DSL-R1: From SQL to DSL for Training Retrieval Agents across Structured and Unstructured Data with Reinforcement Learning

Anonymous ACL submission

Abstract

Effective retrieval in complex domains requires bridging the gap between structured metadata and unstructured content. Existing systems typically isolate these capabilities, relying on either symbolic filtering or vector similarity, failing to capture their interplay. In this work, we propose DSL-R1, a unified framework that synergizes logical reasoning with semantic matching via a novel Domain-Specific Language (DSL). By embedding vector primitives within SQL-style operators, our approach leverages the complementary strengths of symbolic precision and semantic coverage. We further introduce a reinforcement learning mechanism where rule-based execution feedback and retrieval quality rewards jointly optimize the DSL generation, balancing structural correctness and semantic alignment. Evaluations on a large-scale industrial email benchmark demonstrate that DSL-R1 achieves a +12.3% improvement in Hit@1/3, consistently outperforming decoupled baselines and establishing a robust paradigm for hybrid retrieval.

1 Introduction

The explosive growth of data presents pressing challenges for retrieval systems. Symbolic query languages such as SQL (Zhong et al., 2017) or SPARQL (Berant et al., 2013) provide strong interpretability and precise logical control but struggle with unstructured modalities such as free-form text or document content. Conversely, vector-based models (e.g., CLIP (Radford et al., 2021), BLIP-2 (Li et al., 2023)) excel at semantic matching but lack the ability to enforce logical constraints and offer explainability. Bridging these two paradigms is critical for building retrieval systems that are both robust and interpretable.

To this end, we propose a DSL-driven retrieval framework that integrates SQL operators with vector search under a unified syntax. This design enables compositional queries that capture both logical constraints and semantic intents within a single, language-like interface. A key challenge lies in generating and executing DSL queries reliably under ambiguous or noisy user inputs. We address this by introducing a reinforcement learning framework that jointly optimizes correctness and retrieval utility through rule-based reward functions, enhancing both execution robustness and semantic coverage.

We evaluate our framework on a large-scale retrieval dataset constructed from business emails, which combines structured metadata with unstructured text and attachments. To demonstrate cross-domain robustness, we further validate our approach on the ArxivQA benchmark. Experiments using GRPO (Sun et al., 2024) and DAPO (Wang et al., 2025) optimization algorithms show substantial accuracy gains over symbolic-only and vector-only baselines, with DAPO offering the best trade-off between stability and efficiency.

Our main contributions are summarized as follows: **Large-scale hybrid retrieval benchmarks:** we construct a challenging dataset from business emails integrating structured metadata with unstructured textual and visual content. **A DSL-agent reinforcement learning framework:** a unified DSL combining SQL-style logical operators with vector-based retrieval, optimized through RL signals using LLM-simulated feedback. Experiments demonstrate that our approach significantly outperforms baselines, with DAPO providing the best trade-off between stability and efficiency.

2 Related Work

2.1 Retrieval

Retrieval methods have revolutionized the processing of unstructured data by mapping inputs into a semantic space. Early works such as DPR (Karpukhin et al., 2020) and ANCE (Xiong et al.) demonstrated the effectiveness of dual-encoder architectures for text retrieval.

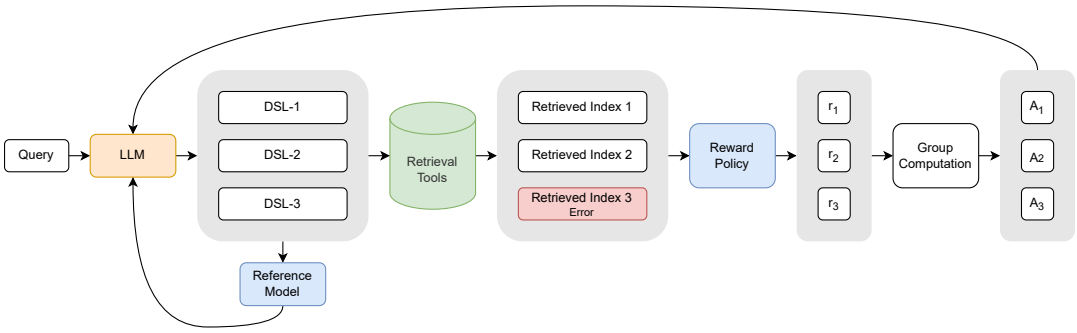

Figure 1: Overview of the reinforcement learning framework for DSL-based retrieval

This paradigm has been successfully extended to multimodal contexts, where models like CLIP (Radford et al., 2021), ALIGN (Jia et al., 2021), and BLIP-2 (Li et al., 2023) capture semantic alignment across text, image, and video modalities. In contrast, symbolic retrieval methods like SQL (Zhong et al., 2017) and SPARQL (Berant et al., 2013) offer structural precision and interpretability but are less effective for unstructured or ambiguous inputs. Recent hybrid methods, including ColPali (Kenton et al., 2024), and ColQwen (Chen et al., 2024), enhance dense retrieval with contextualized representations, while fusion-based systems such as FiD (Izacard and Grave, 2021), and InstructBLIP (Dai et al., 2023) explore reasoning by integrating retrieval and generation. However, most existing works operate at the representation level, lacking an explicit symbolic layer for compositional reasoning. We address this by introducing a trainable DSL that unifies symbolic operators with vector retrieval, optimized via reinforcement learning to ensure both logical correctness and semantic quality.

2.2 DSLs in AI Systems

DSLs have been widely applied in program synthesis (Balog et al., 2017; Nye et al., 2021; Chen et al., 2021), data analysis (Wang et al., 2023; Zhang et al., 2024), and reasoning with language models (Jiang et al., 2023; Wei et al., 2022; Gupta et al., 2023). DSLs enable structured expressivity and verifiable reasoning but traditionally lack adaptability without external optimization or learning signals. Recent advances have explored combining DSLs with reinforcement learning or large language models to improve control and generalization (Xu et al., 2024; Yao et al., 2023; Gou et al., 2024). Our framework continues this direction by treating the DSL not as a static interface but as a learnable policy space, jointly optimized through rule-based rewards and feedback to achieve semantic alignment and robust execution in retrieval contexts.

3 Method

3.1 Overview

Our framework consists of three main *components*: (i) a *DSL-based retrieval agent*, which translates natural-language inputs into executable programs that combine structured operators with vector-based similarity search; (ii) a *data generation component*, which constructs retrieval corpora containing both structured and unstructured elements, and produces paired queries, programs, and gold targets to support supervised and evaluative training; and (iii) a *reinforcement learning mechanism*, which optimizes the agent policy by leveraging execution-derived rewards and preference-based regularization. Together, these components ensure both interpretability and adaptability, enabling effective retrieval across heterogeneous data sources.

3.2 DSL Agent Design

The core of our system is a *DSL-based retrieval agent* that integrates symbolic and vector-based operators within a unified execution framework. Specifically, the DSL program is composed of two parts: (i) an *SQL-like clause* that handles structured conditions over metadata, and (ii) a *vector search query list* that retrieves candidate items from unstructured modalities. Case samples are included in Appendix A.4.

Given a natural-language query x, the agent first generates a DSL program y containing both structured filters and vector queries. The vector search module computes nearest neighbors in the embedding space and returns a set of candidate primary keys $\{id_1, \ldots, id_k\}$. These candidate identifiers are then injected into the SQL execution engine

as constraints, where structured predicates are applied. The final result set is obtained by merging the outcomes of the SQL filter with the candidate list returned by the vector search.

This design allows the agent to exploit the precision of symbolic filtering while leveraging the semantic coverage of vector similarity, ensuring both interpretability and robustness in retrieval tasks.

3.3 Data Preparation

Our corpus $\mathcal{D}$ consists of email records $d = (m, u)$, where m denotes structured metadata (sender, date, category) and u represents unstructured content (subject, body, attachments). For each record, we randomly select 1–3 metadata attributes to form a structured filter F_s, retrieving a candidate set $\mathcal{C}_s = \{d' \in \mathcal{D} \mid m'(a_i) = m(a_i)\}$. A gold record $\hat{d}$ is then sampled from $\mathcal{C}_s$, and several semantic cues from its content u are used to construct a query embedding q. A vector search refines the candidate set to $\mathcal{C}_{su} = \{d' \in \mathcal{C}_s \mid \cos(q, u') \geq \tau\}$, where τ is a similarity threshold. The structured filter F_s and semantic query q are combined into an executable DSL program y that retrieves $\hat{d}$, while a corresponding natural-language query x is generated from the same attributes, forming a supervised triplet $(x, y, \hat{d})$ that couples symbolic constraints with semantic intent and provides supervision. More details are provided in Appendix A.1.

3.4 Reinforcement Learning

As shown in Figure 1, given a natural language query x, the policy model generates multiple DSL candidates $\{y_i\}_{i=1}^{G}$. Each candidate is executed with retrieval tools; valid results are scored by the reward function, and errors receive zero or negative scores. The group of rewards $\{r_i\}$ is then aggregated to compute group-relative advantages $\{A_i\}$, which guide policy updates.

We adopt two algorithms: GRPO (Sun et al., 2024) and its improved variant DAPO (Wang et al., 2025). GRPO enables efficient policy optimization without a value model, while DAPO further stabilizes training through dual-anchor regularization and adaptive reward scaling.

The simplified GRPO objective is:

$$\mathcal{L}_{\text{GRPO}}(\theta) = -\frac{1}{G}\sum_{i=1}^{G} \min\big(r_i A_i,\ \text{clip}(r_i) A_i\big) + \beta D_{\text{KL}}(\pi_\theta \,\|\, \pi_{\text{ref}}), \quad (1)$$

where $r_i = \frac{\pi_\theta(y_i|x)}{\pi_{\text{old}}(y_i|x)}$ is the importance ratio, π_θ denotes the current policy parameterized by θ, π_{old} is the behavior policy used to generate the sampled trajectories, π_{ref} is a fixed reference policy used to regularize the optimization via the KL term, and A_i is the group-relative advantage.

DAPO further improves GRPO by normalizing advantages at the token level and using asymmetric clipping bounds. Its simplified loss is:

$$\mathcal{L}_{\text{DAPO}}(\theta) = -2\frac{1}{N}\sum_{i=1}^{G}\sum_{t=1}^{|y_i|} \min\big(r_{i,t}A_{i,t}, \text{clip}(r_{i,t}, 1-\epsilon_l, 1+\epsilon_h)\, A_{i,t}\big), \quad (2)$$

where $N = \sum_i |y_i|$ is the total number of tokens, $r_{i,t} = \frac{\pi_\theta(y_{i,t}|x, y_{i,<t})}{\pi_{\text{old}}(y_{i,t}|x, y_{i,<t})}$ is the token-level importance ratio, $A_{i,t}$ is the token-level advantage, and (ϵ_l, ϵ_h) are asymmetric clipping bounds that allow different tolerances for underestimation and overestimation.

These two strategies allow stable and efficient optimization of the DSL agent, with DAPO showing better convergence in practice.

3.4.1 Reward Function Design

To optimize the DSL-based retrieval agent, we design a composite reward function that balances structural correctness, execution reliability, semantic accuracy, and response efficiency. The total reward is formulated as

$$R = S_f + S_e + S_r + S_l,$$

where each component provides distinct feedback to guide policy optimization. **Format Reward** (S_f) encourages syntactic compliance by rewarding outputs that correctly encapsulate reasoning and retrieval logic within predefined DSL tags (e.g., `<query>...</query>`). **Execution Reward** (S_e) measures the executability of the generated DSL statement, assigning higher rewards to programs that can be successfully parsed and executed by the retrieval backend. **Result Reward** (S_r) evaluates the alignment between retrieved results and the reference set, computed as a function of retrieval precision and recall. **Length Reward** (S_l) penalizes overly long outputs to promote concise and latency-efficient responses, ensuring the agent maintains high utility under real-world inference.

4 Experiments

4.1 Setup

Datasets. We evaluate on our industrial Email dataset and ArxivQA (Faysse et al., 2024) benchmarks. Both tasks require handling complex interactions between structured and unstructured data.

Table 1: Retrieval performance and latency (ms) comparison on Email and ArxivQA datasets

Category	Model	Email Dataset				ArxivQA	
		Hit@1	Hit@3	MRR	Lat.	NDCG@5	Lat.
Traditional Retrieval	BM25	42.1	64.7	53.2	18	31.6	25
	ColQwen	55.3	78.9	65.4	35	73.9	42
LLM Based	Qwen3-4B	58.3	80.1	69.2	95	55.4	105
	Qwen3-8B	62.1	83.6	72.5	140	61.2	155
	GPT-4o	68.9	88.7	78.8	**30**	70.5	**35**
DSL-R1	Qwen3-4B + GRPO	69.5	88.9	78.6	55	74.8	62
	Qwen3-4B + DAPO	71.7	90.6	80.9	52	75.1	58
	Qwen3-8B + GRPO	75.8	92.9	84.6	82	77.4	90
	Qwen3-8B + DAPO	**77.9***	**94.2***	**86.1**	78	**81.2***	85

Each query pairs a user request with heterogeneous candidates. Please refer to the Appendix A.1 for specific data formats.

Baselines. We compare four groups: (1) *Traditional retrieval models* such as BM25 (Robertson et al., 2009) and ColQwen (Chen et al., 2024), (2) *LLM baselines* (Qwen3-4B, Qwen3-8B, GPT-4o (OpenAI, 2024)) without RL optimization, and (4) our *DSL-R1 agents* trained with GRPO (Sun et al., 2024) and DAPO (Wang et al., 2025).

Metrics. We report standard retrieval metrics: Hit@k, Mean Reciprocal Rank (MRR), and nDCG, which measure the ranking quality of generated DSL queries in our setting. Latency (ms/query) is also included to reflect runtime efficiency. Formal definitions and equations for all metrics are provided in Appendix A.2.

4.2 Main Results

Table 1 summarizes the overall results. Our DSL-R1 agents outperform all baselines across all metrics, confirming the effectiveness of reinforcement learning in aligning the DSL generator with downstream execution quality. Compared with pretrained LLMs, RL optimization improves Hit@1 by 9.6–15.8 points and MRR by up to 13.6 points. Among the two optimization strategies, DAPO consistently yields higher accuracy and more stable convergence than GRPO on both the 4B and 8B backbones. In addition, reinforcement learning substantially reduces response latency by regularizing output length: *Qwen3-8B + DAPO* achieves 78 ms/query, a 45% reduction compared with the vanilla pretrained model.

Table 2: Ablation study of reward components

Reward Function	Accuracy (%)
Qwen3-8B (pretrained)	62.1
$S_f + S_e + S_r + S_l$	**77.9**
- w/o S_f (Format Reward)	75.3 ($\downarrow$2.6)
- w/o S_e (Execution Reward)	75.6 ($\downarrow$2.3)
- w/o S_r (Result Reward)	77.0 ($\downarrow$0.9)
- w/o S_l (Length Reward)	75.9 ($\downarrow$2.0)

4.3 Ablation Studies

Table 2 quantifies the contribution of each reward component on our Email dataset. Removing any term reduces accuracy, confirming their complementary effects. Format and execution rewards (S_f, S_e) are essential for generating valid and executable DSLs, while the result reward (S_r) enhances semantic correctness and the length reward (S_l) controls verbosity for faster responses. These rewards yield a 15.8-point improvement over the pretrained baseline.

Our ablation analysis reveals consistent trends: removing the SQL filter lowers precision by losing attribute constraints, excluding the vector retrieval module hurts recall on paraphrased queries, and omitting the feedback signal slows convergence and weakens cross-domain generalization.

5 Conclusion

We introduced DSL-R1, unifying SQL logic and vector retrieval in a single DSL, optimized via reward-guided RL. To facilitate rigorous evaluation, we constructed a benchmark encompassing both structured and unstructured retrieval tasks. Across these scenarios, DSL-R1 demonstrates superior accuracy and robustness.

6 Limitations

While our DSL-R1 framework demonstrates strong retrieval performance and generality, several limitations remain. First, the current system operates in a single-turn setting and does not yet support multi-turn query refinement or conversational retrieval, which would require temporal context modeling and dialogue-state tracking. Second, the framework currently adopts a single-agent reinforcement process; extending it to a multi-agent or collaborative setting, such as coordinated query generation, execution, and verification, could further enhance interpretability and robustness. Third, although our DSL syntax unifies symbolic and vector-based retrieval, the present experiments are limited to textual and structured inputs. Extending the framework to handle inputs including images, audio, and videos, as well as cross-modal reasoning, remains an open direction. Finally, our rule-based reward design simplifies supervision but may constrain adaptability when scaling to more complex and heterogeneous real-world datasets.

References

Matej Balog and 1 others. 2017. Deepcoder: Learning to write programs. In *ICLR*.

Jonathan Berant and 1 others. 2013. Semantic parsing on freebase from question-answer pairs. In *EMNLP*.

Mark Chen and 1 others. 2021. Program of thoughts prompting: Debugging reasoning in language models step-by-step. In *NeurIPS*.

Zhenyu Chen and 1 others. 2024. Colqwen: Column-aware dense retrieval via multimodal embedding alignment. *arXiv preprint arXiv:2405.08212*.

Wenliang Dai and 1 others. 2023. Instructblip: Towards general-purpose vision-language models with instruction tuning. In *NeurIPS*.

Manuel Faysse, Hugues Sibille, Tony Wu, Bilel Omrani, Gautier Viaud, Céline Hudelot, and Pierre Colombo. 2024. Colpali: Efficient document retrieval with vision language models. *Preprint*, arXiv:2407.01449.

Jianyu Gou and 1 others. 2024. Llmcompiler: Converting natural language to optimized code via reinforcement learning. *arXiv preprint arXiv:2402.09231*.

Ananya Gupta and 1 others. 2023. Programmatic reasoning with large language models. *arXiv preprint arXiv:2310.06136*.

Gautier Izacard and Edouard Grave. 2021. Leveraging passage retrieval with generative models for open domain question answering. In *EACL*.

Chao Jia and 1 others. 2021. Scaling up visual and vision-language representation learning with noisy text supervision. In *ICML*.

Albert Jiang and 1 others. 2023. Program-aided language models for reasoning and verification. In *ICLR*.

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen Tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020*, pages 6769–6781. Association for Computational Linguistics (ACL).

Jacob Kenton and 1 others. 2024. Colpali: Contextual late interaction for multimodal dense retrieval. *arXiv preprint arXiv:2403.12345*.

Junnan Li and 1 others. 2023. Blip-2: Bootstrapped language-image pre-training with frozen image encoders and large language models. In *ICML*.

Lei Li, Yuqi Wang, Runxin Xu, Peiyi Wang, Xiachong Feng, Lingpeng Kong, and Qi Liu. 2024. Multimodal ArXiv: A dataset for improving scientific comprehension of large vision-language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14369–14387, Bangkok, Thailand. Association for Computational Linguistics.

Maxwell Nye and 1 others. 2021. Show your work: Scratchpads for intermediate computation with language models. In *ICLR*.

OpenAI. 2024. Gpt-4o: Multimodal large language model. *Technical Report*.

Alec Radford and 1 others. 2021. Learning transferable visual models from natural language supervision. In *ICML*.

Stephen Robertson, Hugo Zaragoza, and 1 others. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2024. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*.

Yi Sun and 1 others. 2024. Group relative policy optimization for large language model alignment. *arXiv preprint arXiv:2402.10821*.

Liang Wang and 1 others. 2023. Dsl-sql: A unified domain-specific language for compositional data analytics. In *SIGMOD*.

Liang Wang and 1 others. 2025. Dapo: Dual-anchor policy optimization for efficient llm fine-tuning. *arXiv preprint arXiv:2503.14476*.

Jason Wei and 1 others. 2022. Chain of thought prompting elicits reasoning in large language models. In *NeurIPS*.

Shirley Wu, Shiyu Zhao, Michihiro Yasunaga, Kexin Huang, Kaidi Cao, Qian Huang, Vassilis N. Ioannidis, Karthik Subbian, James Zou, and Jure Leskovec. 2024. Stark: Benchmarking llm retrieval on textual and relational knowledge bases. In *NeurIPS Datasets and Benchmarks Track*.

Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N Bennett, Junaid Ahmed, and Arnold Overwijk. Approximate nearest neighbor negative contrastive learning for dense text retrieval. In *International Conference on Learning Representations*.

Yutong Xu and 1 others. 2024. Rl-code: Reinforcement learning for code generation and optimization. *arXiv preprint arXiv:2403.08711*.

Shunyu Yao and 1 others. 2023. React: Synergizing reasoning and acting in language models. In *ICLR*.

Wei Zhang and 1 others. 2024. Datadsl: A domain-specific language for declarative data transformation. *arXiv preprint arXiv:2401.10212*.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. In *ACL*.

A

A.1 Datasets

We evaluate the alignment of structured metadata with unstructured content across two distinct domains. For each scenario, we utilize 2,000 samples for training and a separate 500 samples for evaluation.

Industrial Email. This in-house dataset represents enterprise workflows with heterogeneous attachments (e.g., invoices, images). The detailed keyword taxonomy and schema definitions are presented in Table 4. We adopted the STaRK framework (Wu et al., 2024) to generate high-quality retrieval instances. For each query, we first select a gold document, then construct a challenging candidate pool by sampling negatives based on structured metadata overlap (Stage 1) and unstructured content similarity (Stage 2). Finally, composite queries are synthesized using the gold document's attributes. We utilize 2,000 samples for training and a distinct 500 samples for evaluation. As shown in Table 3, our dataset features complex thread structures and a balanced distribution of structure-dominated versus content-dominated queries.

Table 3: Statistical profile of the Industrial Email dataset

Metric	Train	Eval	Total
CORPUS TOPOLOGY			
Corpus Size (N)	2,000	500	2,500
Avg. Doc. Length (tokens)	154.2	156.8	154.7
QUERY CONSTRAINT STATISTICS			
Avg. Keyword Counts (k)			
Structured Keys (k_{str})	1.8	2.1	2.0
Unstructured Keys (k_{uns})	3.1	3.0	3.1
Total Complexity (k_{total})	4.9	5.1	5.0
Modality Distribution (%)			
Structure-Dominated ($k_{str} > k_{uns}$)	30.0	28.5	29.7
Content-Dominated ($k_{uns} > k_{str}$)	45.0	46.5	45.3
Balanced ($k_{str} \approx k_{uns}$)	25.0	25.0	25.0

Scientific Papers. This domain focuses on retrieving information from visually rich documents characterized by complex layouts containing embedded images, tables, and formulas, enriched with structured metadata such as domain, category, author, and publish time. For evaluation, we strictly employ the ArXivQA subset from the ViDoRe benchmark (Faysse et al., 2024) to ensure comparability. Conversely, training utilizes the distinct original ArXivQA dataset (Li et al., 2024), ensuring the model learns generalizable alignment features without overfitting to the benchmark distribution.

Table 4: Definition of keywords in the Industrial Email dataset, categorized by information type

Category	Keyword	Description
Structured Info	`account_email`	The email address of the account owner.
	`received_date`	Timestamp when the email was received.
	`is_draft`	Boolean flag indicating if the email is a draft.
	`draft_created_date`	Timestamp for draft creation.
	`draft_modified_date`	Timestamp for last draft modification.
	`is_read`	Boolean status indicating if the email has been read.
	`is_starred`	Boolean status indicating if the email is starred.
	`is_archived`	Boolean status indicating if the email is archived.
	`thread_msg_count`	Number of messages in the current thread.
Unstructured Info	`sender_email`	Email address of the sender.
	`sender_name`	Display name of the sender.
	`recipient_list`	List of primary recipient email addresses.
	`cc_list`	List of carbon copy (CC) recipients.
	`bcc_list`	List of blind carbon copy (BCC) recipients.
	`folder_labels`	Labels or folders associated with the email.
	`attachment_list`	Metadata of files attached to the email.
	`subject`	The subject line text of the email.
	`content`	The main body text of the email message.

A.2 Metrics

We evaluate system performance from both retrieval accuracy and program executability. Let $\mathcal{Q}$ denote the set of evaluation queries, and for each query $q \in \mathcal{Q}$, let r_q be the rank position of the first relevant (ground-truth) item in the retrieved list.

Hit@k. Hit@k measures whether the correct item appears within the top-k retrieved results:

$$\text{Hit@}k = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} \mathbf{1}[r_q \leq k], \tag{3}$$

where $\mathbf{1}[\cdot]$ is the indicator function.

MRR. MRR evaluates ranking quality by assigning higher scores to correct items appearing earlier in the ranking:

$$\text{MRR} = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} \frac{1}{r_q}, \tag{4}$$

nDCG@5. nDCG@5 accounts for graded relevance and penalizes lower-ranked correct items using logarithmic discounting. The discounted cumulative gain at cutoff $K = 5$ is defined as:

$$\text{DCG@5} = \sum_{i=1}^{5} \frac{2^{\text{rel}_i} - 1}{\log_2(i+1)}, \tag{5}$$

where rel_i denotes the relevance score of the item at rank i. The normalized score is obtained by dividing by the ideal DCG:

$$\text{nDCG@5} = \frac{\text{DCG@5}}{\text{IDCG@5}}, \tag{6}$$

Latency. We additionally report average latency, measured in milliseconds per query:

$$\text{Latency} = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} t_q, \tag{7}$$

where t_q is the wall-clock execution time for query q, including both retrieval and DSL generation.

A.3 Implementation Details.

We implemented our training pipeline utilizing the `verl` framework (Sheng et al., 2024), adopting the ColQwen2.5 (Chen et al., 2024) model as our RAG embedding backbone. Fine-tuning was conducted on a cluster of 4 NVIDIA H100 GPUs for 10 epochs with a learning rate of 1×10^{-6}. All inference benchmarks and latency measurements were performed on a single NVIDIA A100 GPU.

A.4 Query-DSL Cases

The following examples demonstrate the mapping between natural language user queries and our Domain Specific Language (DSL). Each case illustrates how a user's intent is parsed into a hybrid structure consisting of an SQL statement for relational filtering and a vector query list for semantic retrieval.

Case 1: Budget Drafts

Query: "Find starred draft emails modified in the last 7 days containing 'budget' in the body."

DSL Output:

```
{
  "sql": "SELECT DISTINCT message_id FROM email WHERE
      is_draft = 1 AND draft_modified_date >=
      date('now', '-7 day') AND is_starred = 1 AND
      message_id IN <vector_0>",
  "vector_query_list": [
    {"email_content": "budget"}
  ]
}
```

Case 2: Important Folders

Query: "Find folders where the label name contains 'important'."

DSL Output:

```
{
  "sql": "SELECT DISTINCT
      json_extract(json_each.value, '$.id') AS
      folder_id FROM email,
      json_each(email.folder_labels) WHERE
      json_extract(json_each.value, '$.id') IN
      <vector_0>",
  "vector_query_list": [
    {"folder_labels": "important"}
  ]
}
```

Case 3: Contract Attachments

Query: "Find attachment IDs for emails where the subject contains 'contract'."

DSL Output:

```
{
  "sql": "SELECT DISTINCT
      json_extract(json_each.value, '$.id') AS
      attachment_id FROM email,
      json_each(email.attachment_list) WHERE
      message_id IN <vector_0> AND
      json_extract(json_each.value, '$.id') IN
      <vector_1>",
  "vector_query_list": [
    {"subject": "contract"},
    {"attachment_list": "contract"}
  ]
}
```