with the initial instruction:

Why is PNC offering \$100 to new customers if they're confident they'll make more than \$100 in profit over the lifetime of that customer?

Query generated with the refined instruction:

why is pnc giving \$100 just for opening an account

Initial-instruction plan:

```
{"used_identifier": ["PNC"], "used_attributes": {"query focus": "bank incentive strategy", "user perspective": "individual consumer", "answer form": "explanation of rationale"}}
```

Refined-instruction plan:

```
{"used_identifier": ["PNC"], "used_attributes": {"query focus": "bank incentive strategy", "financial decision type": "personal banking choice", "reasoning depth": "basic factual inquiry"}}
```

Complete initial query-generation instruction (attempt 1):

Synthesizer Role

The synthesizer should adopt the role of an individual seeking practical financial clarity, acting as a retail investor, taxpayer, or household budgeter. Their goal is to type concise, direct questions into a search engine or community forum to resolve specific points of confusion, compare financial strategies, or understand the practical implications of a financial rule.

Query Extraction Process

* **Identify the Core Financial Dilemma:** Pinpoint the specific problem, trade-off, or conceptual confusion the passage addresses, such as a choice between two investment vehicles, a misunderstanding of a fee, or the tax consequences of an action.

- * ****Determine the User's Motivation:**** Infer whether the user is looking for a "how-to" guide (reducing expenses), a "what's the difference" comparison (trading vs. flipping), or a "can I do this" permission-based inquiry (using short-sale proceeds).
- * ****Adopt a Direct, Natural Voice:**** Frame the query using everyday financial vocabulary; avoid overly formal or academic language in favor of the terminology a non-expert would likely use.
- * ****Abstract Specifics into Scenarios:**** If the document provides a detailed numerical example or specific case study, generalize it into a broader situational query while retaining the essential financial variables.
- * ****Focus on the Underpinning Intent:**** Ensure the query strikes at the heart of the document's unique insight rather than just matching keywords; if the document debunks a myth or offers a psychological tip, frame the query to elicit that specific guidance.
- * ****Maintain Concision:**** Limit the query to a single sentence or a brief phrase that captures the primary information need, omitting any extraneous details from the passage that the user would not yet know.

Linguistic DNA

- * ****Varied Intent Syntax:**** Queries should range from direct "How-to" questions and "What is" definitions to complex situational scenarios and mechanical "Is it possible" inquiries.
- * ****Pragmatic Tone:**** The tone must be inquisitive and utilitarian, reflecting an individual seeking actionable financial advice or clarification on technical market rules.
- * ****Variable Length:**** Synthesize queries across a spectrum of brevity, from short keyword-based phrases (4-6 words) to detailed multi-part scenarios (15-25 words).
- * ****Domain-Specific Vernacular:**** Use common financial terminology naturally (e.g., "401k," "shorting," "tax implications," "long position," "initial payment").
- * ****Informal Polish:**** Maintain a style that mimics real-world user behavior—grammatically functional but often lacking perfect capitalization or formal academic structure.

Relevance Heuristics

- * ****Scenario-Action Coupling:**** Focus on queries that pair a specific financial situation (e.g., being in your early 20s) with a specific decision point (e.g., 401k investment choices).
- * ****Mechanism Inquiries:**** Emphasize the "how" of financial movement, such as the liquidity of funds between trades or the timing of down payments.
- * ****Comparative Analysis:**** Generate queries that require the document to distinguish between two related but distinct concepts (e.g., day trading vs. flipping).
- * ****Outcome Orientation:**** Ensure the query seeks a material benefit, such as spending less, reducing tax liability, or accelerating a home purchase timeline.
- * ****Constraint-Based Questions:**** Frame queries around limitations or permissions within financial regulations and account types.

Negative Constraints

- * ****No Generic Financial Literacy:**** Avoid overly broad "What is money?" or "What is the stock market?" queries that lack a specific user intent or scenario.
- * ****No Hyper-Dense Jargon:**** Avoid queries that read like textbook headings; they should feel like a human asking for help, not a search engine index.
- * ****No Opinion-Seeking:**** Avoid subjective prompts like "What is the best stock?" or "Do you like Vanguard?"; focus on factual, procedural, or strategic information needs.
- * ****No Excessive Preamble:**** Eliminate "filler" phrases like "I was wondering if you could tell me..." Start directly with the core question or scenario.
- * ****No Unrelated Personal Anecdotes:**** Keep scenarios concise; do not include irrelevant biographical details that do not impact the financial logic of the question.

Complete refined query-generation instruction (attempt 3):

Synthesizer Role
You are a person with a specific money problem searching for an immediate answer on a community forum (like Reddit) or a mobile search bar. You are the "Uninformed Real-World Actor"—a retail investor, a stressed spouse, or a first-time homebuyer. You do not have the document's expert knowledge. You have a "Minimum Viable Inquiry" mindset: you type only the necessary words to find a solution. You are pragmatic, blunt, and often use shorthand. You never act like a student or a researcher; you act like someone whose bank balance is at stake.

Query Synthesis Methodology

- * ****The "Pre-Knowledge" Shift:**** Before writing, identify the technical jargon in the document and consciously forget it. If the document says "Capital Gains," you think "profit tax." If it says "Amortization," you think "monthly payment."
- * ****The Problem-Source Pivot:**** Do not summarize what the document says. Instead, imagine the messy, real-life scenario that forced someone to look for this information. (e.g., Document: "KYC Regulations." User: "why does the bank need my ID?")
- * ****Translate Jargon to Street-Talk:**** Use the most common, non-technical equivalent of words. Replace "Leave Encashment" with "getting paid for unused vacation"; replace "Identity Verification" with "bank asking for my ID."

* ****The Financial Forced-Angle:**** If a document is non-financial (e.g., about psychology or hobbies), you MUST extract or invent a financial motive. Focus exclusively on cost, price, or saving money. (e.g., If the document is about free vs paid content, the query is "how to spend less on subscriptions" or "why pay for it when its free.")

* ****Identify the Core Conflict:**** Search for a "this vs. that" or a "can I do [Action X]" logic. Use the document as a "Yes" or "No" map for a specific user action.

Linguistic DNA & Style

* ****Short & Blunt:**** Keep queries between 5-10 words. No complex clauses.

* ****Search Bar Syntax:**** Use sentence-case or all lowercase. Avoid perfect punctuation.

* ****Contractions & Shorthand:**** Use "401k," "IRS," "crypto," "stock," "tax," "house." Use "whats" instead of "what is."

* ****Zero Preamble:**** Never use "Can you," "I need to know," or "I am looking for." Start directly with the noun or the verb. (e.g., "trading vs flipping taxes" NOT "What is the difference between...")

* ****Register:**** Utilitarian, informal, and impatient.

Failure Mode Contrast (Instructional Examples)

* ****Too Technical (FAIL):**** "What are the tax implications of leave encashment at retirement?"

* ****Authentic (PASS):**** "do i pay tax on unused vacation payout when I quit"

* ****Too Clinical (FAIL):**** "Is there a financial benefit to utilizing free internet resources?"

* ****Authentic (PASS):**** "how can i spend less on this" or "is it worth paying for"

* ****Echoing (FAIL):**** "Know your customer identification requirements for bank accounts."

* ****Authentic (PASS):**** "why the bank needs my id just to open an account"

Negative Constraints

* ****FORBID Specific Nouns:**** Do not use the specific technical titles or headers found in the document. If the title is "Amortization Schedule," you are forbidden from using "Amortization" or "Schedule."

* ****NO Self-Labeling:**** Never say "As a 20 year old..." or "I am a beginner." Incorporate context naturally: "best 401k for early 20s" or "how to start investing."

* ****NO Summarization:**** Do not try to encompass the whole document. Target one specific, actionable question that the document answers.

* ****NO Academic Words:**** Ban words like: "implications," "considerations," "factors," "pertaining," "regarding," "significant," or "process."

Final Quality Filter

Does this query look like it was typed into a phone with one hand while standing in a checkout line?

* Gold Standard Example: "Whats the difference between day trading and flipping and their tax implications?" (Note: "Tax implications" is the only technical term allowed because it is a common search-engine shorthand).

* Gold Standard Example: "Buying a House and Taking Part of 20% to move initial payment date forward"

Relevance-annotation example and instructions

The following known-positive few-shot pair is the original example summarized in Fig. 5b. The initial instruction corresponds to the first recorded attempt; the refined instruction corresponds to the second. The recorded comparison changes from **hard_negative** to **positive**, matching the reference label after refinement.

Original query:

Buying a House and Taking Part of 20% to move initial payment date forward

Original positive document:

In the prior PMI discussions here, it's been stated that the bank is not obligated to remove PMI until the mortgage's natural amortization puts the debt at 78% LTV. So, paying in advance like this will not automatically remove the PMI. Nor will a lump sum payment be certain to move the next payment ahead a year. If it's entered as a principal prepayment, the next month's payment is still due. In the world of coupon books, if you sent in a year's payments, you'd not benefit from the interest saved, in one year you'd owe what the amortization table tells you. There's no free lunch when it comes to mortgages or finance in general. This is why we usually caution that one should not be cash poor the day after buying a house. Best to save 30%, put down 20%, and have a cushion after the closing.

Complete initial relevance-annotation instruction (attempt 1):

You are a financial relevance annotator. For each query-document pair, judge whether the document substantially addresses the specific financial scenario, decision, or concept raised by the query and provides information that would materially help answer the user's financial information need.

****Scoring Rubric (Absolute 0–3)****

- ****Score 3 – Positive****: Directly and fully addresses the query's precise financial scenario, decision, or concept. Contains the critical details, conditions, or actionable guidance the user actually needs. No essential dimension is omitted.
- ****Score 2 – Positive****: Adequately addresses the core need but is less complete, less specific, or less authoritative than a Score 3. It still provides material help—just missing some nuance, depth, or coverage of a secondary aspect. Valid positive, but not the strongest answer.
- ****Score 1 – Hard Negative****: Plausible at a glance (shares keywords, context, or the broad topic) but fails the relevance definition. It does ****not**** materially help answer the query because it either misses the specific financial scenario/decision, addresses a different angle, omits the decisive constraint, or provides information that is tangential, outdated, or inapplicable to the user's situation. Only score 1 when a reasonable reader could initially think it might be relevant, but careful analysis shows it isn't.
- ****Score 0 – Easy Negative****: Clearly unrelated, off-topic, or outside the task's relevance criteria. No plausible connection to the query.

****Hard Negative Reasoning (Score 1)****

To identify hard negatives, ask: "What is the exact financial information need?" and "What makes this document look relevant, yet fail?"

Common hard-negative patterns in this task:

- ****Topic overlap, wrong decision context****: The document discusses the same general instrument or concept (e.g., "Roth IRA"), but the query asks about a specific strategy, tax treatment, or eligibility condition that the document never addresses (e.g., query: "tax implications of a backdoor Roth IRA" \u2192 document explains the basics of a Roth IRA without mentioning backdoor conversions or their tax nuances).
- ****Partial match missing the operative constraint****: The query includes a critical qualifier (e.g., "for self-employed," "in retirement," "for crypto") that materially changes the answer. The document provides generic information without that qualifier, rendering the response inapplicable or misleading.
- ****Wrong temporal or jurisdictional scope****: The document is about an outdated rule, a different country's regulation, or a repealed provision when the query implies current, actionable guidance.
- ****Looks helpful but solves the wrong problem****: The document explains how to perform an action, but the query asks whether the action is advisable in a specific context, or asks for comparison/alternatives the document never covers.

In your reasoning, explicitly state which part of the query's need the document fails to meet, and why a surface-level similarity is misleading.

****Input Format****

You will receive:

- A query string (personal finance or investment question)
- A list of candidate documents, each with a unique `doc\_id`.

****Annotation Protocol****

1. Process documents in the given order. Annotate every document exactly once; no duplication or omission.
2. Evaluate each document independently. Do not compare documents; assign an absolute score based solely on the query-document relationship.
3. Apply the rubric strictly. If in doubt between Score 1 and Score 2, re-read the query's specific scenario/decision and check whether the document's information materially advances an answer.

****Output Format****

Return only a valid JSON object, no additional text:

```
{
  "query": "original query string",
  "annotations": [
    {
      "doc_id": "doc_id",
      "score": 0-3,
      "classification": "positive|hard negative|easy negative",
      "reasoning": "Brief, specific explanation of why the document does or does not satisfy the relevance definition. For hard negatives, pinpoint the plausible surface connection and the missing critical element."
    }
  ],
  "positive_doc_ids": ["list of doc_id with score 2 or 3"],
  "hard_negative_doc_ids": ["list of doc_id with score 1"],
  "easy_negative_doc_ids": ["list of doc_id with score 0"]
}
```

Complete refined relevance-annotation instruction (attempt 2):

You are a financial relevance annotator. For each query-document pair, judge whether the document substantially addresses the specific financial scenario, decision, or concept raised by the query and provides information that would materially help answer the user's financial information need. Material help includes information that conclusively resolves the query—whether by confirming, refuting, or correcting the query's underlying premise—as long as it directly engages with the core financial decision or scenario.

****Scoring Rubric (Absolute 0–3)****

- ****Score 3 – Positive****: Directly and fully addresses the query's precise financial scenario, decision, or concept. Contains the critical details, conditions, or actionable guidance the user actually needs, or provides a definitive resolution (including a clear, authoritative explanation that the proposed strategy is impossible, inadvisable, or based on a misunderstanding). No essential dimension is omitted.
- ****Score 2 – Positive****: Adequately addresses the core need but is less complete, less specific, or less authoritative than a Score 3. It still provides material help—just missing some nuance, depth, or coverage of a secondary aspect. A document that supplies a largely correct answer but lacks full detail, or a refutation that omits some conditions, would fall here. Valid positive, but not the strongest answer.
- ****Score 1 – Hard Negative****: Plausible at a glance (shares keywords, context, or the broad topic) but fails the relevance definition. It does ****not**** materially help answer the query because it either misses the specific financial scenario/decision, addresses a different angle, omits the decisive constraint, or provides information that is tangential, outdated, or inapplicable to the user's situation. Only score 1 when a reasonable reader could initially think it might be relevant, but careful analysis shows it isn't.
- ****Score 0 – Easy Negative****: Clearly unrelated, off-topic, or outside the task's relevance criteria. No plausible connection to the query.

****Hard Negative Reasoning (Score 1)****

To identify hard negatives, ask: "What is the exact financial information need?" and "What makes this document look relevant, yet fail?"

Common hard-negative patterns in this task:

- ****Topic overlap, wrong decision context****: The document discusses the same general instrument or concept (e.g., "Roth IRA"), but the query asks about a specific strategy, tax treatment, or eligibility condition that the document never addresses (e.g., query: "tax implications of a backdoor Roth IRA" \u2192 document explains the basics of a Roth IRA without mentioning backdoor conversions or their tax nuances).
- ****Partial match missing the operative constraint****: The query includes a critical qualifier (e.g., "for self-employed," "in retirement," "for crypto") that materially changes the answer. The document provides generic information without that qualifier, rendering the response inapplicable or misleading.
- ****Wrong temporal or jurisdictional scope****: The document is about an outdated rule, a different country's regulation, or a repealed provision when the query implies current, actionable guidance.
- ****Looks helpful but solves the wrong problem****: The document explains how to perform an action, but the query asks whether the action is advisable in a specific context, or asks for comparison/alternatives the document never covers. ****Caution****: If the document fully engages with the user's scenario and provides a resolution (including a valid reason the action won't work or is inadvisable), it solves the right problem and is relevant (Score 2 or 3). Do not mistake a corrective/dismissive answer for solving the wrong problem.

****Critical distinction—correction vs. mismatch****

A document that conclusively addresses the query's scenario by showing the proposed strategy is not feasible, not advisable, or based on a misunderstanding is ****highly relevant****. It provides exactly the material help the user needs to make a correct financial decision. For example:

- Query: "Can I use a lump-sum prepayment to move my next mortgage payment date forward?"
- Document: "A lump-sum principal payment does not change your next payment due date; it only reduces the total interest and shortens the loan term."

This document directly and definitively answers the user's uncertainty. It is a ****positive**** (Score 3 or 2), not a hard negative, because it engages with the exact scenario and provides a conclusive resolution.

If a document only tangentially mentions the topic or addresses a different aspect without engaging the core question, it may be a hard negative. Always check: does the document speak to the user's stated financial decision or scenario, even if it contradicts the user's initial assumption? If yes, it is relevant.

****Input Format****

You will receive:

- A query string (personal finance or investment question)
- A list of candidate documents, each with a unique `doc\_id`.

****Annotation Protocol****

1. Process documents in the given order. Annotate every document exactly once; no duplication or omission.
2. Evaluate each document independently. Do not compare documents; assign an absolute score based solely on the query-document relationship.

3. Apply the rubric strictly. If in doubt between Score 1 and Score 2, re-read the query's specific scenario/decision and check whether the document's information materially advances an answer (including by definitively showing the proposed action is impossible or inadvisable).

****Output Format****

Return only a valid JSON object, no additional text:

```
{
  "query": "original query string",
  "annotations": [
    {
      "doc_id": "doc_id",
      "score": 0-3,
      "classification": "positive|hard negative|easy negative",
      "reasoning": "Brief, specific explanation of why the document does or does not satisfy the relevance definition. For hard negatives, pinpoint the plausible surface connection and the missing critical element. For positives that refute the query's premise, note that it directly addresses and resolves the core scenario."
    }
  ],
  "positive_doc_ids": ["list of doc_id with score 2 or 3"],
  "hard_negative_doc_ids": ["list of doc_id with score 1"],
  "easy_negative_doc_ids": ["list of doc_id with score 0"]
}
```

Supplementary Note 2: Experimental configurations

Test-mode models. A commonly used test-mode configuration uses GPT-4.1-mini to evaluate identifier-attribute outputs and refine their generation guidance. Gemini 3 Flash Preview generates, evaluates and refines task definitions and query-generation instructions, and constructs and refines relevance-annotation instructions using feedback from programmatic few-shot checks. Qwen3-30B-A3B-Instruct-2507 generates identifiers, attributes, queries and relevance labels in both test and production modes.

Supplementary Table 1 summarizes the shared training settings for the self-improvement comparison, component ablations and MS MARCO query-source comparison. Supplementary Table 2 gives the settings for adaptation of strong embedding models and the MS MARCO data-scaling experiment.

Supplementary Table 1: Shared training settings for the self-improvement comparison, component ablations and MS MARCO query-source comparison.

Parameter	Setting
Initialization	Qwen3-0.6B-auto-eos
Pooling	Last token
Embedding normalization	L2 normalization
LoRA configuration	Rank 32, alpha 64, dropout 0.1
Learning rate	10^{-4}
Global query batch size	96
Training epochs	1
Warmup ratio	0.1
Random seed	42

Supplementary Table 2: Training settings for adaptation of strong embedding models and the MS MARCO data-scaling experiment. The strong-model settings apply to LeCaRDv2, EvidenceBench, SciRepEval Search, TheoremQA questions, SpartQA and WinoGrande.

Parameter	BGE-M3	Qwen3-Embedding-8B	Qwen3-8B-auto-eos (scaling)
Learning rate	10^{-5}	10^{-4}	10^{-4}
Training epochs	1; 5 for SpartQA	1	1
Query batch size per GPU	2; 16 for SpartQA	16	16
Pooling	CLS token	Last token	Last token
Embedding normalization	L2	L2	L2
Maximum query / passage length (tokens)	512 / 512	512 / 512	512 / 512
Warmup ratio	0.1	0.1	0.1
Random seed	42	42	42

For BGE-M3 on SpartQA, we increased the number of training epochs and the batch size because preliminary runs indicated insufficient training under the default settings. Both Qwen3 models in Supplementary Table 2 use LoRA with rank 32, alpha 64 and dropout 0.1. For tasks with existing labelled training data, the original-data and original-plus-SYNTRA conditions use the same training settings. The scaling experiment uses the same training settings and evaluation protocol across all six data scales (10,000, 20,000, 40,000, 80,000, 160,000 and 320,000).

Human-evaluation sampling. For the seven-annotator relevance-label evaluation, we randomly sampled 50 query-document pairs without replacement from the LLM-annotated candidate pairs for each of MS MARCO, Text2SQL and TheoremQA (theorems), yielding 150 pairs in total.