PROGRAM: Programmatic Retrieval Optimization with Generative Reasoning and Augmented Multi-queries

Gun Il Kim[1], Jungkyu Shin[1], Jong Wook Kim*[2], Beakcheol Jang*[1]

[1]Graduate School of Information, Yonsei University, Seoul, South Korea
[2]Department of Computer Science, Sangmyung University, Seoul, South Korea
{kim_gunil_94, jkshin0903, bjang}@yonsei.ac.kr
jkim@smu.ac.kr

Abstract

Current retrieval-augmented generation (RAG) methods struggle with complex multi-hop reasoning, relying on unstructured semantic matching that lacks the logical structure needed to systematically guide retrieval. We introduce Programmatic Retrieval Optimization with Generative Reasoning and Augmented Multi-queries (PROGRAM), a novel framework that elevates retrieval to structured, program-guided reasoning. PROGRAM treats retrieval as execution of specific program types, such as logical, temporal, causal, and so forth, through three stages of 'Program-Type Selection' with dual-metric optimization, 'Iterative Active Program Pruning' with evidence accumulation, and 'Final Answer Generation' with reranking. Evaluated on five benchmarks including HotPotQA, 2WikiMultihopQA, ARC-Challenge, MMLU-Pro, and MedQA with various LLMs, PROGRAM achieves state-of-the-art performance with up to 24% relative improvement on HotPotQA and 13.2% on MedQA over strong baselines including FLARE, ProbTree and Self-RAG. We release our code in our https://github.com/lurker18/PROGRAM

1 Introduction

Retrieval-augmented generation (RAG) has emerged as a foundational paradigm for addressing knowledge-intensive question answering (QA) and complex reasoning tasks, where large language models (LLMs) alone often struggle with temporal knowledge and domain-specific expertise (Lewis et al., 2020; Gao et al., 2023a). By dynamically retrieving relevant information from external knowledge bases and integrating it into the generation process, RAG systems have demonstrated substantial improvements across diverse benchmarks (Yang et al., 2018; Ho et al., 2020; Wang et al., 2024b; Jin et al., 2021). Recent advances in dense retrieval methods (Zhan et al., 2021) and query augmentation strategies (Wang et al., 2023; Nogueira et al., 2019) have further enhanced the retrieval quality, enabling more accurate and contextually grounded responses. However, as the complexity of questions increases, the limitations of conventional retrieval approaches become increasingly apparent.

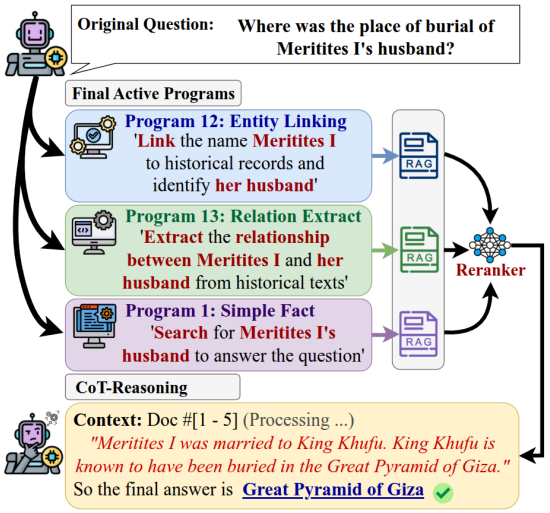

Figure 1: An illustration of programmatic retrieval optimization with generative reasoning and augmented multi-queries from 2WikiMultihopQA with Mistral.

Despite these advances, current retrieval strategies, particularly those employed in Active RAG methods such as FLARE (Jiang et al., 2023b) and Self-RAG (Asai et al., 2024), remain fundamentally limited by their reliance on unstructured, token-level confidence scores and generic semantic matching mechanisms. While FLARE introduces iterative retrieval triggered by low-confidence tokens and Self-RAG incorporates reflection tokens for self-critique, both approaches lack a high-level logical structure to systematically guide *what* and *how* information should be retrieved. Specifically, these methods treat retrieval as a homogeneous similarity search process, failing to recognize that

* Corresponding authors

different question types such as causal comparison, temporal reasoning, and argmax/argmin queries, demand fundamentally distinct retrieval strategies. For example, a question requiring 'causal comparison' necessitates retrieving evidence about multiple cause-effect relationships and contrasting them, whereas a 'symptom analysis' question requires the diagnosis of the medical problem along with its treatment plan. The gap between unstructured semantic matching and the structured logical requirements of complex questions represents a critical bottleneck in advancing retrieval-augmented reasoning systems.

To address this gap, we introduce **P**rogrammatic **R**etrieval **O**ptimization with **G**enerative **R**easoning and **A**ugmented **M**ulti-queries (**PROGRAM**), a novel framework that elevates retrieval from unstructured matching to structured, program-guided reasoning. Figure 1 provides the illustration of PROGRAM. Inspired by recent work on programmatic reasoning (Chen et al., 2023; Gao et al., 2023b), PROGRAM treats retrieval not as a generic similarity search, but as the execution of specific program types such as 'logical', 'temporal', 'causal', etc. Our framework operates through three main stages of 'Program-Type Selection' with dual-metric optimization, 'Iterative Active Program Pruning' with evidence accumulation, and 'Final Answer Generation' with reranking. By structuring retrieval as programmatic operations, PROGRAM ensures that retrieved context is not only semantically relevant but also structurally coherent with the underlying reasoning demands of knowledge-intensive QA tasks.

Our main contributions are as follows:

- We propose PROGRAM, a framework that elevates retrieval from unstructured matching to structured, program-guided reasoning, bridging the gap between generic semantic search and logical query requirements.
- We introduce a dual-metric combining information gain and semantic score for candidate subquery selection and a statistical t-test pruning mechanism to ensure only high-utility reasoning paths are pursued to improve its retrieval quality.
- Our proposed framework achieves the state-of-the-art performance on diverse benchmark datasets across multiple LLM backbones consistently outperforming baselines particularly in tasks requiring complex reasoning.
- Our comprehensive ablation studies further validate the effectiveness of each component of the PROGRAM framework, demonstrating the critical role of programmatic structure in advancing RAG for knowledge-intensive reasoning tasks.

2 Related Work

Adaptive and Iterative Retrieval Strategies Standard RAG models often retrieve information once before generation, which is insufficient for complex queries (Lewis et al., 2020). Recent advancements have shifted toward iterative and adaptive retrieval paradigms. FLARE (Jiang et al., 2023b) introduced an active strategy that triggers retrieval only when the generation probability of low-confidence tokens falls below a threshold. Building on this, Self-RAG (Asai et al., 2024) trains a critic model to generate "reflection tokens" that evaluate the relevance and utility of retrieved passages.

More recently, research has focused on adapting the retrieval strategy to the complexity of the query. Adaptive-RAG (Jeong et al., 2024) utilizes a complexity classifier to dynamically select between iterative, single-step, or no-retrieval strategies, optimizing efficiency without sacrificing accuracy. Similarly, Unified Active Retrieval (UAR) (Cheng et al., 2024) introduces a standardized criteria set for determining when to retrieve, addressing diverse user instructions. While these methods introduce dynamism, they largely rely on token-level confidence or generic complexity classifiers. In contrast, our proposed PROGRAM utilizes a more rigorous statistical support score mechanism to iteratively prune ineffective retrieval paths, ensuring that only statistically significant evidence accumulates.

Structured Reasoning and Query Decomposition To handle complex reasoning tasks, methods like Chain-of-Thought (CoT) (Wei et al., 2022) prompting encourage models to articulate intermediate steps. However, CoT prompting alone often struggles with faithful evidence grounding. Least-to-Most Prompting (Zhou et al., 2023) and Tree-of-Thoughts (ToT) (Yao et al., 2023) advanced this by decomposing problems into subquestions or exploring reasoning paths as a search tree. TRACE (Fang et al., 2024) further extends this by constructing "knowledge-grounded reasoning chains", where each link in the chain is explicitly connected to a knowledge triple.

Despite their success, these decomposition methods often treat all reasoning steps as equal. Our proposed PROGRAM differs by categorizing rea-

soning steps into explicit program-types, allowing for distinct optimization strategies tailored to the specific logical nature of each sub-query.

Programmatic and Tool-Augmented Generation A stream of research seeks to decouple reasoning from computation by binding LLMs to executable programs. Program-of-Thoughts (PoT) (Chen et al., 2023) delegates computation steps to an external Python interpreter to avoid calculation errors in numerical reasoning. Similarly, DSPy (Khattab et al., 2024) treats language model interactions as declarative modules that can be compiled and optimized into pipelines. StructRAG (Li et al., 2025) also attempts to structure scattered knowledge into organized formats before generation.

Our proposed PROGRAM synthesizes the benefits of these programmatic approaches with iterative retrieval. Unlike PoT, PROGRAM applies "programmatic thinking" to the retrieval process itself. By defining retrieval as a set of logical operations, we ensure that the retrieved context is structurally aligned with the reasoning requirements of the original question, bridging the gap between unstructured RAG and structured logic.

3 Methods

We propose PROGRAM framework designed to optimize query formulation through program-based reasoning. Our goal is to identify the optimal program-type query that best retrieves documents relevant to the original query Q. As illustrated in Figure 2, the framework operates in three distinct stages of (1) Program-Type Selection (PTS) with dual-metric optimization, (2) Iterative Active Program Pruning (IAPP) with evidence accumulation, and (3) Final Answer Generation (FAG) with reranking. Detailed algorithm is in Appendix B.

3.1 Program-Type Selection

Given an input question Q and a Knowledge Base K, the framework first interprets the query through diverse logical perspectives by selecting the most applicable reasoning structures.

Instead of relying on a static set of reasoning paths, we instruct the LLM $\mathcal{M}$ to select as many relevant program types as needed from a predefined library. As summarized in Table 1, this library is organized into two groups, 'Basics' and 'Advanced', to accommodate different levels of query complexity. The Basics category covers fundamental reasoning operations, while the Advanced category addresses more complex, specialized, and domain-specific reasoning requirements.

Table 1: Program-types categorized into 'Basic' and 'Advanced'.

Category	Program-Types
Basic	Simple Fact / ArgMax / ArgMin Logical (AND) / Logical (OR) / Causal Comparison / Temporal / Count / Filter
Advanced	Multi-hop / Entity-Linking / Relation Extraction Bridge Entity / Diagnosis / Symptom Analysis Treatment Reasoning / Medical Condition Domain-Specific / Multi-Task / Causal Chain Scientific Reasoning / Knowledge Integration Analogical Reasoning / Hypothesis Testing

Once the relevant set of program types $\mathcal{P}_{selected}$ is identified, the LLM generates multiple candidate sub-queries $C_i = \{c_1, ..., c_M\}$ for each program $p_i \in \mathcal{P}_{selected}$. To ensure that the sub-query chosen for retrieval is high-quality, we evaluate each candidate using two metrics of Information Gain (S_{IG}) and Semantic Score (S_{sem}) respectively. Information Gain measures the utility of a candidate query c_j using Kullback-Leibler (KL) Divergence (Kullback and Leibler, 1951) to quantify the "new" information it contributes to ($P(E|c_j, p_i)$) relative to the program type alone ($P(E|p_i)$), where E denotes evidence. Semantic Score (Zhang et al., 2020) measures the semantic alignment of the candidate query c_j to the original question Q to ensure the optimization does not drift from the user's intent. These metrics are summed to compute a final candidate score:

$$c^* = \text{argmax}_{c_j \in C_i}(S_{IG}(c_j) + S_{sem}(Q, c_j)) \quad (1)$$

The candidate c^* that maximizes the score is finalized as the optimal sub-query, as it best balances semantic relevance to the original question Q with the ability to retrieve richer contextual information under program type p_i.

3.2 Iterative Active Program Pruning

With the optimal sub-queries established for each program type p_i, we enter an iterative retrieval and refinement loop. This stage aims to filter out ineffective reasoning paths while deepening the search for promising ones.

For every finalized sub-query q_i, we retrieve an initial set of documents D_i^{accum} from K. We establish a baseline for relevance by calculating the 'Support Score' (S_{sup}), which measures the semantic similarity between the representative sub-query q_i and its retrieved documents D_i^{accum} using a pretrained language model (e.g., MiniLM-L6-v2 (Wang et al., 2020), MPNet (Song et al.,

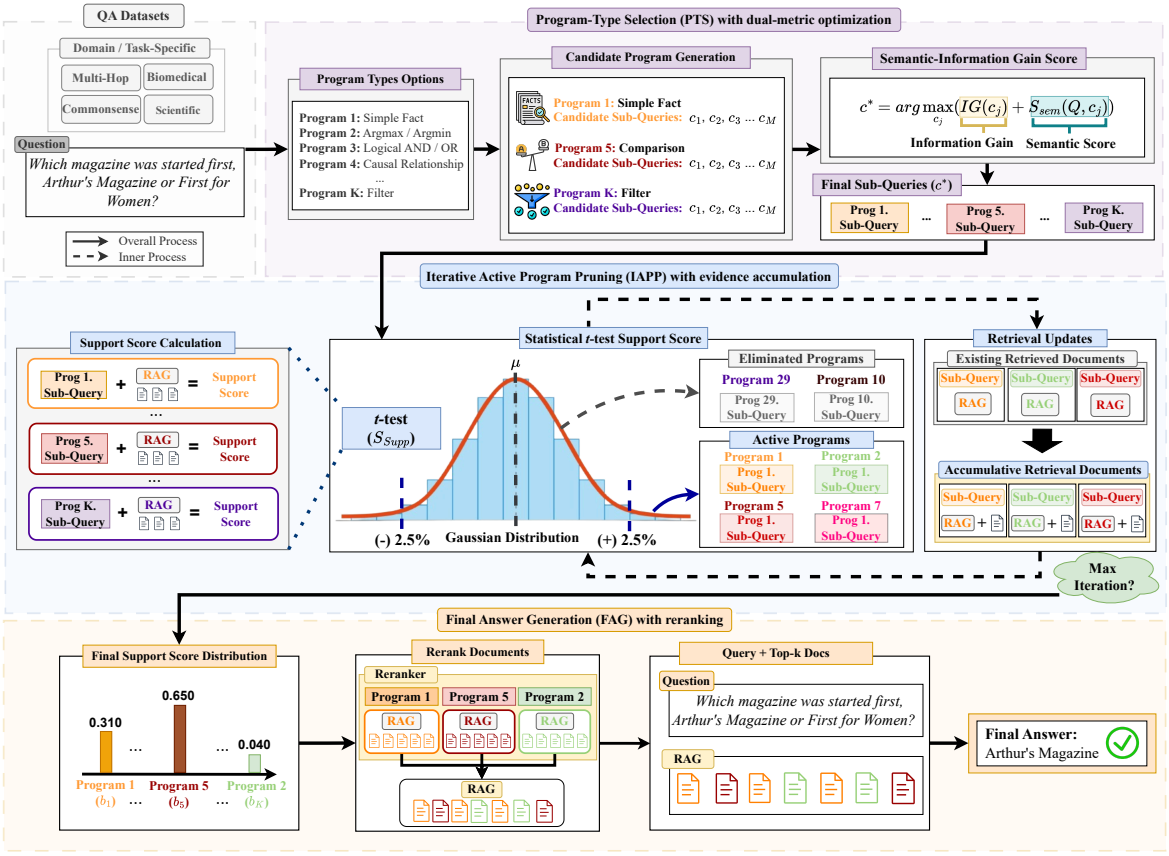

Figure 2: Overall process of the proposed PROGRAM framework.

2020)). The following equation computes the 'Support Score' (S_{sup}):

$$S_{sup}(p_i) = \text{Sim}(q_i, \mathcal{D}_i^{accum}) \quad (2)$$

The refinement process iterates through $t = 1$ to T steps. We analyze the distribution of Support Scores across all current programs. We employ a statistical t-test to determine if a program's retrieval quality is statistically significant compared to the group mean. Programs that fail to meet the minimum threshold are eliminated. The programs that survive this pruning are designated as 'Active Programs'. Functionally, an Active Program represents a reasoning path that has been empirically validated by the retrieved evidence, distinguishing effective retrieval strategies from those that merely introduce noise. This distinction ensures that the framework adapts its logic to the actual data distribution rather than relying on static, potentially irrelevant reasoning structures.

For each Active Program, the LLM evaluates the accumulated documents to determine if sufficient information has been retrieved to answer the original question Q. If the LLM determines that more information is needed, more documents for each Active Program based on each optimal sub-query are retrieved and accumulated. In contrary, if the model deems the evidence sufficient, the program is 'locked'. It is preserved in the active set but requires no further retrieval operations. The procedure iterates until either the maximum number of iterations (T) is reached, or only a single Active Program remains.

3.3 Final Answer Generation

Upon completing the iteration stage, we aggregate the evidence to synthesize the final answer A. We collect all accumulated retrieved documents from the remaining Active Programs including both running and 'locked' programs to form a final evidence pool, D_{final}. To eliminate noise and redundancy, we employ a reranker model to reorder D_{final} based on relevance to the original question Q. We select the top-k most relevant documents from this reranked list.

$$\mathcal{D}_{reranked} = \text{Rerank}(\mathcal{D}_{final}, Q) \quad (3)$$

Finally, these top-k documents are fed into the answer generator. We utilize a few-shot CoT prompt-

ing strategy to reason through the optimized evidence and generate the final answer.

$$A = \mathcal{M}_{CoT}(Q, \mathcal{D}_{reranked}) \quad (4)$$

4 Experimental Setup

4.1 Datasets and Baselines

We evaluated PROGRAM on various benchmark datasets spanning diverse reasoning regimes including multi-hop (HotpotQA (Yang et al., 2018), 2WikiMultihopQA (Ho et al., 2020)), commonsense (ARC-Challenge (Clark et al., 2018)), scientific (MMLU-Pro (Wang et al., 2024b)), and biomedical (MedQA (Jin et al., 2021)) domains. Following prior state-of-the-art (SOTA) work (Shao et al., 2023; Wang et al., 2024a), we evaluated on the first 500 questions of each dataset, using the development splits for multi-hop datasets and the test splits for commonsense, scientific, and biomedical datasets. For retrieval, we used the Generalized Text Embeddings (GTE) (Li et al., 2023) model as our retriever. The knowledge base for multi-hop, commonsense, and scientific tasks is built from the 2017 Wikipedia abstract dumps, following the implementation of Khattab et al. (2024), while biomedical experiments use a PubMed abstract corpus constructed by Xiong et al. (2024). We compared PROGRAM against a suite of strong retrieval-based baselines including IRCoT (Trivedi et al., 2023), FLARE (Jiang et al., 2023b), ProbTree (Cao et al., 2023), Self-RAG (Asai et al., 2024), and SeaKR (Yao et al., 2025).

4.2 Backbones and Rerankers

To evaluate of our method, we used four instruction-tuned LLMs of GPT-4o-mini[1], Gemma-2-9B (Riviere et al., 2024), Qwen-2.5-7B (Yang et al., 2024), and Mistral-Small-24B (Jiang et al., 2023a). For retrieval reranking, we explore three types covering different efficiency–quality trade-offs of BGE-large-en-v1.5 (Xiao et al., 2024) as an efficient Bi-Encoder baseline, ColBERTv2 (Santhanam et al., 2022) as a late-interaction model balancing speed and granularity, and GTE-Reranker-ModernBERT-Base (Li et al., 2023) as a Cross-Encoder that targets maximal semantic relevance at higher computational cost.

5 Overall Results of PROGRAM

PROGRAM establishes a new state-of-the-art on knowledge-intensive reasoning, particularly by acting as a "logic scaffold" for smaller architectures like Gemma-2-9B-Instruct and GPT-4o-mini. As shown in Tables 2, on HotPotQA, the substantial 24.0% F1 gain over Self-RAG underscores the superiority of structured intent over unstructured reflection; this efficacy extends to proprietary models as well, where PROGRAM enables GPT-4o-mini to achieve a dominant 64.2% F1, setting the absolute ceiling for the task among all evaluated backbones in Table 3. While Self-RAG relies on scalar confidence tokens that may fail to capture semantic nuance, PROGRAM's explicit logical sub-queries (e.g., 'temporal', 'causal') enforce a rigorous retrieval path that prevents the reasoning drift typical in standard CoT or token-level critiques.

Beyond standard multi-hop datasets, PROGRAM demonstrates exceptional domain adaptability compared to uncertainty-based baselines like SeaKR and tree-search methods like ProbTree. On the ARC-Challenge with Qwen-2.5-7B in Table 4, PROGRAM achieves a remarkable 92.6% accuracy, significantly outperforming SeaKR (79.0%) and ProbTree (88.8%). This performance differential highlights that while SeaKR's internal uncertainty metrics effectively capture when to retrieve, they lack the structural guidance on what to retrieve for commonsense reasoning. Furthermore, on specialized biomedical tasks like MedQA in Table 5, PROGRAM with Mistral-Small-24B attains 82.4% accuracy which surpasses the ProbTree by nearly 10 points. This confirms that our "Advanced" program types (e.g., 'Diagnosis', 'Treatment Reasoning') successfully bridge the gap between generalist LLM capabilities and domain-specific rigor, a feat that unstructured, confidence-driven active retrieval methods like FLARE struggle to replicate without explicit structural decomposition.

Finally, PROGRAM proves to be a highly efficient reasoning amplifier for smaller open-source models. On the comprehensive MMLU-Pro benchmark, PROGRAM enables the Gemma-2-9B model to achieve 60.2% accuracy, effectively matching or exceeding the performance of significantly larger architectures using standard RAG approaches. This result is driven by our dual-metric optimization strategy, which ensures that sub-queries are not only semantically relevant but informationally additive. By mathematically filtering out low-utility reasoning paths before they consume context window space, PROGRAM mitigates the "reasoning-in-noise" problem that plagues iterative baselines like IRCoT, allowing compact models to maintain long-

[1] https://platform.openai.com/docs/models/gpt-4o-mini

Table 2: Comparison of RAG Methods (Gemma-2-9B-Instruct)

	HotPotQA		2WikiMultihopQA		ARC-C	MMLU-Pro	MedQA
Method	**Exact Match**	**F1**	**Exact Match**	**F1**	**Accuracy**	**Accuracy**	**Accuracy**
Direct	0.3560	0.4404	0.2580	0.3087	0.8780	0.5520	0.5200
CoT	0.3360	0.4175	0.2200	0.2612	0.8780	0.5820	0.5080
IRCoT	0.3260	0.3583	0.2220	0.2546	0.7680	0.5120	0.5700
FLARE	0.3640	0.3958	0.3640	0.3988	0.8820	0.4380	0.5080
ProbTree	0.3500	0.4205	0.3160	0.3638	0.8540	0.5880	0.5600
Self-RAG	0.4560	0.4765	0.4540	0.4667	0.8800	0.5360	0.5820
SeaKR	0.3440	0.4071	0.3720	0.4120	0.7620	0.3960	0.3940
PROGRAM (Ours)	**0.5520**	**0.5910**	**0.5220**	**0.5640**	**0.9020**	**0.6020**	**0.5920**

Table 3: Comparison of RAG Methods (GPT-4o-mini)

	HotPotQA		2WikiMultihopQA		ARC-C	MMLU-Pro	MedQA
Method	**Exact Match**	**F1**	**Exact Match**	**F1**	**Accuracy**	**Accuracy**	**Accuracy**
Direct	0.3620	0.4156	0.3440	0.3752	0.8880	0.6660	0.6500
CoT	0.4000	0.4485	0.3960	0.4221	0.9120	0.7020	0.7560
IRCoT	0.4920	0.5543	0.5300	0.5586	0.8620	0.6000	0.7260
FLARE	0.4380	0.4801	0.4060	0.4397	0.8940	0.4060	0.7360
ProbTree	0.4220	0.4701	0.5600	0.5980	0.9200	0.7280	0.7680
Self-RAG	0.5480	0.5583	0.5720	0.5781	0.8260	0.6600	0.6640
SeaKR	0.5300	0.5878	0.5340	0.5750	0.6820	0.6200	0.6980
PROGRAM (Ours)	**0.5920**	**0.6420**	**0.5900**	**0.6111**	**0.9520**	**0.7760**	**0.8220**

Table 4: Comparison of RAG Methods (Qwen-2.5-7B-Instruct)

	HotPotQA		2WikiMultihopQA		ARC-C	MMLU-Pro	MedQA
Method	**Exact Match**	**F1**	**Exact Match**	**F1**	**Accuracy**	**Accuracy**	**Accuracy**
Direct	0.3380	0.3953	0.3120	0.3668	0.8580	0.4940	0.5140
CoT	0.3380	0.3968	0.2680	0.3036	0.8580	**0.6080**	0.4900
IRCoT	0.4020	0.4427	0.3240	0.3581	0.8780	0.5440	0.5380
FLARE	0.3120	0.3501	0.2840	0.3248	0.8480	0.3860	0.5260
ProbTree	0.3240	0.3733	0.3860	0.4135	0.8880	0.6000	0.5460
Self-RAG	0.4540	0.4649	0.4640	0.4711	0.8400	0.4460	0.5380
SeaKR	0.4840	0.5171	0.5120	0.5386	0.7900	0.5840	0.4680
PROGRAM (Ours)	**0.4900**	**0.5301**	**0.5520**	**0.5801**	**0.9260**	**0.6080**	**0.5480**

Table 5: Comparison of RAG Methods (Mistral-Small-24B-Instruct)

	HotPotQA		2WikiMultihopQA		ARC-C	MMLU-Pro	MedQA
Method	**Exact Match**	**F1**	**Exact Match**	**F1**	**Accuracy**	**Accuracy**	**Accuracy**
Direct	0.4580	0.5049	0.3800	0.4142	0.9040	0.7360	0.5340
CoT	0.3620	0.3985	0.4580	0.4948	0.9140	0.7480	0.7280
IRCoT	0.5080	0.5617	0.5360	0.5693	0.8720	0.5260	0.5540
FLARE	0.3600	0.4021	0.3800	0.4268	0.8900	0.4420	0.6340
ProbTree	0.4080	0.4637	0.5880	0.6148	**0.9340**	**0.7600**	0.7280
Self-RAG	0.5380	0.5478	0.5280	0.5333	0.9020	0.7060	0.6280
SeaKR	0.5280	0.5673	0.4740	0.5104	0.7760	0.5980	0.4920
PROGRAM (Ours)	**0.5980**	**0.6350**	**0.6300**	**0.6610**	**0.9340**	0.7400	**0.8240**

Table 6: Performance comparison across different module-wise using GPT-4o-mini. PTS represents Program-Type Selection and IAPP stands for Iterative Active Program Pruning, and FAG imply the Final Answer Generation.

	HotpotQA		2WikiMultihopQA		ARC-C	MMLU-Pro	MedQA
	Exact Match	F1	Exact Match	F1	Accuracy	Accuracy	Accuracy
PROGRAM	**0.5920**	**0.6420**	**0.5900**	**0.6111**	**0.9520**	**0.7760**	**0.8220**
w/ PTS-only	0.1440	0.3166	0.1980	0.2658	0.8880	0.3040	0.6560
w/ IAPP-only	0.1480	0.3332	0.1700	0.2572	0.8960	0.3160	0.6820
w/ FAG-only	0.2580	0.3380	0.3620	0.4133	0.9000	0.3000	0.7120
w/ PTS+IAPP	0.1640	0.3420	0.1660	0.2598	0.8800	0.3260	0.6620
w/ PTS+FAG	0.2420	0.3235	0.2780	0.3180	0.9060	0.3820	0.7400
w/ IAPP+FAG	0.2560	0.3345	0.2840	0.3436	0.9060	0.3920	0.7120

Table 7: Comparison between the program types of basic-only, basic + advanced programs, and all program types on MMLU-Pro, HotpotQA, and 2WikiMultihopQA (2WMQA) datasets with the Gemma-2-9B-Instruct. ACC stands for Accuracy, EM represents the Exact Match, and LAT is the latency.

Program Types	MMLU-Pro		HotpotQA		2WMQA	
	ACC	LAT	EM	LAT	EM	LAT
Basic-only	0.5920	-	0.5120	-	0.5400	-
Basic+Adv.	0.6120	2.2×	0.5400	1.0×	0.5560	1.0×
All Types	0.6240	5.1×	0.5400	2.0×	0.5600	2.8×

horizon coherence in complex multi-step reasoning tasks. Several cases on MMLU-Pro show marginal improvements over other benchmark datasets. This is likely due to the dataset's requirement for not only understanding scientific theories and terminology, but also performing computation over LaTeX-formatted equations, which aligns with limitations of LLMs in mathematical reasoning (Chen et al., 2023; Gao et al., 2023b).

6 Analysis

6.1 Ablation Study

We conducted a series of ablation studies to examine how key design choices of PROGRAM affect performance and efficiency. For all ablation studies, we sampled 250 instances from each dataset, ensuring no overlap across different ablation experiments. Also, we fixed to our best configuration of our framework to a maximum of 7 iterations, 10 candidate sub-queries per iteration, and 10 retrieved documents per candidate for benchmark datasets.

Impact of module-wise configurations. Table 6 (see also Appendix section for other results) presents the ablation study for GPT on module-wise comparison. The isolated performance of individual components confirms their structural interdependence, as no single module can effectively handle complex reasoning. On HotpotQA, using only the PTS (0.3166 F1) or IAPP (0.3332 F1) results in a catastrophic drop from the full framework's 0.6420 F1. A similar pattern appears on MedQA, where even the strongest single component, 'FAG-only' (0.7120), significantly underperforms the complete system (0.8220). These results indicate that structured intent (PTS) without execution (IAPP) is theoretical, while retrieval (IAPP) without intent lacks direction.

Combining only two modules yields negligible gains, demonstrating that PROGRAM functions as a tightly coupled tripartite system. On HotpotQA, configurations like PTS+IAPP (0.3420 F1) and PTS+Rerank (0.3235 F1) barely surpass single-module baselines and remain 30 points behind the full framework. This reveals that the Reranker acts as a critical "noise gate"; without it, the diverse evidence collected by the IAPP overwhelms the model with noise. The massive performance leap of doubling the F1 score occurs only when all three components work in tandem, validating the necessity of the complete end-to-end pipeline.

Efficiency of program type selection. While program diversity could potentially enhance performance, the extent of its contribution remains unclear. We evaluated program type diversity on MMLU-Pro as it covers various domains, comparing 'Basic-only', 'Basic+Advanced', and 'All Program Types'. 'Basic+Advanced' extends 'Basic-only' by selecting additional 'Advanced' program types. Table 7 shows that selecting more program types improves accuracy at the cost of increased latency. Specifically, 'Basic+Advanced' and 'All Pro-

Table 8: Performance comparison of reranker architectures on MMLU-Pro dataset using Gemma-2-9B-Instruct, Qwen-2.5-7B-Instruct, and GPT-4o-mini models. ACC stands for accuracy and LAT represents the latency.

Rerank Types	Gemma		Qwen		GPT	
	ACC	LAT	ACC	LAT	ACC	LAT
Bi-Encoder	0.6400	65.61	0.6040	50.05	0.6880	66.32
ColBERT	0.6200	65.48	0.6060	50.79	0.6880	64.37
Cross-Encoder	0.5960	63.12	0.6120	52.22	0.6840	66.16

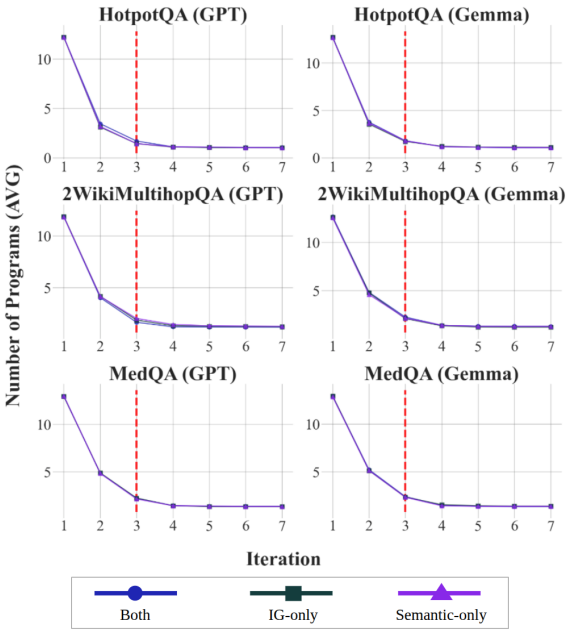

Figure 3: Average number of remaining programs per iteration for different Semantic-Information Gain scoring methods across HotpotQA, 2WikiMultihopQA, and MedQA with GPT-4o-mini and Gemma-2-9B-Instruct.

gram Types' incur approximately 2.2× and 5.1× the latency of 'Basic-only' respectively.

This result follows a similar pattern in HotpotQA and 2WikiMultihopQA datasets. This trade-off occurs because each additional program type requires independent retrieval and reasoning operations, causing computational cost to scale linearly with the number of active programs. This makes it clear that selectively enabling only high-utility programs provide a better accuracy-efficiency trade-off than activating all available program types.

Comparison of reranker architectures. While the presence of reranking is crucial, the specific choice of reranker architecture between Bi-Encoder, ColBERT and Cross-Encoder shows variable impact depending on the backbone model. Results on the MMLU-Pro dataset as shown in Table 8 indicate no single architecture consistently outperforms the others as the Bi-Encoder achieves the highest EM for Gemma (0.6400), whereas the Cross-Encoder performs best for Qwen (0.6120). For the GPT model, performance remains largely stable across all types, with marginal differences (0.6880 for both Bi-Encoder and ColBERT and; 0.6840 for Cross-Encoder). This suggests that while reranking is vital, the system's overall effectiveness is not primarily dependent on the specific reranker type, as also reflected in minimal latency gaps across different architectures.

Table 9: Comparison of the final program selection modes between multiple and single program(s) in terms of its accuracy performance with the confidence level of p-value. ACC stands for accuracy.

Program Modes	MMLU-Pro	
	ACC	p-value
Multiple	0.6240	$p < 0.05$
Single	0.6120	$p < 0.05$

Final program selection mode. Given that programs are compared based on their support scores, selecting only the program with the highest support score might be sufficient, rather than using multiple programs in the final stage. We controlled the program-selection mode at the 'Final Answer Generation' stage on MMLU-Pro. Table 9 shows that the multiple-program approach outperforms single-program selection. The performance gains can be attributed to aggregating evidence from diverse reasoning paths, which yields richer context and more comprehensive support for answer generation than relying solely on documents retrieved by a single program. This suggests that leveraging multiple well-supported reasoning trajectories is more effective than committing to a single path, even when that path has the highest support score.

Semantic-information gain score. We disentangled our scoring mechanism on HotpotQA, 2WikiMultihopQA and MedQA to assess the relative importance of Information Gain (IG) and Semantic Score (SEM). IG-only scores documents by incremental information contribution, while SEM-only ranks by semantic similarity. Table 10 reveals dataset-dependent patterns in which the HotpotQA, combining both components (IG+SEM) achieves the best performance, while the 2WikiMultihopQA, IG-only performs best. This suggests that IG plays a more critical role than SEM in our framework. Figure 3 demonstrates the scoring method compari-

Table 10: Comparison of Scoring Methods. IG and SEM represents the Information Gain and Semantic scores.

Scoring Methods	Gemma-2-9B-Instruct				GPT-4o-mini			
	HotPotQA		2WikiMultihopQA		HotPotQA		2WikiMultihopQA	
	EM	F1	EM	F1	EM	F1	EM	F1
IG-only	0.4880	0.5205	**0.5040**	**0.5480**	0.5160	0.5661	**0.5960**	**0.6228**
SEM-only	0.4960	0.5372	0.5000	0.5432	0.5160	0.5623	0.5880	0.6176
IG+SEM	**0.5320**	**0.5640**	0.4920	0.5372	**0.5240**	**0.5689**	0.5760	0.6117

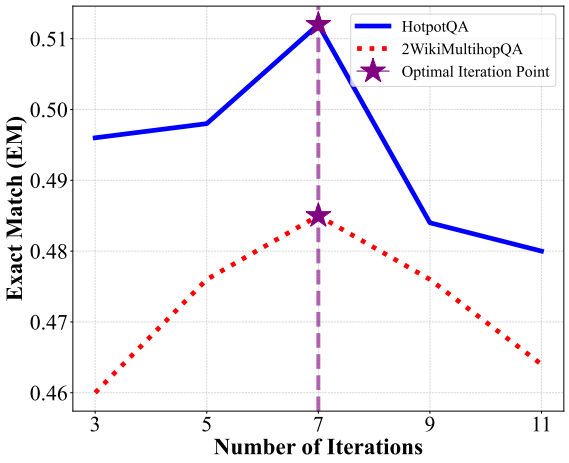

Figure 4: Performance comparison across different number of iterations on HotpotQA and 2WikiMultihopQA with Gemma-2-9B-Instruct.

Table 11: Performance comparison across different number of candidate sub-queries on HotpotQA and 2WikiMultihopQA using Gemma-2-9B-Instruct. IMP represents the percentage (%) of improvement based on the optimal number of sub-queries of 10 sub-queries.

Num. of Sub-queries	HotpotQA		2WikiMultihopQA	
	EM	IMP	EM	IMP
1	0.4920	-8.13	0.4480	-20.54
5	0.5040	-5.56	0.5240	-3.05
10	**0.5320**	-	**0.5400**	-
15	0.5240	-1.53	0.5120	-5.47
20	0.5280	-0.76	0.5000	-8.00

son of all three approaches that exhibit rapid initial program reduction followed by convergence toward a single program, indicating that our iterative filtering mechanism effectively identifies high-quality reasoning paths independent of the specific scoring formulation.

6.2 Sensitivity Study

In this section, we examined how the performance changes with different hyperparameter settings. We used the same best configuration settings while varying the number of iterations and candidate sub-queries to isolate its impact.

Number of iterations. Figure 4 evaluates the performance across varying number of iteration limits on the HotpotQA and 2WikiMultihopQA datasets. The results demonstrate that increasing the number of iterations initially improves accuracy, validating the benefits of iterative refinement. However, performance peaks at 7 iterations, achieving the highest evaluation scores of 0.5120 and 0.4850, respectively. Extending the process beyond this threshold leads to a decline in performance across both datasets, identifying 7 iterations as the optimal setting for our framework.

Number of candidate sub-queries. Table 11 shows how performance changes with the number of candidate sub-queries generated per program during the 'Program-Type Selection' stage. We observed that the evaluation metric generally increases as the number of candidates grows up to 10, and then decreases beyond this point. This consistent pattern across datasets indicates that using 10 candidate sub-queries is an optimal choice, matching to our best configuration settings.

7 Conclusion

We introduced PROGRAM, a framework that elevates retrieval-augmented generation from unstructured semantic matching to structured, program-guided reasoning. By treating retrieval as execution of specific program types, and integrating dual-metric optimization with statistical pruning, PROGRAM achieves SOTA performance across five benchmarks with four LLM backbones. It demonstrates substantial gains on complex multi-hop reasoning, outperforming strong baselines including IRCoT, FLARE, Self-RAG, and SeaKR by up to 24% relative improvement on HotpotQA and 13.2% on MedQA.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF), funded by the Korean government under Grant No. RS-2023-00273751.

Limitations

Although PROGRAM demonstrates strong performance across diverse reasoning tasks, our program type space is currently defined through manual design, which may not fully capture other effective reasoning paths when it comes to other domains that we have not yet explored on such as financial, visual, and table QA. Fortunately, our empirical observations showed some promising performance across various domains of which are interested to the majority of researchers and practitioners alike, and therefore, we strongly believe that our PROGRAM can be tested on these domains in the future work.

References

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations*.

Shulin Cao, Jiajie Zhang, Jiaxin Shi, Xin Lv, Zijun Yao, Qi Tian, Lei Hou, and Juanzi Li. 2023. Probabilistic tree-of-thought reasoning for answering knowledge-intensive complex questions. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 12541–12560, Singapore. Association for Computational Linguistics.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Trans. Mach. Learn. Res.*, 2023.

Qinyuan Cheng, Xiaonan Li, Shimin Li, Qin Zhu, Zhangyue Yin, Yunfan Shao, Linyang Li, Tianxiang Sun, Hang Yan, and Xipeng Qiu. 2024. Unified active retrieval for retrieval augmented generation. In *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, pages 17153–17166. Association for Computational Linguistics.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *ArXiv*, abs/1803.05457.

Jinyuan Fang, Zaiqiao Meng, and Craig MacDonald. 2024. TRACE the evidence: Constructing knowledge-grounded reasoning chains for retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, pages 8472–8494. Association for Computational Linguistics.

Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. 2023a. Precise zero-shot dense retrieval without relevance labels. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 1762–1777. Association for Computational Linguistics.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023b. PAL: program-aided language models. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 10764–10799. PMLR.

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

Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong Park. 2024. Adaptive-rag: Learning to adapt retrieval-augmented large language models through question complexity. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 7036–7050. Association for Computational Linguistics.

Albert Qiaochu Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023a. Mistral 7b. *ArXiv*, abs/2310.06825.

Zhengbao Jiang, Frank Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023b. Active retrieval augmented generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7969–7992, Singapore. Association for Computational Linguistics.

D. Jin, E. Pan, N. Oufattole, W.-H. Weng, H. Fang, and P. Szolovits. 2021. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *Applied Sciences*, 11(13):6421.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2024. Dspy: Compiling declarative language model calls into state-of-the-art pipelines. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Solomon Kullback and R. A. Leibler. 1951. On information and sufficiency. *Annals of Mathematical Statistics*, 22:79–86.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. 2023. Towards general text embeddings with multi-stage contrastive learning. *CoRR*, abs/2308.03281.

Zhuoqun Li, Xuanang Chen, Haiyang Yu, Hongyu Lin, Yaojie Lu, Qiaoyu Tang, Fei Huang, Xianpei Han, Le Sun, and Yongbin Li. 2025. Structrag: Boosting knowledge intensive reasoning of llms via inference-time hybrid information structurization. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Rodrigo Nogueira, Wei Yang, Jimmy Lin, and Kyunghyun Cho. 2019. Document expansion by query prediction. *CoRR*, abs/1904.08375.

Gemma Team Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, L'eonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ram'e, Johan Ferret, Peter Liu, Pouya Dehghani Tafti, Abe Friesen, Michelle Casbon, Sabela Ramos, Ravin Kumar, Charline Le Lan, Sammy Jerome, Anton Tsitsulin, and 176 others. 2024. Gemma 2: Improving open language models at a practical size. *ArXiv*, abs/2408.00118.

Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. 2022. ColBERTv2: Effective and efficient retrieval via lightweight late interaction. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3715–3734, Seattle, United States. Association for Computational Linguistics.

Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 9248–9274, Singapore. Association for Computational Linguistics.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2020. Mpnet: Masked and permuted pre-training for language understanding. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10014–10037, Toronto, Canada. Association for Computational Linguistics.

Haoyu Wang, Ruirui Li, Haoming Jiang, Jinjin Tian, Zhengyang Wang, Chen Luo, Xianfeng Tang, Monica Xiao Cheng, Tuo Zhao, and Jing Gao. 2024a. BlendFilter: Advancing retrieval-augmented large language models via query generation blending and knowledge filtering. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1009–1025, Miami, Florida, USA. Association for Computational Linguistics.

Liang Wang, Nan Yang, and Furu Wei. 2023. Query2doc: Query expansion with large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 9414–9423. Association for Computational Linguistics.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. 2024b. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35:*

Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. 2024. C-pack: Packed resources for general chinese embeddings. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 641–649, New York, NY, USA. Association for Computing Machinery.

Guangzhi Xiong, Qiao Jin, Zhiyong Lu, and Aidong Zhang. 2024. Benchmarking retrieval-augmented generation for medicine. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 6233–6251, Bangkok, Thailand. Association for Computational Linguistics.

Qwen An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxin Yang, Jingren Zhou, Junyang Lin, and 25 others. 2024. Qwen2.5 technical report. *ArXiv*, abs/2412.15115.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Zijun Yao, Weijian Qi, Liangming Pan, Shulin Cao, Linmei Hu, Liu Weichuan, Lei Hou, and Juanzi Li. 2025. SeaKR: Self-aware knowledge retrieval for adaptive retrieval augmented generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 27022–27043, Vienna, Austria. Association for Computational Linguistics.

Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. 2021. Optimizing dense retrieval model training with hard negatives. In *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021*, pages 1503–1512. ACM.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. Bertscore: Evaluating text generation with BERT. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V. Le, and Ed H. Chi. 2023. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

A Appendix

Table 12: Comparison between document selection strategies of using reranking (active) and initial document slicing (inactive) on HotpotQA and 2WikiMultihopQA with Qwen-2.5-7B-Instruct.

Rerank Activation	HotpotQA		2WikiMultihopQA	
	EM	**F1**	**EM**	**F1**
Inactive	0.4200	0.4532	0.3240	0.3543
Active	**0.4720**	**0.5061**	**0.4280**	**0.4682**

Impact of reranking activation. Activating the reranking module consistently enhances performance across multi-hop reasoning tasks compared to the naive approach of simply *slicing* ("Inactive") the top-k documents. As shown in Table 12, employing reranking ("Active") with the Qwen-2.5-7B model on HotpotQA boosts the Exact Match (EM) score from 0.4200 to 0.4720 and the F1 score from 0.4532 to 0.5061. A similar trend is observed on 2WikiMultihopQA, where performance increases from 0.3240 EM and 0.3543 F1 (Inactive) to 0.4280 EM and 0.4682 F1 (Active). These results confirm that reranking accumulated documents effectively filters relevant evidence from multiple reasoning paths, yielding more accurate answers than relying solely on initial retrieval positions.

B Pseudo-code for PROGRAM

We provide our pseudo-code in Algorithm 1 for our proposed PROGRAM framework which consists of three main stages of Program-Type Selection with dual-metric optimization, Iterative Active Program Pruning with evidence accumulation, and Final Answer Generation with reranking. Each line of pseudo-code provides a brief description of its operation.

Table 13: Performance comparison across different module-wise using Gemma-2-9B-Instruct. PTS represents Program-Type Selection, IAPP stands for Iterative Active Program Pruning, and FAG imply the Final Answer Generation.

	HotpotQA		2WikiMultihopQA		ARC-C	MMLU-Pro	MedQA
	Exact Match	F1	Exact Match	F1	Accuracy	Accuracy	Accuracy
PROGRAM	**0.5520**	**0.5910**	**0.5220**	**0.5640**	**0.9020**	**0.6020**	**0.5920**
w/ PTS-only	0.2120	0.3348	0.1880	0.3029	0.8720	0.3600	0.5700
w/ IAPP-only	0.2420	0.3822	0.2000	0.3268	0.8740	0.3620	0.5700
w/ FAG-only	0.3520	0.4621	0.2940	0.3472	0.8840	0.3800	0.5880
w/ PTS+IAPP	0.2540	0.3980	0.2100	0.3344	0.8600	0.3500	0.5660
w/ PTS+FAG	0.2320	0.3093	0.1440	0.1801	0.8680	0.4480	0.5860
w/ IAPP+FAG	0.2880	0.3828	0.2180	0.2530	0.8740	0.4400	0.5660

Table 14: Performance comparison across different module-wise using Qwen2.5-7B-Instruct.

	HotpotQA		2WikiMultihopQA		ARC-C	MMLU-Pro	MedQA
	Exact Match	F1	Exact Match	F1	Accuracy	Accuracy	Accuracy
PROGRAM	**0.4900**	**0.5301**	**0.5520**	**0.5801**	**0.9260**	**0.6080**	**0.5480**
w/ PTS-only	0.2240	0.3390	0.2280	0.2944	0.8380	0.3220	0.5180
w/ IAPP-only	0.2480	0.3759	0.2300	0.3101	0.8440	0.3420	0.5220
w/ FAG-only	0.2840	0.3554	0.2640	0.3237	0.8760	0.3140	0.5300
w/ PTS+IAPP	0.2640	0.3936	0.2620	0.3304	0.8580	0.3360	0.5120
w/ PTS+FAG	0.2500	0.3356	0.1680	0.2213	0.8160	0.3060	0.5200
w/ IAPP+FAG	0.2880	0.3259	0.2000	0.2512	0.8540	0.3080	0.5320

Table 15: Performance comparison across different module-wise using Mistral-Small-24B-Instruct.

	HotpotQA		2WikiMultihopQA		ARC-C	MMLU-Pro	MedQA
	Exact Match	F1	Exact Match	F1	Accuracy	Accuracy	Accuracy
PROGRAM	**0.5980**	**0.6350**	**0.6300**	**0.6610**	**0.9340**	**0.7400**	**0.8240**
w/ PTS-only	0.3080	0.4347	0.2700	0.3560	0.8780	0.4160	0.6580
w/ IAPP-only	0.3460	0.4705	0.3080	0.3893	0.8900	0.4320	0.6480
w/ FAG-only	0.4000	0.5298	0.4160	0.4853	0.9140	0.5600	0.7120
w/ PTS+IAPP	0.3300	0.4603	0.3320	0.4028	0.8820	0.4220	0.7560
w/ PTS+FAG	0.3340	0.4563	0.3040	0.3914	0.8820	0.4600	0.7020
w/ IAPP+FAG	0.3640	0.4835	0.3580	0.4304	0.9060	0.5600	0.7360

C Prompt Templates for PROGRAM

In this section, we provide our representative prompting templates that we used in testing on all benchmark datasets. Figure 5 provide the examples of our prompt template including instructions for extracting entities, generating sub-query for each programs, and checking for sufficiency of its accumulated documents. Our prompting is the same throughout our experiment, varied by CoT-style few-shot examples for each dataset.

Algorithm 1 PROGRAM

Require: Question Q, Knowledge Base $\mathcal{K}$, LLM $\mathcal{M}$, Max Iterations T
Ensure: Final Answer A

$\mathcal{P} \leftarrow \{p_i\}_{i=1}^{K}$ where $p_i \sim \mathcal{M}_{gen}(Q)$ ▷ Generate candidate programs
$\mathcal{P}_{active} \leftarrow \mathcal{P}$
$\mathcal{P}_{locked} \leftarrow \emptyset$
$\mathcal{P}_{selected} \leftarrow \emptyset$
▷ **1. Program-Type Selection with dual-metric optimization**
for $p_i \in \mathcal{P}$ **do**
  $q_i^{(0)} \leftarrow \mathcal{M}_{gen}(Q, p_i)$ ▷ Generate program-specific sub-query
  $\mathcal{D}_i^{accum} \leftarrow \emptyset$
  $S_{sup}^{(0)}(p_i) \leftarrow \text{Sim}(q_i^{(0)}, \mathcal{D}_i^{accum})$
end for
for $p_i \in \mathcal{P}$ **do**
  $\mathcal{C}_i \leftarrow \{c_j\}_{j=1}^{M}$ where $c_j \sim \mathcal{M}_{gen}(Q, q_i^{(0)}, p_i)$ ▷ Generate candidate sub-queries per program
  $D_{i,c_j} \leftarrow \text{Retrieve}(c_j, \mathcal{K})$ for each $c_j \in \mathcal{C}_i$
end for

for $p_i \in \mathcal{P}$ **do**
  for $c_j \in \mathcal{C}_i$ **do**
    $S_{IG}(c_j) \leftarrow \frac{1}{|\mathcal{P}|} \sum_{p_k \in \mathcal{P}} D_{KL}(P(E|c_j, p_k) || P(E|p_k))$ ▷ Information Gain on evidence
    $S_{sem}(c_j) \leftarrow \text{Sim}(Q, c_j)$ ▷ Semantic similarity to original question
  end for
  $q_i^* \leftarrow \text{argmax}_{c_j \in \mathcal{C}_i}(S_{IG}(c_j) + S_{sem}(c_j))$
  $D_i^* \leftarrow D_{i,q_i^*}$
  $\mathcal{D}_i^{accum} \leftarrow \mathcal{D}_i^{accum} \cup \bigcup_{c_j \in \mathcal{C}_i} D_{i,c_j}$
end for
▷ **2. Iterative Active Program Pruning with evidence accumulation**
for $t = 1$ to T **do**
  if $\mathcal{P}_{active} = \emptyset$ **then break**
  end if
  $\mathcal{P}_{active} \leftarrow \mathcal{P}_{active} \setminus \mathcal{P}_{locked}$ ▷ Update active programs
  for $p_i \in \mathcal{P}_{active}$ **do**
    $S_{sup}^{(t)}(p_i) \leftarrow \text{Sim}(q_i^*, \mathcal{D}_i^{accum})$
    is_suff $\leftarrow \mathcal{M}(Q, \mathcal{D}_i^{accum}, p_i)$ ▷ Check if program has sufficient documents
    if is_suff **then** $\mathcal{P}_{locked} \leftarrow \mathcal{P}_{locked} \cup \{p_i\}$
    end if
  end for
  if $t < T$ **then**
    for $p_i \in \mathcal{P}_{active}$ **do**
      $\mathcal{D}_i^{accum} \leftarrow \mathcal{D}_i^{accum} \cup \text{Retrieve}(q_i^*, \mathcal{K})$
    end for
  end if
  $\mathcal{P}_{locked} \leftarrow \mathcal{P}_{locked} \cup t\text{-test}(\{S_{sup}^{(t)}(p) | p \in \mathcal{P}_{active}\})$ ▷ Lock statistically significant programs
  $\mathcal{P}_{selected} \leftarrow \mathcal{P}_{selected} \cup \mathcal{P}_{locked}$
end for
▷ **3. Final Answer Generation with reranking**
$\mathcal{D}_{final} \leftarrow \bigcup_{p_i \in \mathcal{P}_{selected}} \mathcal{D}_i^{accum}$
$\mathcal{D}_{reranked} \leftarrow \text{Rerank}(\mathcal{D}_{final}, Q)$
$A \leftarrow \mathcal{M}_{CoT}(Q, \mathcal{D}_{reranked})$
return A

Prompt for Extracting Entities

```
Extract the key entities, keywords, and
important terms from the following query.
Focus on extracting:
- Named entities (e.g. people, places,
organizations, dates, etc.)
- Important keywords and concepts
- Numbers and quantities
- Key phrases that are essential to the query

Query: {sub_query}

Output ONLY a JSON array of extracted
entities/keywords without any explanations or
additional text.
Format: ["entity1", "entity2", "entity3", ...]
```

Prompt for Program Hypothesis Generation

```
You are an expert reasoning engine. The
following default reasoning strategies have
already been applied to answer the question:

Basic Strategies: {basic_types}
Advanced Strategies: {advanced_types}
Input Question: {question}

Task:
1. Analyze whether the basic strategies are
sufficient to answer the question.
2. If additional strategies are needed, select
the most relevant additional program types
from the "Advanced Available Strategies" list
above. Choose as many as needed - there is no
limit.
3. If no additional strategies are needed, return
an empty array [ ].
4. For each selected advanced type, generate a
specific program description and slots.

Output ONLY a JSON array of objects (or
empty array [ ] if no advanced programs
needed) using exactly this format:
[
    {{
        "type": "multi_hop",
        "description": "Break down the question
into step-by-step queries",
        "slots": {{}}
    }}
]
```

Prompt for Generating Sub-Query for Programs

```
You are an AI language model assistant.
Your task is to generate a highly specific
search query targeting the {prog_type}
({prog_desc}) aspect of the user question.
You must also AVOID these list of sub-
queries: {previous_queries}.

By generating a strategy-aligned version of
the user question, your goal is to help the
user overcome the limitations of
{prog_type} reasoning by strictly following
these rules:
1. If the strategy is 'temporal', you MUST
use words like "when", "year", "timeline",
"date".
                    ...

Output ONLY the rewritten question.
Original question: {question}
```

Prompt for Checking Document Sufficiency

```
You are evaluating whether the retrieved
documents are sufficient to answer the
following question.

Question: {question}

Program Type: {program_type}

Retrieved Documents:
{docs_text}

Task: Determine if the retrieved documents
contain enough information to answer the
question. Consider:
1. Do the documents directly address the
question?
2. Are there any critical gaps in information?
3. Is the information complete enough to
provide a confident answer?

Output ONLY "SUFFICIENT" if the
documents are enough to answer the
question, or "INSUFFICIENT" if more
information is needed. Do not provide any
explanations, only output one word: either
"SUFFICIENT" or "INSUFFICIENT".
```

Figure 5: A list of instruction-prompt templates for our proposed PROGRAM framework.