Scaling Test-Time Inference with Policy-Optimized, Dynamic Retrieval-Augmented Generation via KV Caching and Decoding

Sakhinana Sagar Srinivas
Tata Research
Bengaluru, India
sagar.sakhinana@tcs.com

Shivam Gupta
Tata Research
Noida, India
g.shivam4@tcs.com

Akash Das
Tata Research
Bengaluru, India
akash.das4@tcs.com

Venkataramana Runkana
Tata Research
Pune, India
venkat.runkana@tcs.com

Abstract

We present a comprehensive framework for enhancing Retrieval-Augmented Generation (RAG) systems through dynamic retrieval strategies and reinforcement fine-tuning. This approach significantly improves large language models on knowledge-intensive tasks, including open-domain question answering and complex reasoning. Our framework integrates two complementary techniques: Policy-Optimized Retrieval-Augmented Generation (PORAG), which optimizes the use of retrieved information, and Adaptive Token-Layer Attention Scoring (ATLAS), which dynamically determines retrieval timing and content based on contextual needs. Together, these techniques enhance both the utilization and relevance of retrieved content, improving factual accuracy and response quality. Designed as a lightweight solution compatible with any Transformer-based LLM without requiring additional training, our framework excels in knowledge-intensive tasks, boosting output accuracy in RAG settings. We further propose CRITIC, a novel method to selectively compress key-value caches by token importance, mitigating memory bottlenecks in long-context applications. The framework also incorporates test-time scaling techniques to dynamically balance reasoning depth and computational resources, alongside optimized decoding strategies for faster inference. Experiments on benchmark datasets show that our framework reduces hallucinations, strengthens domain-specific reasoning, and achieves significant efficiency and scalability gains over traditional RAG systems. This integrated approach advances the development of robust, efficient, and scalable RAG systems across diverse applications.

Keywords

Retrieval-Augmented Generation, Reinforcement Learning, Test-Time Inference Scaling, Memory-Efficient Inference, Question Answering, Knowledge-Intensive NL

ACM Reference Format:
Sakhinana Sagar Srinivas, Shivam Gupta, Akash Das, and Venkataramana Runkana. 2025. Scaling Test-Time Inference with Policy-Optimized, Dynamic Retrieval-Augmented Generation via KV Caching and Decoding. In . ACM KDD 2025 Toronto, 33 pages.

Conference'31, Toronto, ON, Canada

1 Introduction

Retrieval-Augmented Generation (RAG, [26, 38, 43]) has gained significant interest in Natural Language Processing for enhancing large language models (LLMs) on knowledge-intensive tasks through external information retrieval, with applications across search engines, conversational agents, chatbots, and many other applications. RAG addresses key LLM limitations, including hallucinations, outdated information, and insufficient domain-specific knowledge, particularly in open-domain question answering. Retrieval-Augmented Fine-Tuning (RAFT [60]) advances this approach by integrating retrieval methods with language model supervised fine-tuning. Unlike traditional RAG, which simply retrieves documents for generation, RAFT trains the language model alongside the retrieval mechanism, teaching it to dynamically leverage external knowledge, prioritize relevant content while ignoring distractors for improved performance in domain-specific RAG contexts (e.g., open-book and in-domain question answering). Building on advancements in LLM training methodologies, DeepSeek has enhanced its AI models, notably DeepSeek-R1 [19, 29, 36], by implementing Group Relative Policy Optimization (GRPO), an advanced reinforcement learning algorithm that improves training efficiency and model performance beyond traditional supervised fine-tuning. GRPO reduces computational overhead by eliminating the value function, using group-based advantage estimation for simplified reward computation, lowering memory usage, and integrating Kullback-Leibler (KL) divergence regularization for stable, efficient training. It outperforms standard Rejection Sampling Fine-Tuning (RFT), which relies on offline sampling, and Online RFT, which dynamically samples from an evolving policy. GRPO also supports process supervision (GRPO+PS), providing step-by-step feedback for improved reasoning, surpassing outcome supervision (GRPO+OS), which evaluates only final answers. Addressing the limitations of static retrieval in traditional RAG, DRAGIN (Dynamic Retrieval-Augmented Generation based on Information Needs, [38]) is an advanced framework that dynamically determines when and what to retrieve during text generation. Unlike methods with fixed retrieval intervals or simplistic query formulations, DRAGIN employs Real-time Information Needs Detection (RIND) to trigger retrieval only when necessary, considering token uncertainty, semantic importance, and influence on future tokens. Its query formulation based on Self-attention (QFS) generates more effective queries by leveraging the full generated context rather than just recent tokens to fill information gaps.

This adaptive approach minimizes redundant retrievals, improves efficiency, and enhances response accuracy. Despite these advancements, integrating external knowledge during inference through RAG enhances the capabilities of LLMs. However, it also introduces challenges, such as increased computational and memory demands. Key-Value (KV) Caching [15, 20, 53] addresses this issue by efficiently managing the memory load resulting from RAG's expanded context window. It optimizes the storage and retrieval of key-value pairs, preventing memory bottlenecks and accelerating the processing of augmented information. In transformer-based LLMs, KV Caching stores intermediate hidden states (keys and values) of previous tokens during attention computation, enabling faster text generation by reusing them for new tokens. This approach reduces redundant calculations, lowers memory usage, and improves efficiency for long sequences, thereby enhancing the contextuality and coherence of LLMs while mitigating the memory overhead introduced by RAG. Test-Time Scaling Inference Techniques [18, 22, 32, 55] address these challenges by dynamically allocating computational resources based on task complexity. Unlike static inference methods, which apply fixed computational effort regardless of task demands, test-time scaling adaptively adjusts reasoning depth and complexity. For simple questions, it reduces unnecessary overhead, enabling faster responses and minimizing hallucinations. For complex or multi-faceted tasks, it increases reasoning depth to improve accuracy and better integrate retrieved context, enabling LLMs to effectively process and reason with augmented context. This adaptive approach mimics human-like deliberative reasoning for knowledge-intensive tasks without costly retraining, enhancing efficiency and performance while maintaining accuracy and reducing hallucinations. Together, RAFT enhances RAG by integrating retrieval with supervised fine-tuning, enabling models to dynamically leverage external knowledge and prioritize relevant content while ignoring distractors. DRAGIN dynamically determines when and what to retrieve during text generation, minimizing redundant retrievals and improving efficiency. KV Caching optimizes memory usage by storing intermediate hidden states, reducing computational overhead in RAG, while Test-Time Scaling dynamically allocates resources based on task complexity. These advancements enable RAG systems to integrate external knowledge more accurately, efficiently, and at scale, ensuring faster and more effective utilization of retrieved data within the LLM framework. While these recent advancements have enhanced retrieval integration in LLMs, significant challenges remain in balancing retrieval fidelity, response quality, and computational efficiency. Current methods often struggle to dynamically determine when and how much external information to incorporate, sometimes overwhelming the model or sacrificing the coherence of its responses. Motivated by these persistent challenges, our work seeks to refine the synergy between retrieval and generation through a dual approach. First, we fine-tune language models via policy optimization, enabling them to more effectively integrate and utilize retrieved content. This refinement not only improves factual alignment but also enhances overall response quality. Second, we introduce a mechanism that selectively triggers external retrieval based on the model's internal state, ensuring that additional information is incorporated only when necessary. This targeted strategy optimizes computational resources while preserving the language model's coherence. In the following sections, we outline our contributions that extend state-of-the-art methods by addressing both the optimization of retrieval-augmented generation and the efficient management of computational overhead. Our contributions are as follows:

- We introduce two complementary techniques to enhance Retrieval-Augmented Generation (RAG) systems: Policy-Optimized Retrieval-Augmented Generation (PORAG) and Adaptive Token-Layer Attention Scoring for Selective Retrieval (ATLAS). PORAG extends GRPO to the RAG setting, fine-tuning pre-trained LLMs using QLoRA (Quantized Low-Rank Adaptation). The parameter-efficient optimization using QLoRA leads to improved performance on in-domain Question-Answering (QA) tasks while mitigating catastrophic forgetting of pre-trained knowledge. PORAG incorporates group-based advantage estimation and a trust-region constrained policy update to ensure stable and robust fine-tuning in retrieval-dependent contexts. Additionally, PORAG employs a dual reward mechanism that explicitly balances retrieval fidelity—ensuring generated responses remain factually aligned with retrieved information—and response quality, which evaluates coherence, fluency, and overall helpfulness beyond factual accuracy. To effectively implement this, specialized linear layer-based reward heads are integrated after the final layer of the pre-trained LLM with QLoRA adapters. Trained reward heads evaluate retrieval fidelity and response quality, and their combined signals form a composite reward for group-based advantage estimation, thus guiding generation policy optimization. ATLAS, on the other hand, dynamically determines when and what to retrieve by analyzing the language model's internal attention patterns. Using Multi-Layer Attention Gradient (MLAG) to detect information gaps and Layerwise Representation Pooling (LRP) to construct targeted queries, ATLAS retrieves the most relevant external information to fill information gaps, improving retrieval precision and ensuring retrieval occurs only when necessary and precisely aligned with the model's information needs. Together, these techniques create a comprehensive RAG system that optimizes both the utilization of retrieved information and the timing of retrieval, significantly improving efficiency, accuracy, and computational overhead. The integration of PORAG and ATLAS addresses key challenges in RAG systems, such as over-reliance on retrieval, inefficient query formulation, and unstable optimization, paving the way for more robust and resource-efficient language models.
- We present CRITIC (Cache Reduction via Importance-based Token Inclusion Criteria), a method that addresses the memory bottleneck in policy-optimized LLMs inference by selectively retaining only the most important tokens in the KV cache. While traditional KV caching already reduces computational cost from quadratic to linear, memory usage still grows proportionally with sequence length, creating limitations for long-context RAG applications. CRITIC determines token importance using a weighted hybrid approach that combines three complementary strategies: attention-based (relationship strength), entropy-based (attention pattern complexity), and gradient-based (prediction sensitivity).

This integrated approach enables flexible compression behavior, with the framework preserving only the highest-scoring tokens based on a configurable ratio. To further enhance real-world applicability, CRITIC incorporates features such as delayed compression activation and memory-pressure-based adaptive ratios as practical optimizations. The architecture-agnostic solution significantly reduces memory requirements while maintaining performance, leading to faster inference and the ability to process longer contexts, particularly benefiting RAG applications that need extended context windows.

- We study the test-time scaling inference performance of policy-optimized LLMs in RAG contexts, focusing on improving response quality without altering model weights by dynamically adjusting reasoning depth, sampling, and validation during inference. We utilize well-known inference scaling techniques, including Self-Consistency, Best-of-N Sampling, Monte Carlo Tree Search (MCTS), and others, each employing unique strategies to enhance output quality, accuracy, and efficiency. These methods trade off increased computational complexity—often exceeding $O(n)$ for standard inference, where n is the sequence length—for improved reliability and response quality, optimizing inference under resource constraints. Many of these techniques leverage Weak-to-Strong Distillation, iteratively refining outputs to converge on higher-quality responses. Each algorithm presents distinct trade-offs in cost, approach, selection method, and other key factors.

2 Proposed Methodology

Current Retrieval-Augmented Generation (RAG) systems face limitations in their optimization approaches, particularly with log-likelihood-based methods like RAFT. To address these constraints, we introduce two complementary innovations: Policy-Optimized Retrieval-Augmented Generation (PORAG) and Adaptive Token-Layer Attention Scoring for Selective Retrieval (ATLAS). Together, these components create a more robust framework that simultaneously optimizes generation quality and retrieval efficiency. PORAG fundamentally reimagines RAG optimization through a reinforcement learning paradigm built on Group Relative Policy Optimization (GRPO). This approach overcomes RAFT's limitations by moving beyond static reference outputs and undifferentiated treatment of retrieved documents. The system's group-based advantage estimation enables comparative evaluation of multiple candidate generations for each query-retrieval pair. At its core, PORAG implements a dual reward mechanism with two specialized components: (1) a retrieval fidelity reward head that precisely measures how well generated outputs reflect the retrieved evidence, and (2) a response quality reward head that assesses broader linguistic properties including coherence, fluency, and task-aligned helpfulness. These reward signals are optimized jointly with the policy through a carefully designed objective function combining clipped surrogate rewards with KL divergence regularization. This formulation ensures stable training while maintaining the model's generative capabilities. Crucially, PORAG maintains inference-time efficiency through single-shot decoding, avoiding the computational overhead of multi-candidate sampling while preserving the speed of standard autoregressive generation. ATLAS complements this approach with a sophisticated, introspection-based retrieval mechanism operating through two coordinated stages. The first stage employs Multi-Layer Attention Gradient (MLAG) analysis to dynamically detect information gaps. By monitoring shifts in attention distributions across transformer layers and weighting these signals with both token-level uncertainty measures and entropy-normalized attention head importance, the system precisely identifies when retrieval is truly necessary. The second stage implements Layerwise Representation Pooling (LRP) to determine optimal query content. This process evaluates preceding tokens through a hybrid scoring system that combines attention-based salience metrics with deep semantic similarity measures in the model's internal representations. The highest-scoring tokens are then processed through a streamlined prompt template to generate focused, context-aware retrieval queries that directly target the model's knowledge deficiencies. When integrated, PORAG and ATLAS form a comprehensive RAG framework that advances both generation quality and retrieval efficiency. PORAG's learned reward structure ensures outputs maintain high standards of factual accuracy and linguistic quality, while ATLAS's intelligent retrieval mechanism dramatically reduces computational overhead through precision targeting. This dual advancement produces a system that excels in factual reliability, response quality, and operational efficiency - particularly valuable for deployment in scenarios with strict latency or memory constraints. The combined approach represents a significant step forward in developing practical, high-performance RAG systems that maintain both accuracy and efficiency at scale.

3 Experiments

3.1 Datasets

We evaluate our proposed PORAG+ATLAS framework and baselines using three benchmark datasets spanning distinct reasoning tasks: HotpotQA [54], Gorilla [33], and PubMedQA [24]. HotpotQA [54] is a large-scale multi-hop question-answering dataset designed to test RAG frameworks on complex reasoning across multiple sources. Each instance includes a question, an answer, sentence-level supporting facts, and a context comprising multiple Wikipedia paragraphs, each structured as a (title, sentence-list) pair. In the standard distractor setup [54] used during training and evaluation, each question is paired with two gold paragraphs and eight TF-IDF-retrieved distractors, challenging RAG frameworks to identify relevant information amid noise. Gorilla [33], which spans HuggingFace Hub, Torch Hub, and TensorFlow Hub, focuses on code generation from machine learning instructions and is utilized for evaluating RAG frameworks on API call generation. Each JSON entry contains a natural language task description, detailed API documentation specifying the domain (e.g., classification, object detection), framework (PyTorch, TensorFlow), arguments, setup, usage, and functionality, along with the corresponding ground-truth API call. During training, API documentation is concatenated with the instruction to form a retrieval-augmented prompt, enabling the RAG framework to generate context-aware API calls. PubMedQA [24] is a biomedical QA dataset designed to evaluate reasoning over scientific literature. Each sample includes a research question derived from a PubMed title, a context (the abstract excluding its conclusion), a long-form answer (the conclusion), and a ternary

classification label (yes/no/maybe). The dataset combines expert-annotated and machine-generated examples, providing a rigorous benchmark for evidence-based biomedical reasoning.

3.2 Evaluation Metrics

Evaluation metrics are tailored to each dataset's reasoning requirements. For HotpotQA [54], we report Exact Match (EM) and Micro F1 scores for both answer prediction and supporting fact identification, along with Joint EM and Joint F1 scores, which require both components to be correct simultaneously. These joint metrics reflect the RAG framework's combined retrieval and reasoning capabilities. For Gorilla [33], we employ three metrics: (1) Overall Accuracy, based on Abstract Syntax Tree (AST) subtree matching between predicted and ground-truth API calls; (2) Hallucination Error, measuring instances of fabricated APIs; and (3) Wrong API Call Error, capturing valid but incorrectly selected or parameterized APIs [33]. Together, these metrics assess both syntactic correctness and semantic alignment with user intent. For PubMedQA [24], evaluation is framed as a ternary classification task (yes/no/maybe), testing the RAG framework's ability to derive factual conclusions from biomedical abstracts and mirror real-world scientific reasoning.

3.3 Experimental Setup

Our experimental setup rigorously evaluates the integration of Policy-Optimized Retrieval-Augmented Generation (PORAG) and Adaptive Token-Layer Attention Scoring (ATLAS) using Transformer-based LLMs (e.g., Qwen2.5 0.5B/1.5B/3B or Llama 3.2 1B/3B). We selected these base SLMs due to their strong performance, efficient architecture, and compatibility with low-rank fine-tuning techniques, which balance computational efficiency and representational capacity for evaluating PORAG+ATLAS frameworks. We employ Quantized Low-Rank Adaptation (QLoRA) with frozen pre-trained weights quantized to 4-bit NF4, updating only rank-$r = 64$ LoRA adapters ($\alpha = 16$, dropout = 0.05), targeting attention query/value projections and feed-forward layers as the sole trainable parameters. These adapters are optimized using the PORAG objective, which combines group-relative policy improvement with KL-regularized dual reward modeling for retrieval fidelity and response quality. To rigorously evaluate our framework's components, we compare PORAG+ATLAS against six key baselines: (1) **PORAG-only** isolates ATLAS's contribution by showing policy optimization performance without dynamic retrieval; (2) **RAG+ATLAS** evaluates ATLAS's standalone effectiveness with standard retrieval; (3) **RAFT+ATLAS** measures how ATLAS enhances existing retrieval augmented fine-tuning approaches; (4) **PORAG+DRAGIN** benchmarks against alternative dynamic retrieval methods; (5) **GRPO+ATLAS** tests whether RAG-specific policy optimization is necessary; and (6) **RAG-base** establishes the fundamental performance benchmark. Training is conducted using the 8-bit Adam optimizer with weight decay (AdamW), with policy learning rates $\eta_\gamma \in [1\times10^{-6}, 5\times10^{-6}]$; reward model learning rate $\eta_R = 5\times10^{-5}$; group size $G \in \{2, 4\}$; composite reward weighting ($w_{\text{fidelity}} = 0.7$, $w_{\text{quality}} = 0.3$); KL-regularized objectives ($\omega_1 = 100.0$ for policy optimization, $\omega_2 = 0.1$ for divergence control); clipping parameters ($\epsilon = 0.2$ for surrogate objectives, $c_1 = 10.0$ for rewards); and gradient management thresholds ($\sigma_{\min} = 0.1$ for minimum advantage deviation, $c_{\text{value}} = 3.0$, $c_{\text{norm}} = 1.0$). Dual reward heads (ϕ_1, ϕ_2) are jointly optimized using $\mathcal{L}_{\text{fidelity}}$ and $\mathcal{L}_{\text{quality}}$ loss functions, which combine ROUGE-1/2/L, cosine similarity of sentence embeddings, and QA metrics (EM/Micro F1). The ATLAS configuration includes: dynamic retrieval scaling ($\alpha_0 \in [0.7, 1.0], \lambda \in [3, 5]$); Layerwise Representation Pooling with $\beta = 0.7$ attention-representation balance; context selection using $k \in [5, 7]$ tokens; a generation probability threshold $\tau_p = 0.5$; and an embedding temperature $\tau = 2.0$. Using PyTorch hooks to monitor attention weights and hidden states, ATLAS triggers retrieval via Multi-Layer Attention Gradient (MLAG) analysis and constructs queries using focused Layerwise Representation Pooling (LRP). All experiments are conducted on NVIDIA H100 GPUs using PyTorch 2.5 with Hugging Face's Transformers, Datasets, Accelerate, and PEFT libraries.

3.4 Results

Our experimental results demonstrate the superior performance of the PORAG+ATLAS framework across three challenging benchmarks. On the HotpotQA multi-hop question-answering dataset (Table 1), our model achieves state-of-the-art results with 65.37% EM and 78.40% F1 for answer prediction, along with 60.21% EM and 82.01% F1 for supporting fact retrieval. The joint evaluation metrics (45.29% EM and 71.32% F1) represent substantial improvements of +10.41% EM and +22.22% F1 over the RAG-base baseline. For the Gorilla API-aware code generation benchmark (Table 2), the framework achieves 76.38% accuracy while significantly reducing critical errors—5.31% hallucination and 4.98% wrong API calls—which are nearly half those of RAG-base (10.70% and 9.58%, respectively). On the biomedical PubMedQA dataset (Table 3), our model attains 78.35% accuracy and 74.56% F1, outperforming RAG-base by +17.65% accuracy and +15.26% F1. The framework generally surpasses ablation variants (PORAG-only, GRPO+ATLAS, PORAG+DRAGIN) across the three benchmarks (Tables 1–3), demonstrating both the effectiveness of ATLAS integration and PORAG's superior architecture. These comprehensive results validate that PORAG+ATLAS delivers robust improvements in retrieval precision and generation accuracy while significantly reducing critical errors across diverse domains, including multi-hop QA, code generation, and biomedical question answering.

4 Conclusion

We present an integrated framework that enhances RAG through the synergistic combination of Policy-Optimized Retrieval-Augmented Generation (PORAG) and Adaptive Token-Layer Attention Scoring (ATLAS). Our approach demonstrates significant improvements in factual accuracy, reduction of hallucinations, and computational efficiency across diverse benchmarks. Extensive experiments and ablation studies confirm that the framework successfully balances retrieval fidelity with generation quality while maintaining low computational overhead. As a flexible and scalable solution compatible with any Transformer-based language model, our method represents a substantial advancement for knowledge-intensive NLP tasks.

References

[1] Souradip Chakraborty, Sujay Bhatt, Udari Madhushani Sehwag, Soumya Suvra Ghosal, Jiahao Qiu, Mengdi Wang, Dinesh Manocha, Furong Huang, Alec Koppel,

Table 1: HotpotQA Performance (Higher is better for all metrics)

Model	Answer Prediction		Supporting Facts		Joint	
	EM	F1	EM	F1	EM	F1
PORAG+ATLAS (Proposed)	**65.37**	**78.40**	**60.21**	**82.01**	**45.29**	**71.32**
PORAG-only	63.85	77.10	58.32	80.20	44.62	69.88
GRPO+ATLAS	63.24	76.82	58.00	79.60	44.05	69.25
PORAG+DRAGIN	62.10	76.02	57.47	79.21	43.55	68.94
RAG+ATLAS	60.70	74.95	56.25	78.02	42.45	67.22
RAFT+ATLAS	59.85	73.88	55.14	77.15	41.75	66.30
RAG-base	52.10	64.02	44.21	61.28	34.88	49.10

Table 2: Gorilla Performance on Code Generation (Higher Accuracy and Lower Error are better)

Model	Overall Accuracy (%)	Hallucination Error (%)	Wrong API Call Error (%)
PORAG+ATLAS (Proposed)	**76.38**	**5.31**	**4.98**
PORAG-only	70.12	7.38	7.89
GRPO+ATLAS	73.26	6.52	5.83
PORAG+DRAGIN	71.96	6.84	5.92
RAG+ATLAS	70.84	6.40	5.85
RAFT+ATLAS	71.70	7.55	7.00
RAG-base	62.12	10.70	9.58

Table 3: PubMedQA Performance (Higher is better)

Model	Accuracy (%)	F1 Score (%)
PORAG+ATLAS (Proposed)	**78.35**	**74.56**
PORAG-only	75.25	72.83
GRPO+ATLAS	76.80	75.42
PORAG+DRAGIN	75.60	74.30
RAG+ATLAS	74.40	72.90
RAFT+ATLAS	73.20	71.60
RAG-base	60.70	59.30

and Sumitra Ganesh. [n. d.]. Collab: Controlled Decoding using Mixture of Agents for LLM Alignment. In *The Thirteenth International Conference on Learning Representations*.

[2] Brian J Chan, Chao-Ting Chen, Jui-Hung Cheng, and Hen-Hsen Huang. 2024. Don't Do RAG: When Cache-Augmented Generation is All You Need for Knowledge Tasks. *arXiv preprint arXiv:2412.15605* (2024).

[3] Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318* (2023).

[4] Guanzheng Chen, Qilong Feng, Jinjie Ni, Xin Li, and Michael Qizhe Shieh. 2025. Long-Context Inference with Retrieval-Augmented Speculative Decoding. arXiv:2502.20330 [cs.CL] https://arxiv.org/abs/2502.20330

[5] Jiefeng Chen, Jie Ren, Xinyun Chen, Chengrun Yang, Ruoxi Sun, and Sercan Ö Arık. 2025. SETS: Leveraging Self-Verification and Self-Correction for Improved Test-Time Scaling. *arXiv preprint arXiv:2501.19306* (2025).

[6] Yanxi Chen, Xuchen Pan, Yaliang Li, Bolin Ding, and Jingren Zhou. 2024. A simple and provable scaling law for the test-time compute of large language models. *arXiv preprint arXiv:2411.19477* (2024).

[7] Zhenfang Chen, Delin Chen, Rui Sun, Wenjun Liu, and Chuang Gan. 2025. Scaling Autonomous Agents via Automatic Reward Modeling And Planning. *arXiv preprint arXiv:2502.12130* (2025).

[8] Yinlam Chow, Guy Tennenholtz, Izzeddin Gur, Vincent Zhuang, Bo Dai, Sridhar Thiagarajan, Craig Boutilier, Rishabh Agarwal, Aviral Kumar, and Aleksandra Faust. 2024. Inference-aware fine-tuning for best-of-n sampling in large language models. *arXiv preprint arXiv:2412.15287* (2024).

[9] Giulio Corallo and Paolo Papotti. 2024. FINCH: Prompt-guided Key-Value Cache Compression for Large Language Models. *Transactions of the Association for Computational Linguistics* 12 (2024), 1517–1532.

[10] Tri Dao. 2023. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691* (2023).

[11] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems* 35 (2022), 16344–16359.

[12] Souvik Das, Lifeng Jin, Linfeng Song, Haitao Mi, Baolin Peng, and Dong Yu. 2024. Entropy guided extrapolative decoding to improve factuality in large language models. *arXiv preprint arXiv:2404.09338* (2024).

[13] Alessio Devoto, Yu Zhao, Simone Scardapane, and Pasquale Minervini. 2024. A Simple and Effective L_2 Norm-Based Strategy for KV Cache Compression. *arXiv preprint arXiv:2406.11430* (2024).

[14] Xidong Feng, Ziyu Wan, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun Wang. 2023. Alphazero-like tree-search can guide large language model decoding and training. *arXiv preprint arXiv:2309.17179* (2023).

[15] Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S Kevin Zhou. 2024. Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference. *arXiv preprint arXiv:2407.11550* (2024).

[16] Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. 2024. Break the sequential dependency of llm inference using lookahead decoding. *arXiv preprint arXiv:2402.02057* (2024).

[17] Zitian Gao, Boye Niu, Xuzheng He, Haotian Xu, Hongzhang Liu, Aiwei Liu, Xuming Hu, and Lijie Wen. 2024. Interpretable contrastive monte carlo tree

search reasoning. *arXiv preprint arXiv:2410.01707* (2024).
[18] Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. 2025. Scaling up Test-Time Compute with Latent Reasoning: A Recurrent Depth Approach. *arXiv preprint arXiv:2502.05171* (2025).
[19] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948* (2025).
[20] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Sophia Shao, Kurt Keutzer, and Amir Gholami. 2025. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Advances in Neural Information Processing Systems* 37 (2025), 1270–1303.
[21] Gautier Izacard and Edouard Grave. 2020. Leveraging passage retrieval with generative models for open domain question answering. *arXiv preprint arXiv:2007.01282* (2020).
[22] Yixin Ji, Juntao Li, Hai Ye, Kaixin Wu, Jia Xu, Linjian Mo, and Min Zhang. 2025. Test-time Computing: from System-1 Thinking to System-2 Thinking. *arXiv preprint arXiv:2501.02497* (2025).
[23] Jinhao Jiang, Zhipeng Chen, Yingqian Min, Jie Chen, Xiaoxue Cheng, Jiapeng Wang, Yiru Tang, Haoxiang Sun, Jia Deng, Wayne Xin Zhao, et al. 2024. Technical report: Enhancing llm reasoning with reward-guided tree search. *arXiv preprint arXiv:2411.11694* (2024).
[24] Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William W Cohen, and Xinghua Lu. 2019. Pubmedqa: A dataset for biomedical research question answering. *arXiv preprint arXiv:1909.06146* (2019).
[25] Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*. PMLR, 19274–19286.
[26] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems* 33 (2020), 9459–9474.
[27] Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2025. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems* 37 (2025), 22947–22970.
[28] Zongyu Lin, Yao Tang, Xingcheng Yao, Da Yin, Ziniu Hu, Yizhou Sun, and Kai-Wei Chang. 2025. QLASS: Boosting Language Agent Inference via Q-Guided Stepwise Search. *arXiv preprint arXiv:2502.02584* (2025).
[29] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437* (2024).
[30] Runze Liu, Junqi Gao, Jian Zhao, Kaiyan Zhang, Xiu Li, Biqing Qi, Wanli Ouyang, and Bowen Zhou. 2025. Can 1B LLM Surpass 405B LLM? Rethinking Compute-Optimal Test-Time Scaling. *arXiv preprint arXiv:2502.06703* (2025).
[31] Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. 2023. Online speculative decoding. *arXiv preprint arXiv:2310.07177* (2023).
[32] Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393* (2025).
[33] Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2024. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems* 37 (2024), 126544–126565.
[34] Zhenting Qi, Mingyuan Ma, Jiahang Xu, Li Lyna Zhang, Fan Yang, and Mao Yang. 2024. Mutual reasoning makes smaller llms stronger problem-solvers. *arXiv preprint arXiv:2408.06195* (2024).
[35] Hongjin Qian, Peitian Zhang, Zheng Liu, Kelong Mao, and Zhicheng Dou. 2024. Memorag: Moving towards next-gen rag via memory-inspired knowledge discovery. *arXiv preprint arXiv:2409.05591* (2024).
[36] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300* (2024).
[37] Toby Simonds. 2025. Entropy Adaptive Decoding: Dynamic Model Switching for Efficient Inference. *arXiv preprint arXiv:2502.06833* (2025).
[38] W Su, Y Tang, Q Ai, Z Wu, and Y Liu. [n. d.]. DRAGIN: Dynamic Retrieval Augmented Generation based on the Real-time Information Needs of Large Language Models. arXiv 2024. *arXiv preprint arXiv:2403.10081* ([n. d.]).
[39] Weihang Su, Yichen Tang, Qingyao Ai, Junxi Yan, Changyue Wang, Hongning Wang, Ziyi Ye, Yujia Zhou, and Yiqun Liu. 2025. Parametric Retrieval Augmented Generation. *arXiv preprint arXiv:2501.15915* (2025).
[40] Xinyu Tang, Xiaolei Wang, Wayne Xin Zhao, and Ji-Rong Wen. 2024. Dawn-icl: Strategic planning of problem-solving trajectories for zero-shot in-context learning. *arXiv preprint arXiv:2410.20215* (2024).
[41] Evan Wang, Federico Cassano, Catherine Wu, Yunfeng Bai, Will Song, Vaskar Nath, Ziwen Han, Sean Hendryx, Summer Yue, and Hugh Zhang. 2024. Planning in natural language improves llm search for code generation. *arXiv preprint arXiv:2409.03733* (2024).
[42] Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. 2024. Mixture-of-agents enhances large language model capabilities. *arXiv preprint arXiv:2406.04692* (2024).
[43] Liang Wang, Haonan Chen, Nan Yang, Xiaolong Huang, Zhicheng Dou, and Furu Wei. 2025. Chain-of-Retrieval Augmented Generation. *arXiv preprint arXiv:2501.14342* (2025).
[44] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171* (2022).
[45] Xuezhi Wang and Denny Zhou. 2024. Chain-of-thought reasoning without prompting. *arXiv preprint arXiv:2402.10200* (2024).
[46] Zilong Wang, Zifeng Wang, Long Le, Huaixiu Steven Zheng, Swaroop Mishra, Vincent Perot, Yuwei Zhang, Anush Mattapalli, Ankur Taly, Jingbo Shang, et al. 2024. Speculative rag: Enhancing retrieval augmented generation through drafting. *arXiv preprint arXiv:2407.08223* (2024).
[47] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.
[48] Jinyang Wu, Mingkuan Feng, Shuai Zhang, Ruihan Jin, Feihu Che, Zengqi Wen, and Jianhua Tao. 2025. Boosting Multimodal Reasoning with MCTS-Automated Structured Thinking. *arXiv preprint arXiv:2502.02339* (2025).
[49] Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. 2024. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. *arXiv preprint arXiv:2410.10819* (2024).
[50] Yuxi Xie, Anirudh Goyal, Wenyue Zheng, Min-Yen Kan, Timothy P Lillicrap, Kenji Kawaguchi, and Michael Shieh. 2024. Monte carlo tree search boosts reasoning via iterative preference learning. *arXiv preprint arXiv:2405.00451* (2024).
[51] Yuhui Xu, Zhanming Jie, Hanze Dong, Lei Wang, Xudong Lu, Aojun Zhou, Amrita Saha, Caiming Xiong, and Doyen Sahoo. 2024. Think: Thinner key cache by query-driven pruning. *arXiv preprint arXiv:2407.21018* (2024).
[52] Minghao Yan, Saurabh Agarwal, and Shivaram Venkataraman. 2024. Decoding speculative decoding. *arXiv preprint arXiv:2402.01528* (2024).
[53] Jingbo Yang, Bairu Hou, Wei Wei, Yujia Bao, and Shiyu Chang. 2025. KVLink: Accelerating Large Language Models via Efficient KV Cache Reuse. *arXiv preprint arXiv:2502.16002* (2025).
[54] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600* (2018).
[55] Jaesik Yoon, Hyeonseo Cho, Doojin Baek, Yoshua Bengio, and Sungjin Ahn. 2025. Monte Carlo Tree Diffusion for System 2 Planning. *arXiv preprint arXiv:2502.07202* (2025).
[56] Zhouliang Yu, Yuhuan Yuan, Tim Z Xiao, Fuxiang Frank Xia, Jie Fu, Ge Zhang, Ge Lin, and Weiyang Liu. 2025. Generating Symbolic World Models via Test-time Scaling of Large Language Models. *arXiv preprint arXiv:2502.04728* (2025).
[57] Zhiyuan Zeng, Qinyuan Cheng, Zhangyue Yin, Yunhua Zhou, and Xipeng Qiu. 2025. Revisiting the Test-Time Scaling of o1-like Models: Do they Truly Possess Test-Time Scaling Capabilities? *arXiv preprint arXiv:2502.12215* (2025).
[58] Di Zhang, Xiaoshui Huang, Dongzhan Zhou, Yuqiang Li, and Wanli Ouyang. 2024. Accessing gpt-4 level mathematical olympiad solutions via monte carlo tree self-refine with llama-3 8b. *arXiv preprint arXiv:2406.07394* (2024).
[59] Shimao Zhang, Yu Bao, and Shujian Huang. 2024. EDT: Improving Large Language Models' Generation by Entropy-based Dynamic Temperature Sampling. *arXiv preprint arXiv:2403.14541* (2024).
[60] Tianjun Zhang, Shishir G Patil, Naman Jain, Sheng Shen, Matei Zaharia, Ion Stoica, and Joseph E Gonzalez. 2024. Raft: Adapting language model to domain specific rag. In *First Conference on Language Modeling*.
[61] Xuan Zhang, Cunxiao Du, Chao Du, Tianyu Pang, Wei Gao, and Min Lin. 2024. Simlayerkv: A simple framework for layer-level KV cache reduction. *arXiv preprint arXiv:2410.13846* (2024).
[62] Zhihan Zhang, Tao Ge, Zhenwen Liang, Wenhao Yu, Dian Yu, Mengzhao Jia, Dong Yu, and Meng Jiang. 2024. Learn beyond the answer: Training language models with reflection for mathematical reasoning. *arXiv preprint arXiv:2406.12050* (2024).
[63] Yu Zhao, Huifeng Yin, Bo Zeng, Hao Wang, Tianqi Shi, Chenyang Lyu, Longyue Wang, Weihua Luo, and Kaifu Zhang. 2024. Marco-o1: Towards open reasoning models for open-ended solutions. *arXiv preprint arXiv:2411.14405* (2024).

A Technical Appendix

A.1 Ablation Studies

To rigorously validate our framework, we conduct ablation studies examining both PORAG and ATLAS components. (1). For Policy-Optimized RAG (PORAG), we first evaluate the dual reward mechanism by comparing the full model (PORAG-Full) with default fidelity/quality weights ($\alpha = 0.7$, $\beta = 0.3$) against three variants: (a) PORAG-NF, which removes the fidelity reward by setting $\alpha = 0$, $\beta = 1$; (b) PORAG-NQ, which disables the quality reward with $\alpha = 1$, $\beta = 0$; and (c) PORAG-α/β-Var, which tests alternative weightings such as $\alpha = \beta = 0.5$ to analyze trade-offs. (2). We then assess optimization components of PORAG by (a) replacing Group Relative Policy Optimization (GRPO) with standard PPO in the PORAG-PPO variant, (b) varying group sizes with $G \in \{2, 4\}$ using $G = 4$ as the default, and (c) experimenting with different KL divergence regularization strengths, specifically $\omega_2 \in \{0.05, 0.1, 0.2\}$, to investigate its role in preserving model stability and preventing catastrophic forgetting using $\omega_2 = 0.1$ as the default. (3). For Adaptive Token-Layer Attention Scoring (ATLAS), we ablate the Multi-Layer Attention Gradient (MLAG) mechanism by comparing the full method (ATLAS-Full) with default layer weights $\eta_j = j/(L-1)$, scaling factor $\alpha_0 = 0.8$, and decay $\lambda = 4$, against (a) a single-layer variant (ATLAS-Single) to isolate the impact of depth-aware gradients, and (b) modified layer weightings in which higher layers ($j > 2L/3$) are weighted three times more heavily based on their task-relevant abstraction capabilities. (4). To analyze the impact of query formulation, we compare ATLAS-Full, which uses dynamic token selection with a default top-$k = 6$ and attention-representation balance of $\beta = 0.7$, against (a) a fixed-window baseline (ATLAS-FixedLRP) that does not rely on attention dynamics for token selection. (5). We further study the role of the semantic filter s_i by removing it entirely in the ATLAS-noSF variant, which disables the exclusion of stopwords, punctuation, and numeric tokens to assess its effect on retrieval precision. (6). Lastly, we examine the impact of dynamic retrieval scaling by comparing the default exponential schedule, defined as $\alpha = 0.8 \cdot e^{-4C_{\text{current}}/C_{\max}}$ with $C_{\max} = 90\%$ of VRAM usage, against a static variant (ATLAS-Static) that uses a constant sensitivity setting $\alpha \equiv 1.0$. These ablations isolate each individual contribution to the full system and confirm that both PORAG and ATLAS components play critical and complementary roles in enhancing retrieval-augmented generation. The ablation studies (Tables 4-6) demonstrate that both PORAG and ATLAS components contribute significantly to the framework's performance. The complete PORAG+ATLAS framework achieves optimal balance across all components, with the ablation studies confirming that each design choice contributes meaningfully to the final performance. In addition to the comprehensive ablation studies conducted on the PORAG and ATLAS components, we investigate the sensitivity of the MLAG retrieval trigger mechanism in ATLAS (see Table 7), focusing on two critical parameters: the baseline scaling factor (α_0) and the generation probability threshold (τ_p). The parameter α_0 (varied between 0.7–1.0) controls retrieval sensitivity, with higher values increasing retrieval frequency under low computational load, while τ_p (tested at 0.3, 0.5, and 0.7) acts as a confidence threshold—lower values trigger retrieval more readily under model uncertainty, whereas higher values risk missed retrievals. Our experiments on HotpotQA systematically vary these parameters while holding the core PORAG+ATLAS framework constant. Analyzing the results reveals that the combination of $\alpha_0 = 0.8$ and $\tau_p = 0.5$ provides the optimal balance, yielding the best performance across all reported metrics (Answer EM/F1, Fact EM/F1, Joint EM/F1). $\tau_p = 0.5$ effectively balances retrieval timing, triggering interventions when the model's token-generation confidence falls below this threshold, while $\alpha_0 = 0.8$ appropriately modulates the base retrieval sensitivity. These findings demonstrate that fine-tuning these specific trigger parameters maximizes retrieval efficacy—improving answer accuracy and supporting fact recall—while rigorously managing computational overhead. The results underscore the importance of ATLAS's adaptive retrieval mechanism, where precision-tuned thresholds (τ_p) and dynamic scaling (α_0) collectively mitigate unnecessary retrievals without sacrificing factual grounding.

A.2 Additional Experiments

Our experiments on benchmark datasets—HotpotQA, Gorilla, and PubMedQA—using various parameter variants of Qwen2.5 (0.5B, 1.5B, and 3B) and Llama 3.2 (1B and 3B) demonstrate that our integrated PORAG+ATLAS framework consistently outperforms the baseline RAG approach. For HotpotQA (Table 8), PORAG+ATLAS yields substantial improvements, with Joint EM gains reaching up to +10.4 points (Qwen2.5-3B: 45.29% vs 34.88%) and Joint F1 gains exceeding +22.2 points (Qwen2.5-3B: 71.32% vs 49.10%) compared to the baseline models. In the Gorilla code generation task (Table 9), our method achieves higher overall accuracy across all variants (e.g., +14.3 points for Qwen2.5-3B, reaching 76.38%) while significantly reducing both hallucination and API errors (e.g., for Qwen2.5-3B, hallucination reduced from 10.70% to 5.31% and API errors decreased from 9.58% to 4.98%). Likewise, on PubMedQA (Table 10), PORAG+ATLAS consistently delivers markedly improved accuracy and F1 scores, showcasing substantial gains such as +17.6 points for accuracy (Qwen2.5-3B: 78.35% vs 60.71%) and +15.3 points for F1 score (Qwen2.5-3B: 74.56% vs 59.30%). These results validate that our framework robustly enhances retrieval fidelity and generation quality across different LLM sizes and architectures.

A.3 Policy-Optimized Retrieval-Augmented Generation (PORAG)

RAG techniques present unique optimization challenges that Retrieval-Augmented Fine-Tuning (RAFT) often struggles to fully address. PORAG offers a principled solution rooted in Group Relative Policy Optimization (GRPO) by reformulating the optimization problem through a group-based relative advantage framework. Unlike RAFT, which optimizes for log-likelihood of reference outputs, PORAG enables direct optimization for retrieval quality, contextual relevance, and generation coherence through dual reward modeling. In this work, we present a comprehensive mathematical formulation of PORAG, with theoretical justifications and analytical insights. In the traditional RAG framework, the policy model $\pi_\theta(y|x, d)$ generates outputs y conditioned on the input query x and retrieved documents d. The process is formalized as:

$$\pi_\theta(y|x, d) = \prod_{i=1}^{|y|} \pi_\theta(y_i|x, d, y_{<i})$$

Table 4: HotpotQA Ablation Results (Higher is better)

Variant	Ans EM	Ans F1	Fact EM	Fact F1	Joint EM	Joint F1
PORAG+ATLAS (Proposed)	**65.37**	**78.40**	**60.21**	**82.01**	**45.29**	**71.32**
PORAG Reward Variants						
PORAG-NF ($\alpha = 0, \beta = 1$)	58.23	72.54	53.17	75.03	39.52	65.24
PORAG-NQ ($\alpha = 1, \beta = 0$)	57.85	72.06	52.73	74.62	38.91	64.72
PORAG-α/β-Var (0.5/0.5)	62.03	75.85	57.64	79.07	43.22	68.04
PORAG Optimization Variants						
PORAG-PPO (vs GRPO)	60.04	74.13	55.82	77.53	41.52	66.31
PORAG-G2 (Group Size=2)	63.42	76.91	58.35	80.42	44.12	69.53
PORAG-KL-0.05 ($\omega_2 = 0.05$)	63.24	76.82	58.00	79.60	44.05	69.25
PORAG-K-L0.2 ($\omega_2 = 0.2$)	63.91	77.30	58.83	80.71	44.83	70.18
ATLAS Variants						
ATLAS-Single (No MLAG)	63.12	76.23	58.04	79.32	43.83	68.72
ATLAS-FixedLRP (Static Tokens)	61.05	75.43	56.24	78.06	42.03	67.05
ATLAS-noSF (No Semantic Filter)	62.53	76.85	57.83	79.07	43.42	68.23
ATLAS-Static ($\alpha \equiv 1.0$)	60.92	75.03	56.53	78.24	42.32	67.34
ATLAS-Layer3x (High Layer Focus)	63.85	77.12	58.92	80.35	44.62	69.87

Table 5: Gorilla Ablation Results (Higher Accuracy and Lower Errors are better)

Variant	Overall Accuracy (%)	Hallucination Error (%)	Wrong API Error (%)
PORAG+ATLAS (Proposed)	**76.38**	**5.31**	**4.98**
PORAG Reward Variants			
PORAG-NF ($\alpha = 0, \beta = 1$)	71.83	6.91	5.27
PORAG-NQ ($\alpha = 1, \beta = 0$)	70.36	6.74	6.59
PORAG-α/β-Var (0.5/0.5)	74.92	5.14	5.43
PORAG Optimization Variants			
PORAG-PPO (vs GRPO)	73.48	5.23	5.88
PORAG-G2 (Group Size=2)	75.12	5.42	5.12
PORAG-KL-0.05 ($\omega_2 = 0.05$)	74.63	5.67	5.34
PORAG-KL-0.2 ($\omega_2 = 0.2$)	75.84	5.38	5.07
ATLAS Variants			
ATLAS-Single (No MLAG)	72.37	6.68	5.95
ATLAS-FixedLRP (Static Tokens)	71.29	6.82	5.31
ATLAS-noSF (No Semantic Filter)	73.46	5.95	5.78
ATLAS-Static ($\alpha \equiv 1.0$)	72.63	6.82	5.19
ATLAS-Layer3x (High Layer Focus)	75.29	5.41	5.03

where $\pi_\theta(y|x, d)$ represents the probability distribution over the generated outputs y, conditioned on the input query x, retrieved documents d, and previously generated tokens $y_{<i}$. Here, x denotes the input query, $d = \{d_1, d_2, ..., d_k\}$ represents the set of retrieved documents, y_i is the token at position i, and $y_{<i}$ comprises all previously generated tokens. The parameter θ corresponds to the frozen weights of the language model, which remain unchanged during inference. In RAFT, the training objective optimizes the pretrained language model by maximizing the likelihood of reference outputs y^* while incorporating both relevant ("oracle") and irrelevant ("distractor") documents. Since RAFT employs Low-Rank Adaptation (LoRA[21, 26]), only a subset of trainable parameters, denoted as γ, is updated, while the pre-trained language model parameters θ remain frozen. The RAFT loss function is defined as:

$$\mathcal{L}_{\text{RAFT}}(\gamma) = -\mathbb{E}_{(x, d_{\text{oracle}}, d_{\text{distractor}}, y^*) \sim \mathcal{D}} \left[\log \pi_{\theta,\gamma}(y^*|x, d_{\text{oracle}}, d_{\text{distractor}})\right]$$

Table 6: PubMedQA Ablation Results (Higher is better)

Variant	Accuracy (%)	F1 Score (%)
PORAG+ATLAS (Proposed)	**78.35**	**80.56**
PORAG Reward Variants		
PORAG-NF ($\alpha = 0, \beta = 1$)	72.57	74.83
PORAG-NQ ($\alpha = 1, \beta = 0$)	71.92	73.14
PORAG-α/β-Var (0.5/0.5)	75.63	77.29
PORAG Optimization Variants		
PORAG-PPO (vs GRPO)	73.25	75.68
PORAG-G2 (Group Size=2)	76.42	78.93
PORAG-KL-0.05 ($\omega_2 = 0.05$)	76.85	79.12
PORAG-KL-0.2 ($\omega_2 = 0.2$)	77.03	79.84
ATLAS Variants		
ATLAS-Single (No MLAG)	74.81	76.47
ATLAS-FixedLRP (Static Tokens)	72.19	74.36
ATLAS-noSF (No Semantic Filter)	75.29	77.91
ATLAS-Static ($\alpha \equiv 1.0$)	73.94	75.52
ATLAS-Layer3x (High Layer Focus)	76.87	79.25

Table 7: Ablation Study on Retrieval Trigger Sensitivity in ATLAS

α_0	τ_p	Answer EM (%)	Answer F1 (%)	Fact EM (%)	Fact F1 (%)	Joint EM (%)	Joint F1 (%)
0.7	0.3	58.24	70.15	53.12	66.23	50.35	62.41
0.7	0.5	59.53	71.37	54.82	67.91	52.14	64.28
0.7	0.7	57.16	68.93	52.07	65.04	49.28	61.17
0.8	0.3	60.82	72.64	55.93	68.75	53.26	65.37
0.8	0.5	**65.37**	**78.40**	**60.21**	**82.01**	**45.29**	**71.32**
0.8	0.7	60.24	73.18	55.36	68.29	52.83	65.09
0.9	0.3	61.57	74.26	56.78	70.15	54.37	66.58
0.9	0.5	62.89	75.94	57.93	71.34	55.26	67.84
0.9	0.7	61.08	74.83	56.24	69.53	53.76	66.18
1.0	0.3	59.73	72.84	54.92	68.93	52.48	64.73
1.0	0.5	61.28	74.53	56.34	70.28	53.94	66.34
1.0	0.7	60.17	73.69	55.18	69.07	52.68	65.09

Table 8: HotpotQA Performance Comparison (Joint EM/F1; Higher is better)

LLM Variant	Baseline RAG		PORAG+ATLAS	
	Joint EM (%)	Joint F1 (%)	Joint EM (%)	Joint F1 (%)
Qwen2.5-0.5B	25.73	38.42	30.88	43.17
Qwen2.5-1.5B	28.91	41.35	33.64	46.29
Qwen2.5-3B	**34.88**	**49.10**	**45.29**	**71.32**
Llama 3.2-1B	27.56	40.18	32.07	45.83
Llama 3.2-3B	30.24	44.76	38.59	52.41

where x is the input query, d_{oracle} and $d_{\text{distractor}}$ represent the retrieved relevant and irrelevant documents, respectively, and y^* is the reference output. The training dataset $\mathcal{D}$ consists of tuples $(x, d_{\text{oracle}}, d_{\text{distractor}}, y^*)$. The model assigns probability, $\pi_{\theta,\gamma}(y^*|x, d_{\text{oracle}}, d_{\text{distractor}})$ to the correct output, where θ represents the frozen pre-trained language model parameters, and γ represents the trainable parameters of the base language model, specifically Quantized Low-Rank Adaptation (QLoRA) adapters.

Table 9: Gorilla Performance Comparison (Accuracy, Hallucination, API Errors)

LLM Variant	Baseline RAG			PORAG+ATLAS		
	Accuracy (%)	Hallucination (%)	API Error (%)	Accuracy (%)	Hallucination (%)	API Error (%)
Qwen2.5-0.5B	50.62	15.73	14.28	58.39	12.45	11.67
Qwen2.5-1.5B	54.17	13.82	12.91	62.84	10.53	9.24
Qwen2.5-3B	**62.12**	**10.70**	**9.58**	**76.38**	**5.31**	**4.98**
Llama 3.2-1B	52.48	14.36	13.75	60.92	11.83	10.47
Llama 3.2-3B	56.33	12.67	11.89	65.71	9.62	8.53

Table 10: PubMedQA Performance Comparison (Accuracy and F1; Higher is better)

LLM Variant	Baseline RAG		PORAG+ATLAS	
	Accuracy (%)	F1 (%)	Accuracy (%)	F1 (%)
Qwen2.5-0.5B	48.35	50.82	55.67	57.93
Qwen2.5-1.5B	52.91	54.47	60.38	62.14
Qwen2.5-3B	**60.71**	**59.30**	**78.35**	**74.56**
Llama 3.2-1B	50.26	52.73	58.49	60.85
Llama 3.2-3B	54.88	56.42	63.17	65.39

These are small, trainable low-rank matrices added to the frozen pre-trained language model (θ) to govern output generation conditioned on the input and retrieved documents. QLoRA focuses on adapting key layers like attention query/value projections and feed-forward networks. This approach enables efficient fine-tuning by modifying only a small subset of weights, ensuring that the model learns to effectively distinguish relevant information from distractors while leveraging retrieval-augmented generation for adaptation. However, RAFT has several limitations. It cannot differentiate between high- and low-quality retrievals, assumes perfect reference outputs that fully leverage retrieved information, and does not account for multiple valid generation strategies within the same retrieval context. Additionally, it fails to optimize nuanced qualities such as faithfulness to retrieved information. In contrast, PORAG addresses these limitations by enabling direct optimization for multiple quality dimensions simultaneously. Our implementation employs two specialized reward heads—lightweight, parameterized functions attached to the base model's hidden states—calibrated for RAG-specific quality dimensions: a Retrieval-Fidelity Reward $R_{\text{fidelity}}(x, d, y^*; \phi_1)$, which evaluates how faithfully the generated response incorporates and accurately reflects the retrieved information, and a Response-Quality Reward $R_{\text{quality}}(x, d, y^*; \phi_2)$, which evaluates the overall quality, coherence, and helpfulness of the response beyond mere factual accuracy. Here, $\phi = \{\phi_1, \phi_2\}$ represent the trainable reward head parameters. The two reward heads—ϕ_1 for retrieval fidelity and ϕ_2 for response quality—are integrated into the neural network architecture at the final layer, operating on the hidden representations produced by the base model to compute scalar rewards. Parameters ϕ_1 and ϕ_2 (typically implemented via trainable standard linear layers with an intermediate tanh activation) are specifically optimized to evaluate how well the generated response meets the desired qualities (i.e., factual alignment with the retrieved documents and overall quality). The reward heads are trained in conjunction with the base model, facilitating end-to-end optimization of both the generation and the reward function estimation. Consequently, the generation policy is directly informed by these dynamically learned reward signals. This co-adaptation mechanism results in more precise reward evaluations, enhanced training stability, and ultimately, superior performance in RAG. To effectively optimize the RAG context for multiple objectives, we decompose the utility function into orthogonal components, each capturing distinct quality dimensions. This allows the reward heads to focus on specific aspects of generation quality. The utility function is defined as:

$$\mathcal{U}(x, d, y^*) = \alpha \cdot \mathcal{U}_{\text{fidelity}}(x, d, y^*) + \beta \cdot \mathcal{U}_{\text{quality}}(x, y^*) + \lambda \cdot \mathcal{U}_{\text{interaction}}(x, d, y^*)$$

where: $\mathcal{U}_{\text{fidelity}}(x, d, y^*)$ measures the accuracy of the generated text in reflecting the retrieved documents, rewarding correct factual content and penalizing hallucinations; $\mathcal{U}_{\text{quality}}(x, y^*)$ evaluates the inherent quality of the generation (coherence, fluency, relevance to the query), independent of the retrieved content; and $\mathcal{U}_{\text{interaction}}(x, d, y^*)$ captures the synergistic effects between fidelity and quality. Our dual reward heads approximate this decomposition:

$$R_{\text{fidelity}}(x, d, y^*; \phi_1) \approx \mathcal{U}_{\text{fidelity}}(x, d, y^*)$$
$$R_{\text{quality}}(x, d, y^*; \phi_2) \approx \mathcal{U}_{\text{quality}}(x, y^*) + \frac{\lambda}{\beta} \cdot \mathcal{U}_{\text{interaction}}(x, d, y^*)$$

The reward heads compute scalar rewards from a vector representation derived from the hidden states of the base model through parameterized transformation functions:

$$R_{\text{fidelity}}(x, d, y^*; \phi_1) = f_{\phi_1}(h(x, d, y^*))$$
$$R_{\text{quality}}(x, d, y^*; \phi_2) = g_{\phi_2}(h(x, d, y^*))$$

where $h(x, d, y^*) \in \mathbb{R}^d$ is a vector derived from the base language model's hidden states. Transformer models output a hidden state matrix $\mathbb{R}^{n \times d}$ (where n is sequence length, d is hidden dimension). h is obtained by aggregating this matrix, e.g., using the last token's state or pooling. The reward heads $R_{\text{fidelity}} = f_{\phi_1}(h)$ and $R_{\text{quality}} = f_{\phi_2}(h)$ are both multi-layer perceptrons with the form:

$$f_{\phi_i}(h) = W_2^{\phi_i} \cdot \tanh(W_1^{\phi_i} \cdot h + b_1^{\phi_i}) + b_2^{\phi_i}$$

where for $i \in \{1, 2\}$, $W_1^{\phi_i} \in \mathbb{R}^{d \times d}$, $W_2^{\phi_i} \in \mathbb{R}^{d \times 1}$, $b_1^{\phi_i} \in \mathbb{R}^d$, and $b_2^{\phi_i} \in \mathbb{R}$ are the parameters for reward head i. We calculate the combined reward by balancing the competing objectives of retrieval fidelity and response quality. Specifically, we aggregate quality and fidelity rewards as follows:

$$\begin{aligned} R_{\text{comb}}(x, d, y^*) = \alpha \cdot R_{\text{fidelity}}(x, d, y^*; \phi_1) \\ + \beta \cdot R_{\text{quality}}(x, d, y^*; \phi_2) \end{aligned}$$

This weighting scheme ($\alpha = 0.7$ and $\beta = 0.3$ in our implementation) balances the competing objectives of retrieval fidelity and response quality. The theoretical justification for this weighting comes from multi-objective reinforcement learning theory, where the Pareto frontier of optimal policies can be explored through different weightings of reward components. Unlike RAFT, which implicitly weights these objectives based on the training data distribution alone, PORAG allows explicit control over this trade-off, enabling adaptation to different deployment scenarios and user preferences. The combined rewards are normalized and scaled using robust statistical principles:

$$R_{\text{final}}(x, d, y^*) = \text{clip}(R_{\text{comb}}(x, d, y^*), -c_1, c_1) \cdot \gamma_{\text{scale}}$$

where γ_{scale} is the reward scaling factor, and $c_1 = 10.0$ is the clipping threshold. The clipping operation is a form of Winsorization, a statistical technique that reduces the impact of outliers while preserving the ordinal relationships between rewards. We will now discuss Group-based Advantage Estimation for RAG. Given an input query x and retrieved documents d, we generate a batch of G outputs, denoted by $\{y^{(1)}, y^{(2)}, \ldots, y^{(G)}\}$, using the current policy π_γ. This batch of outputs represents a single group of alternatives. Within this group, we compute robust statistical estimators based on the final reward $R_{\text{final}}(x, d, y^{(i)})$, which represents the overall reward for the i-th output $y^{(i)}$ within that group, given the input query x and retrieved documents d:

$$\mu_R(x, d) = \frac{1}{G} \sum_{i=1}^{G} R_{\text{final}}(x, d, y^{(i)})$$

$$\sigma_R^2(x, d) = \frac{1}{G} \sum_{i=1}^{G} \left(R_{\text{final}}(x, d, y^{(i)}) - \mu_R(x, d) \right)^2$$

$$\sigma_R(x, d) = \max\left(\sqrt{\sigma_R^2(x, d) + \epsilon}, \sigma_{\min} \right)$$

where $\mu_R(x, d)$ is the mean reward calculated within the group, $\sigma_R^2(x, d)$ is the variance of the rewards calculated within the group, and $\sigma_R(x, d)$ is the standard deviation of the rewards calculated within the group, clipped below by a minimum value $\sigma_{\min} = 0.1$ to ensure numerical stability. The clipping prevents overly aggressive updates when reward variation is small, which is particularly important in RAG scenarios where retrieved documents might lead to very similar generations within the group. The group-relative advantage for each output $y^{(i)}$ is then calculated as:

$$\hat{A}_i = \frac{R_{\text{final}}(x, d, y^{(i)}) - \mu_R(x, d)}{\sigma_R(x, d)}$$

where $\hat{A}_i$ represents the advantage of the i-th generated output relative to the other outputs within its group. We will now discuss the GRPO objective function for RAG settings. For each token $y_j^{(i)}$ in the RAG output $y^{(i)}$, we compute the probability ratio:

$$r_j(\gamma) = \frac{\pi(y_j^{(i)} | x, d, y_{<j}^{(i)})}{\pi_{\text{old}}(y_j^{(i)} | x, d, y_{<j}^{(i)})}$$

where the ratio $r_j(\gamma)$ quantifies the change in token probability under the current policy relative to the policy that generated the sample, accounting for both the query and retrieved document context. The clipped surrogate objective with a policy constraint for RAG is:

$$L_{\text{clip}}(\gamma) = \frac{1}{G} \sum_{i=1}^{G} \frac{1}{|y^{(i)}|} \sum_{j=1}^{|y^{(i)}|} \min\left(r_j(\gamma) \hat{A}_i, \text{clip}(r_j(\gamma), 1 - \epsilon, 1 + \epsilon) \hat{A}_i \right)$$

The clipping mechanism, with the parameter $\epsilon = 0.2$, serves as a trust region constraint that prevents excessively large policy updates; this is critical in RAG systems, where small changes in the probability distribution can lead to dramatically different retrieval utilization patterns. The KL divergence term prevents the policy from straying too far from the reference model:

$$D_{\text{KL}}(\pi || \pi_{\text{ref}}) = \mathbb{E}_{x, d, y \sim \pi_\gamma} \left[\sum_{i=1}^{|y|} \text{KL}(\pi_{\text{ref}}(\cdot | x, d, y_{<i}) || \pi_\gamma(\cdot | x, d, y_{<i})) \right]$$

Here, π_{ref} represents the reference policy, specifically the policy from the previous iteration of training, denoted as $\pi_{\gamma_{\text{old}}}$, where γ_{old} are the policy parameters before the current update. Using the KL divergence with respect to the previous policy stabilizes training by preventing drastic changes in the policy distribution in each update step. In the RAG context, this regularization term serves a critical function: it preserves the base knowledge encoded in the model while allowing for targeted improvements in retrieval utilization. Without this constraint, aggressive optimization toward retrieval-grounded responses might cause the model to forget its pre-trained knowledge. Using the unbiased estimator:

$$D_{\text{KL}}(\pi_\gamma || \pi_{\text{ref}}) = \mathbb{E}_{x, d, y \sim \pi_\gamma} \left[\frac{\pi_{\text{ref}}(y | x, d)}{\pi_\gamma(y | x, d)} - \log \frac{\pi_{\text{ref}}(y | x, d)}{\pi_\gamma(y | x, d)} - 1 \right]$$

The complete GRPO objective for RAG optimization is:

$$J_{\text{GRPO-RAG}}(\gamma) = \omega_1 \cdot L_{\text{clip}}(\gamma) - \omega_2 \cdot D_{\text{KL}}(\pi_\gamma || \pi_{\text{ref}})$$

where $L_{\text{clip}}(\gamma)$ is the clipped surrogate objective that measures the policy improvement using the relative advantage estimates, and $D_{\text{KL}}(\pi_\gamma || \pi_{\text{ref}})$ is the KL divergence between the current policy π_γ and the reference policy π_{ref}, acting as a regularizer. The weighting coefficients $\omega_1 = 100.0$ and $\omega_2 = 0.1$ balance policy improvement and divergence regularization; this balance is particularly important in RAG contexts to prevent overreliance on retrieved information at the expense of the model's pre-existing knowledge. The policy parameters γ are updated to maximize the GRPO-RAG objective:

$$\gamma_{k+1} = \gamma_k + \eta_\gamma \nabla_\gamma J_{\text{GRPO-RAG}}(\gamma_k)$$

The learning rate η_γ (typically 1×10^{-6} to 5×10^{-6} for RAG optimization) controls the step size of each update. Unlike RAFT, which often uses larger learning rates, GRPO-RAG typically requires smaller steps due to the complexity of the reward landscape. To prevent instability in RAG optimization, gradients are regularized both by value and by norm:

$$\nabla_\gamma J_{\text{clipped}} = \text{clip}(\nabla_\gamma J_{\text{GRPO-RAG}}(\gamma_k), -c_{\text{value}}, c_{\text{value}})$$

$$\nabla_\gamma J_{\text{normalized}} = \frac{\nabla_\gamma J_{\text{clipped}}}{||\nabla_\gamma J_{\text{clipped}}||_2} \cdot \min(||\nabla_\gamma J_{\text{clipped}}||_2, c_{\text{norm}})$$

The clipping thresholds $c_{\text{value}} = 3.0$ and $c_{\text{norm}} = 1.0$ prevent extreme gradient values that could destabilize training; this is especially important in RAG systems where the retrieval distribution can introduce high variance in gradients. The reward model parameters are updated using gradients derived from minimizing their respective reward loss functions, $\mathcal{L}_{\text{fidelity}}$ and $\mathcal{L}_{\text{quality}}$.

$$\phi_{1,k+1} = \phi_{1,k} + \eta_R \nabla_{\phi_1} \mathcal{L}_{\text{fidelity}}(\phi_{1,k})$$

$$\phi_{2,k+1} = \phi_{2,k} + \eta_R \nabla_{\phi_2} \mathcal{L}_{\text{quality}}(\phi_{2,k})$$

The reward model learning rate η_R (typically 5×10^{-5}) is usually higher than the policy learning rate, allowing the reward models to adapt more quickly to preference signals. The reward heads are updated separately using their respective reward losses with their own learning rate η_R. The gradients from the reward loss update only these differentiable parameters and do not affect the base model's weights θ or γ, thereby producing well-calibrated, scalar reward values for accurately evaluating retrieval fidelity and response quality in RAG contexts. Training the reward heads to yield reliable scalar rewards improves advantage estimation, leading to more stable policy updates and enhanced PORAG performance in RAG context. The reward losses are divided into two components corresponding to $\mathcal{L}_{\text{fidelity}}$ and $\mathcal{L}_{\text{quality}}$: $\mathcal{L}_{\text{fidelity}}$ evaluates how well the generated output reflects the retrieved documents by measuring lexical overlap with ROUGE scores (e.g., ROUGE-1, ROUGE-2, ROUGE-L), capturing content similarity at multiple granularities, while $\mathcal{L}_{\text{quality}}$ assesses overall response quality by combining semantic evaluation—using cosine similarity between sentence embeddings of the generated text and the reference—with question-answering metrics, including Exact Match and F1 scores, to balance precision and recall. In summary, while γ directly controls the generation behavior of the base model, ϕ is dedicated to assessing and guiding that behavior by providing reward signals. This separation allows the PORAG framework to optimize both the output generation (via γ) and the nuanced reward assessment (via ϕ) concurrently.

A.4 Adaptive Token-Layer Attention Scoring for Selective Retrieval (ATLAS)

ATLAS enhances RAG through a two-stage process that leverages the policy-optimized LLM's internal states. The Multi-Layer Attention Gradient (MLAG) mechanism detects when the model lacks necessary information by analyzing shifts in attention patterns across layers, triggering retrieval only at critical moments. Once retrieval is triggered, Layerwise Representation Pooling (LRP) selects the most relevant previously generated tokens to construct precise queries that address the model's specific information gaps. This ensures that external knowledge is retrieved only when needed and targeted effectively, resulting in factually accurate responses with minimal computational overhead. Let us define a sequence of tokens $\mathbf{T} = \{t_1, t_2, \ldots, t_n\}$ processed by a fixed pretrained LLM. Throughout this formulation: i indexes the current position in the sequence, L denotes the total number of layers in the model, H represents the number of attention heads per layer, and V is the vocabulary of the language model. The Multi-Layer Attention Gradient (MLAG) mechanism determines when to trigger retrieval by analyzing attention patterns across model layers:

$$\text{MLAG}(t_i) = \alpha \cdot G_i \cdot D_i \cdot s_i$$

Each component serves a specific purpose and is computed directly from observable model states. The gradient factor (G_i) quantifies attention pattern shifts across layers for token t_i:

$$G_i = \sum_{j=1}^{L-1} \eta_j \cdot \left|\bar{A}_{j+1,i} - \bar{A}_{j,i}\right|$$

where $\bar{A}_{j,i}$ is the normalized average attention to the token t_i in layer j:

$$\bar{A}_{j,i} = \frac{\sum_{h=1}^{H} \sum_{k=1}^{i-1} A_{j,h,k,i}}{\max_{m=1}^{i} \sum_{h=1}^{H} \sum_{k=1}^{i-1} A_{j,h,k,m}}$$

where $A_{j,h,k,i}$ is the attention weight from token t_k to token t_i in head h at layer j. Also, $A_{h,i,L}$ is the average attention received by token t_i in head h at layer L:

$$A_{h,i,L} = \frac{1}{i-1} \sum_{k=1}^{i-1} A_{L,h,k,i}$$

Note that for average attention, $A_{h,i,L}$ excludes t_i by averaging over $i-1$ tokens (since a token doesn't attend to itself in autoregressive models). $\eta_j = \frac{j}{L-1}$ is a layer-specific coefficient giving more weight to higher layers. The gradient factor captures shifts in attention patterns between consecutive layers during forward propagation. Consistent patterns suggest the model has adequate information, while sudden changes indicate it may be searching for missing information. Layer weighting (η_j) prioritizes higher layers, which encode more abstract and task-relevant representations, making them critical for detecting when external knowledge is needed. The depth-weighted information density (D_i) measures the importance of token t_i based on model uncertainty and attention distribution:

$$D_i = (1 - p_i(t_i)) \cdot \sum_{h=1}^{H} \phi_h \cdot A_{h,i,L}$$

where the generation probability ($p_i(t_i)$) represents the model's confidence in generating token t_i at position i:

$$p_i(t_i) = \frac{\exp(z_i(t_i))}{\sum_{v \in V} \exp(z_i(v))}$$

where $z_i(t_i)$ is the raw logit (pre-softmax score) for token t_i at position i from the model's final output layer, which is a direct measure of the model's certainty. ϕ_h is a head importance coefficient derived from attention entropy:

$$\phi_h = \frac{\mathcal{H}(A_{L,h})}{\sum_{h'=1}^{H} \mathcal{H}(A_{L,h'})}$$

where $\mathcal{H}(A_{L,h})$ is the entropy of the attention distribution of head h at layer L attending to all preceding tokens $t_1, \ldots, t_i$:

$$\mathcal{H}(A_{L,h}) = -\sum_{j=1}^{i}\sum_{k=1}^{i} A_{L,h,j,k}\log(A_{L,h,j,k}+\epsilon)$$

where ϵ is a small constant (typically 1e-10) to avoid log(0), and $A_{L,h,j,k}$ is the attention weight from token t_j to token t_k in head h at layer L. The entropy $\mathcal{H}(A_{L,h})$ is computed over the full attention distribution within head h at layer L for the current token position i. The depth-weighted information density combines two key signals: model uncertainty, where $(1 - p_i(t_i))$ increases when the model is less confident about generating t_i, and importance of attention, measured by $\sum_{h=1}^{H} \phi_h \cdot A_{h,i,L}$, which quantifies how much the model focuses on t_i across attention heads. Entropy-based head weighting (ϕ_h) is particularly relevant for policy-optimized LLMs, as it prioritizes heads with distributed attention patterns. These heads excel at integrating broader information rather than local patterns, making them more effective at detecting information needs. The Semantic Filter (s_i) excludes tokens unlikely to indicate information needs:

$$s_i = \begin{cases} 0, & \text{if } t_i \in S \text{ or } \texttt{IsNumeric}(t_i) \text{ or } \texttt{IsPunctuation}(t_i) \\ 1, & \text{otherwise} \end{cases}$$

where S is a predefined set of stopwords. This filter improves efficiency and accuracy by focusing on semantically meaningful tokens. The scaling factor α dynamically modulates retrieval sensitivity based on computational load, ensuring efficient operation through a graceful reduction in retrieval frequency. Essentially, when the LLM is "relaxed" (low demand), α maintains higher retrieval sensitivity, prioritizing external information lookup. Conversely, as the LLM becomes "stressed" (resource constraints approach), α smoothly reduces retrieval sensitivity to prevent overload.

$$\alpha = \alpha_0 \cdot e^{-\lambda \frac{C_{\text{current}}}{C_{\max}}}$$

Here, α_0 (typically 0.7-1.0) sets the baseline sensitivity at minimal load, and λ (typically 3-5) is the decay coefficient controlling the reduction rate. Careful selection of these hyperparameters, α_0 and λ, is important to balance retrieval effectiveness and computational efficiency. $C_{\max}$ is the maximum computational budget, and C_{current} reflects real-time resource usage. For RAG, $C_{\max}$ should be configured to 80-90% of available VRAM, with C_{current} monitored via metrics like GPU memory consumption. This exponential decay mechanism prioritizes retrieval when demand is low, smoothly scaling it back under resource pressure, thus maintaining efficiency and preventing system overload. In summary, MLAG analyzes attention patterns across layers and tokens to selectively trigger external information retrieval during text generation. Once retrieval is triggered by MLAG, an effective mechanism is needed to determine what information to retrieve. We propose Layerwise Representation Pooling (LRP), which constructs retrieval queries by selecting tokens from the preceding context based on their relevance to the current token. Formally, for a given token t_i at position i in the sequence, LRP selects a subset of preceding tokens:

$$\text{LRP}(t_i) = \texttt{SelectTopKTokens}(\{t_j : j < i\}, k, \text{relevance})$$

where k is the number of tokens to select (typically 5-7 tokens), and relevance(t_j) is a scoring function that measures the importance of token t_j relative to the current token t_i. The `SelectTopKTokens` function selects the top-k tokens from the preceding context $\{t_j : j < i\}$ based on their relevance scores. We compute this relevance as a weighted combination of attention-based and representation-based similarities:

$$\text{relevance}(t_j) = \beta \cdot \text{AttenScore}(t_j) + (1-\beta) \cdot \text{RepScore}(t_j)$$

where $\beta \in [0, 1]$ is a balancing parameter (optimally set to 0.7 in our experiments). This parameter balances the contribution of attention and representation scores. The attention score quantifies the importance of token t_j based on the attention patterns across all layers and heads:

$$\text{AttenScore}(t_j) = \sum_{l=1}^{L} \psi_l \cdot \frac{1}{H}\sum_{h=1}^{H} A_{l,h,i,j}$$

where $A_{l,h,i,j}$ represents the attention weight from token t_i to token t_j in head h at layer l. Note that unlike MLAG which uses attention towards the current token ($A_{j,h,k,i}$), LRP uses attention from the current token to preceding tokens ($A_{l,h,i,j}$) to capture the relevance of past tokens in the context of the current token being generated. ψ_l is a layer importance coefficient defined as:

$$\psi_l = \begin{cases} 0.2 \cdot \frac{l}{L/3}, & \text{if } l < L/3 \\ 0.5 \cdot \frac{l-L/3}{L/3}, & \text{if } L/3 \le l < 2L/3 \\ 0.3 \cdot \frac{L-l}{L/3}, & \text{otherwise} \end{cases}$$

This piecewise linear layer-weighting scheme, empirically tuned for models like Qwen and LlaMA, prioritizes middle layers, as they are found to encode richer contextual information crucial for effective query formulation, and this specific design has shown strong empirical performance for the targeted LLM architectures. The representation score captures semantic similarity between tokens using their contextualized representations:

$$\text{RepScore}(t_j) = \cos(e_j, e_i)$$

where e_j and e_i are contextualized embeddings for tokens t_j and t_i, respectively, computed as weighted averages of layer-specific hidden states:

$$e_j = \sum_{l=1}^{L} \delta_l \cdot h_{l,j}$$

Here, $h_{l,j}$ represents the hidden state of token t_j at layer l, and δ_l is a layer-specific weight defined as:

$$\delta_l = \frac{\exp(l/\tau)}{\sum_{l'=1}^{L}\exp(l'/\tau)}$$

where τ is a temperature parameter (typically set to 2.0). This temperature parameter concentrates weights towards higher layers, emphasizing the role of deeper representations in capturing token semantics. While LRP does involve computations for attention and representation scores, including embedding calculations and cosine similarity, the overall computational overhead is managed by triggering LRP only when MLAG detects an information need, thus maintaining efficiency compared to always-on retrieval methods. After selecting the top-k tokens based on their relevance scores, we arrange them in their original sequence order to preserve grammatical coherence. We then leverage the language capabilities of the policy-optimized LLM itself to formulate a coherent query by passing these tokens through a simple prompt to produce a more effective retrieval query. For instance, a prompt like "Formulate a search query from these tokens: [selected tokens]" can be used. The performance of LRP has been observed to be superior to simpler query construction methods such as using only the current

token or a fixed window of preceding tokens, as LRP dynamically selects semantically relevant tokens based on both attention and representation metrics. To maintain computational efficiency and prevent the retrieval process from becoming a bottleneck, we employ a selective approach where LRP is not triggered for every generated token. Instead, a computationally inexpensive check first determines if a potential information gap exists. If True, indicating model uncertainty and semantic importance, it signals a potential need for external knowledge. In such cases, we then engage the MLAG mechanism—detailed in ATLAS—to rigorously confirm this information need through deeper analysis of the model's internal states. Only if MLAG confirms retrieval is necessary do we proceed with LRP for query construction. The `ComputeRelevance` check is defined as:

$$\texttt{ComputeRelevance}(t_i) = \begin{cases} \texttt{True}, & \text{if } p_i(t_i) < \tau_p \text{and } s_i = 1 \\ \texttt{False}, & \text{otherwise} \end{cases}$$

where $p_i(t_i)$ is the generation probability of token t_i, τ_p is a probability threshold (typically 0.5), and s_i is a binary semantic filter.

A.4.1 Computational Workflow and Implementation of ATLAS:. The complete ATLAS workflow operates sequentially across two key phases. In the token analysis phase, for each generated token t_i, the system first computes its probability $p_i(t_i) = \frac{\exp(z_i(t_i))}{\sum_{v \in V} \exp(z_i(v))}$ from model logits and applies the semantic filter s_i to identify meaningful tokens. When conditions for analysis are met ($p_i(t_i) < \tau_p$ **and** $s_i = 1$), ATLAS calculates the Multi-Layer Attention Gradient score $\text{MLAG}(t_i) = \alpha \cdot G_i \cdot D_i \cdot s_i$ by analyzing attention patterns across layers. If this score is deemed sufficiently high to warrant retrieval, the system activates its retrieval mechanism. The query formulation phase then begins, wherein Layerwise Representation Pooling computes relevance scores for preceding tokens through a balanced attention and semantic similarity formula: $\text{relevance}(t_j) = \beta \cdot \text{AttenScore}(t_j) + (1 - \beta) \cdot \text{RepScore}(t_j)$. Using these scores, ATLAS selects the top-k most relevant tokens via $\text{LRP}(t_i) = \text{SelectTokens}(\{t_j : j < i\}, k, \text{relevance})$, preserves their original sequence order for coherence, and constructs a focused retrieval query. After acquiring external knowledge with this targeted query, it incorporates the retrieved information into the generation context, enabling the language model to produce factually enhanced outputs without modifying its underlying parameters.

A.5 CRITIC: Cache Reduction via Importance-based Token Inclusion Criteria

Key-Value (KV) caching is essential in modern large language models (LLMs) because it dramatically reduces computational redundancy during autoregressive text generation. When generating text token by token, traditional approaches recalculate attention for all previous tokens with each new prediction, leading to quadratic computational complexity ($O(n^2)$) that severely limits efficiency for long sequences. In the standard self-attention mechanism, given a sequence of input tokens, each token is transformed into a query vector ($\mathbf{Q}$), a key vector ($\mathbf{K}$), and a value vector ($\mathbf{V}$) through learnable weight matrices: $\mathbf{Q} = \mathbf{X}\mathbf{W}^Q$, $\mathbf{K} = \mathbf{X}\mathbf{W}^K$, and $\mathbf{V} = \mathbf{X}\mathbf{W}^V$, where $\mathbf{X} \in \mathbb{R}^{n \times d}$ is the matrix of input token embeddings, with n being the sequence length and d the embedding dimension. Without caching, for each new token, the attention weights are calculated as $\text{softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_h}})$, where $\mathbf{Q}$ is the query matrix, $\mathbf{K}$ is the key matrix, and d_h is the head dimension. The scaling factor $\sqrt{d_h}$ prevents extremely small gradients in the softmax operation. The context vector is then computed as $\text{softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_h}})\mathbf{V}$. KV caching stores these previously computed key ($\mathbf{K}$) and value ($\mathbf{V}$) tensors from each layer of the attention mechanism, eliminating the need to recompute them for each generated token and reducing complexity from quadratic to linear ($O(n)$). Specifically, for the t-th token t, we compute $\mathbf{Q}_t$, $\mathbf{K}_t$, and $\mathbf{V}_t$ for the new token only. The cached keys and values, $\mathbf{K}_{cached}$ and $\mathbf{V}_{cached}$, contain the keys and values from tokens 1 to $t-1$. The attention weights are then computed as $\text{softmax}(\frac{\mathbf{Q}_t\mathbf{K}^T}{\sqrt{d_h}})$, where $\mathbf{K} = [\mathbf{K}_{cached}; \mathbf{K}_t]$ denotes the concatenation of the cached keys and the current key. The context vector is then computed as $\text{softmax}(\frac{\mathbf{Q}_t[\mathbf{K}_{cached};\mathbf{K}_t]^T}{\sqrt{d_h}})[\mathbf{V}_{cached}; \mathbf{V}_t]$. This significantly reduces computation because we only need to compute the attention weights and context vector for the current token relative to the cached keys and values, rather than recomputing the entire attention matrix for all tokens at each step. This optimization yields substantial speedups—often 2-10x faster inference—and enables processing of much longer contexts than would otherwise be possible given hardware constraints. However, as sequence length grows, even with KV caching, memory usage becomes prohibitive since the cache size scales linearly with sequence length and model size (number of layers, attention heads, and hidden dimension). The memory requirement is proportional to $(L \times H \times 2 \times n \times d_h \times b)/8$ bytes, where L is the number of layers, H is the number of attention heads per layer, the factor of 2 accounts for both keys and values, n is the sequence length, d_h is the head dimension, and b is the number of bits in the data type. It's crucial to consider the data type's precision when estimating memory usage; for instance, using half-precision('bfloat16') (b=16) significantly reduces memory compared to full-precision('float32') (b=32). This creates a fundamental tension: while larger context windows enhance model capabilities by providing more information, they also demand significantly more memory resources, creating a need for KV cache optimization techniques. The challenge becomes particularly acute in real-world RAG applications that benefit from extended contexts. To mitigate the KV cache memory bottleneck, a variety of compression techniques are employed, each with its own trade-offs in terms of memory reduction, computational overhead, and potential impact on model accuracy. Quantization, a common technique, reduces numerical precision by converting floating-point values to lower-bit integers using the formula $x_{int} = \text{round}(\frac{x - x_{min}}{x_{max} - x_{min}} \times (2^b - 1))$, where b represents the target bit width. This directly decreases the memory footprint per value by representing values with fewer bits, allowing for more efficient storage of the KV cache. Pruning selectively removes key-value pairs associated with less important attention heads, guided by importance scores such as $s_h = \mathbb{E}_{x \sim \mathcal{D}}[||A_h(x)||_F]$, where $\mathbb{E}_{x \sim \mathcal{D}}$ denotes expectation over the data distribution, $A_h(x)$ is the attention matrix for head h, and $||\cdot||_F$ is the Frobenius norm. This score s_h quantifies the average importance of attention head h. By removing the key-value pairs generated by these less important heads, pruning effectively reduces the representation of tokens

Algorithm 1 Group Relative Policy Optimization for Retrieval-Augmented Generation (PORAG)

Input: Initial RAG policy model $\pi_{\gamma_{\text{init}}}$ (with QLoRA adapters γ), reward models with parameters ϕ_1 and ϕ_2 (reward heads), RAG training dataset $\mathcal{D} = \{(x_i, d_i, y_i^*)\}_{i=1}^N$, hyperparameters: clipping parameter ϵ (=0.2), fidelity reward weight α (=0.7), quality reward weight β (=0.3), reward clipping threshold c_1 (=10.0), reward scaling factor γ_{scale}, policy update iterations μ, group size G, policy learning rate η_γ, reward model learning rate η_R ($\eta_R > \eta_\gamma$), KL divergence weight ω_2, clipped surrogate objective weight ω_1, minimum standard deviation σ_{min}, gradient clipping value c_{value} (=3.0), gradient norm clipping c_{norm} (=1.0)
Output: Optimized RAG policy model π_γ

(1) Initialize RAG policy model: $\gamma \leftarrow \gamma_{\text{init}}$ (QLoRA adapters)
(2) For iteration $i = 1, 2, \ldots, I$ do: (Main Training Epoch - Iterating over the dataset)
  (a) Set reference model: $\pi_{\text{ref}} \leftarrow \pi_\gamma$
  (b) For step $j = 1, 2, \ldots, M$ do: (Mini-batch Update Step - Processing a batch of data)
    (i) Sample batch $\mathcal{B}_j$ from dataset $\mathcal{D}$
    (ii) Set old policy: $\pi_{\gamma_{\text{old}}} \leftarrow \pi_\gamma$
    (iii) For each $(x, d) \in \mathcal{B}_j$: (Group Output Generation and Reward Calculation for each data point in batch)
      (A) Sample G outputs: $\{y^{(1)}, y^{(2)}, \ldots, y^{(G)}\} \sim \pi_{\gamma_{\text{old}}}(\cdot|x, d)$
      (B) Compute dual rewards using reward heads (ϕ_1, ϕ_2):

$$r_{\text{fidelity}}^{(i)} = R_{\text{fidelity}}(x, d, y^{(i)}; \phi_1)$$
$$r_{\text{quality}}^{(i)} = R_{\text{quality}}(x, d, y^{(i)}; \phi_2)$$

      (C) Compute combined rewards: $R_{\text{combined}}^{(i)} = \alpha \cdot r_{\text{fidelity}}^{(i)} + \beta \cdot r_{\text{quality}}^{(i)}$
      (D) Compute final reward with clipping and scaling: $R_{\text{final}}^{(i)} = \text{clip}(R_{\text{combined}}^{(i)}, -c_1, c_1) \cdot \gamma_{\text{scale}}$
      (E) Compute group statistics using $R_{\text{final}}^{(i)}$:

$$\mu_R = \frac{1}{G} \sum_{i=1}^{G} R_{\text{final}}^{(i)}$$
$$\sigma_R = \max\left(\sqrt{\frac{1}{G} \sum_{i=1}^{G} (R_{\text{final}}^{(i)} - \mu_R)^2}, \sigma_{\text{min}} \right)$$

      (F) Calculate advantages: $\hat{A}_i = \frac{R_{\text{final}}^{(i)} - \mu_R}{\sigma_R}$
    (iv) For GRPO iteration $k = 1, 2, \ldots, \mu$ do:(Inner Policy Optimization Loop - Multiple GRPO updates per mini-batch)
      (A) Compute policy objective (token-level clipped surrogate objective):

$$L_{\text{clip}}(\gamma) = \frac{1}{G} \sum_{i=1}^{G} \frac{1}{|y^{(i)}|} \sum_{t=1}^{|y^{(i)}|} \min\left(r_t(\gamma)\hat{A}_i, \text{clip}(r_t(\gamma), 1-\epsilon, 1+\epsilon)\hat{A}_i \right)$$ // Using sample-wise advantage $\hat{A}_i$ for all tokens in $y^{(i)}$

      (B) Compute KL regularization (sample-based approximation with token-averaging):

$$D_{\text{KL}}(\pi_\gamma || \pi_{\text{ref}}) = \frac{1}{|\mathcal{B}_j|} \sum_{(x,d) \in \mathcal{B}_j} \frac{1}{G} \sum_{i=1}^{G} \frac{1}{|y^{(i)}|} \sum_{t=1}^{|y^{(i)}|} \text{KL}(\pi_{\text{ref}}(\cdot|x, d, y_{<t}^{(i)}) || \pi_\gamma(\cdot|x, d, y_{<t}^{(i)}))$$

      (C) Compute total objective: $J_{\text{GRPO-RAG}}(\gamma) = \omega_1 \cdot L_{\text{clip}}(\gamma) - \omega_2 \cdot D_{\text{KL}}(\pi_\gamma || \pi_{\text{ref}})$
      (D) Compute gradients: $\nabla_\gamma J_{\text{GRPO-RAG}}(\gamma)$
      (E) Clip gradients by value: $\nabla_\gamma J_{\text{clipped}} = \text{clip}(\nabla_\gamma J_{\text{GRPO-RAG}}(\gamma), -c_{\text{value}}, c_{\text{value}})$
      (F) Normalize gradients by norm: $\nabla_\gamma J_{\text{normalized}} = \frac{\nabla_\gamma J_{\text{clipped}}}{||\nabla_\gamma J_{\text{clipped}}||_2} \cdot \min(||\nabla_\gamma J_{\text{clipped}}||_2, c_{\text{norm}})$
      (G) Update policy (γ - QLoRA adapters only) with normalized gradients: $\gamma \leftarrow \gamma + \eta_\gamma \nabla_\gamma J_{\text{normalized}}$
    (v) Update reward models (reward heads ϕ_1, ϕ_2) using reward losses: // $\mathcal{L}_{\text{fidelity}}$ (ROUGE), $\mathcal{L}_{\text{quality}}$ (Semantic/QA Metrics)

$$\phi_1 \leftarrow \phi_1 + \eta_R \nabla_{\phi_1} \mathcal{L}_{\text{fidelity}}(\phi_1)$$
$$\phi_2 \leftarrow \phi_2 + \eta_R \nabla_{\phi_2} \mathcal{L}_{\text{quality}}(\phi_2)$$

    // Gradients do not affect base model weights
(3) Return optimized RAG policy π_γ

Algorithm 2 Adaptive Token-Layer Attention Scoring for Selective Retrieval (ATLAS)

Input: Token sequence **T** // **T**: Input sequence of tokens, Pre-trained LLM // Pre-trained LLM: Fixed Pre-trained Large Language Model, Hyperparameters ($\tau_p, \theta, k, \beta, \tau, \alpha_0, \lambda, C_{\max}$) // Hyperparameters for ATLAS: τ_p: Probability threshold, θ: MLAG threshold, k: Top-k tokens for LRP, β: Relevance balance, τ: Embedding temperature, α_0: Base scaling factor, λ: Decay coefficient, $C_{\max}$: Max compute budget, Stopword set S // S: Set of stopwords, Model parameters ($L, H, V, \psi_l, \delta_l$) // Model parameters: L: Layers, H: Heads, V: Vocabulary, ψ_l: LRP layer weights, δ_l: Embedding layer weights

(1) **1. Initialization:**
   (a) 1.1. Set scaling factor: $\alpha = \alpha_0 \cdot e^{-\lambda \frac{C_{\text{current}}}{C_{\max}}}$ // α: Scaling factor, C_{current}: Current compute usage
(2) **2. Token Analysis Phase (MLAG):** // MLAG: Multi-Layer Attention Gradient
   - 2.1. For each token t_i in the sequence **T**: // t_i: i-th token in sequence **T**
   (a) 2.1.1. Compute Generation Probability: $p_i(t_i)$ // $p_i(t_i)$: Generation probability of token t_i
   (b) 2.1.2. Apply Semantic Filter: Determine s_i (0 or 1) based on t_i // s_i: Semantic filter (1 if token is semantically meaningful, 0 otherwise)
   (c) 2.1.3. If $p_i(t_i) < \tau_p$ and $s_i = 1$: // τ_p: Probability threshold
      – 2.1.3.1. Compute Multi-Layer Attention Gradient Score: $\text{MLAG}(t_i) = \alpha \cdot G_i \cdot D_i \cdot s_i$ // G_i: Gradient factor, D_i: Depth-weighted information density
      – 2.1.3.2. If $\text{MLAG}(t_i) > \theta$: // θ: MLAG score threshold
         ∗ 2.1.3.2.1. Retrieval Triggered for token t_i
         ∗ 2.1.3.2.2. Go to **Query Formulation Phase (LRP)** // LRP: Layerwise Representation Pooling
(3) **3. Query Formulation Phase (LRP):**
   - 3.1. If Retrieval Triggered:
   (a) 3.1.1. Compute Relevance Scores: relevance(t_j) for all preceding tokens t_j // t_j: Preceding token, relevance(t_j): Relevance score of token t_j
   (b) 3.1.2. Select Top-k Tokens: $\{t_{j_1}, \ldots, t_{j_k}\} = \text{SelectTopK}(\{t_j : j < i\}, k, \text{relevance})$ // k: Number of top tokens to select
   (c) 3.1.3. Formulate Query from Top-k Tokens
   (d) 3.1.4. **Output:** Retrieval Query
   (e) 3.2. Else:
      (i) 3.2.1. **Output:** No Retrieval Triggered

within the cache from the perspective of these less critical heads. This leads to a smaller memory footprint because fewer key-value pairs are stored for each token. Low-rank approximations decompose the key matrix $\mathbf{K}$ into the product $\mathbf{USV}^T$, where $\mathbf{U} \in \mathbb{R}^{n \times r}$, $\mathbf{S} \in \mathbb{R}^{r \times r}$, $\mathbf{V} \in \mathbb{R}^{d_k \times r}$, and the rank r is much smaller than both the sequence length n and the key dimension d_k. This decomposition dramatically reduces the memory required to store the key matrix by representing it with lower-dimensional components. Windowing strategies, such as sliding window attention, preserve only the most recent w tokens ($\mathbf{K}_{cached} = \mathbf{K}_{t-w:t-1}$). By limiting the context window to the most recent tokens, windowing directly reduces the sequence length and, consequently, the memory needed for the keys and values in the cache. These implementations can be categorized as either static (where compression parameters are fixed before inference) or dynamic (where parameters are adapted during inference based on content importance). Dynamic approaches have the potential to preserve generation quality by allocating resources more efficiently. Ultimately, effective KV cache implementation requires careful consideration of hardware characteristics, memory management strategies, data layout optimization, efficient kernel design, and the trade-offs between memory reduction, computational cost, and model accuracy. The impact of these techniques on model accuracy can be measured through metrics like attention entropy: $H(A_i) = -\sum_j A_{ij} \log A_{ij}$, where A_{ij} represents the normalized attention score from token i to token j. Higher entropy indicates more distributed attention patterns, which may be more sensitive to aggressive compression techniques.

A.6 Proposed Method

To address the substantial memory demands of large language models during inference, this work introduces an adaptive Key-Value (KV) cache compression strategy. This technique selectively retains tokens based on their calculated importance (I), optimizing the trade-off between memory footprint and model performance. The framework is designed to be architecture-agnostic and implements a hybrid token importance strategy that integrates attention-based, entropy-based, and gradient-based importance measures. These measures are combined through a weighted formulation to identify critical tokens within each attention layer of the language model. (a) The attention-based importance strategy (I_{attn}) quantifies the strength of a token's relationships by calculating normalized attention scores across the sequence. The process begins with computing attention scores as the scaled dot product of the query ($Q \in \mathbb{R}^{n \times d_k}$) and key ($K \in \mathbb{R}^{n \times d_k}$) matrices, represented as $S \in \mathbb{R}^{n \times n}$, where $d_k = \frac{d_{\text{model}}}{h}$ is the dimension of each attention head in a multi-head attention mechanism. These scores are then transformed into probability distributions using the softmax function, yielding attention weights $A \in \mathbb{R}^{n \times n}$. Since large language models have multiple layers (L), these computations occur independently at each layer, where Q^l, K^l, V^l are computed for every layer $l \in \{1, ..., L\}$. The

importance of each token is computed by summing the absolute values of these attention weights across all attention heads (h) and all positions (j) in the sequence: $\text{strength}_i = \sum_{h,j} |A^l_{h,i,j}|$, where $A^l_{h,i,j}$ represents the attention weight of the i-th token in the l-th layer. This raw strength metric is then normalized to the range $[0, 1]$ as follows:

$$I_{\text{attn}}(i) = \frac{\text{strength}_i - \min(\text{strength})}{\max(\text{strength}) - \min(\text{strength}) + \epsilon},$$

where ϵ is a small constant to prevent division by zero. This normalization ensures comparable importance scores across different sequences, model states, and layers. In short, randomly discarding tokens from the KV cache can degrade model performance by losing important contextual information. Token importance varies across inputs and contexts, making a dynamic approach essential. The attention-based measure quantifies token importance on-the-fly using current attention patterns, ensuring the retention of the most relevant tokens that impact model predictions. By leveraging existing attention computations during inference, it minimizes additional computational overhead. (b) The entropy-based importance strategy (I_{entropy}) leverages information theory principles to quantify the complexity and diversity of a token's attention patterns. After computing attention probabilities using the standard scaled dot-product attention mechanism:

$$A^l = \text{softmax}\left(\frac{Q^l (K^l)^T}{\sqrt{d_k}}\right), \quad A^l \in \mathbb{R}^{n \times n},$$

where $Q^l, K^l, V^l \in \mathbb{R}^{n \times d_k}$ are the query, key, and value matrices at the l-th layer, and $d_k = \frac{d_{\text{model}}}{H}$ represents the key dimension per attention head. The Shannon entropy for each token's attention distribution is then calculated as:

$$H^l(i) = -\sum_{j=1}^{n} A^l_{i,j} \log(A^l_{i,j} + \epsilon),$$

where $A^l_{i,j}$ is the attention probability that the i-th token assigns to the j-th token in the l-th layer, and $H^l(i)$ is the total entropy for the i-th token at layer l. This entropy value captures how widely and evenly a token distributes its attention across the sequence—higher entropy suggests the token has more complex relationships with other tokens. The entropy values are averaged across all attention heads (H) to obtain a comprehensive metric:

$$\bar{H}^l(i) = \frac{1}{H} \sum_{h=1}^{H} H^l_h(i),$$

where $H^l_h(i)$ represents the Shannon entropy computed for the i-th token in the h-th attention head of the l-th layer, and $\bar{H}^l(i)$ is the entropy averaged across all heads for the i-th token at layer l. Finally, these average entropy values are normalized using min-max scaling:

$$I^l_{\text{entropy}}(i) = \frac{\bar{H}^l(i) - \min(\bar{H}^l)}{\max(\bar{H}^l) - \min(\bar{H}^l) + \epsilon},$$

where ϵ is a small constant to prevent division by zero. This normalization ensures comparable entropy-based importance scores across different sequences and layers. Not all tokens contribute equally to the model's understanding—some have simple, predictable relationships, while others exhibit complex interactions. The entropy-based measure quantifies attention pattern complexity to identify and retain tokens with richer relationships. Tokens with higher entropy-based importance scores maintain more complex relationships within the sequence and are therefore prioritized for retention during compression. By leveraging existing attention computations during inference, this approach minimizes additional computational overhead. (c) The gradient-based importance strategy ($\mathcal{I}^l_{\text{grad}}(i)$) directly measures each token's contribution to model prediction consistency using gradient information. It evaluates the consistency between the current attention output and the attention output of the same layer from the previous token generation step, representing the model's prior belief as follows:

$$L^l = \text{MSE}(\text{Attention}^l(Q^l, K^l, V^l), \text{Prev}^l),$$

where: $\text{Attention}^l(Q^l, K^l, V^l) \in \mathbb{R}^{n \times d_k}$ represents the current attention operation at layer l, $\text{Prev}^l \in \mathbb{R}^{n \times d_k}$ denotes the attention output from the same attention layer l in the previous decoding step. To mitigate memory consumption, the implementation employs gradient checkpointing. The gradients of this loss with respect to the key (K^l) and value (V^l) representations are computed as follows:

$$G^l_K = \frac{\partial L^l}{\partial K^l} \in \mathbb{R}^{n \times d_k}, \quad G^l_V = \frac{\partial L^l}{\partial V^l} \in \mathbb{R}^{n \times d_k},$$

The importance of each token is then determined by summing the absolute values of these gradients across all attention heads (H) at layer l:

$$\mathcal{I}^l_{\text{grad}}(i) = \sum_{h=1}^{H} \left(|G^l_{K,h,i}| + |G^l_{V,h,i}| \right) \in \mathbb{R},$$

where: $\mathcal{I}^l_{\text{grad}}(i)$ denotes the gradient-based importance score for the i-th token at layer l, $G^l_{K,h,i} \in \mathbb{R}$ and $G^l_{V,h,i} \in \mathbb{R}$ are the gradients of the loss function L^l with respect to the key and value representations for attention head h at layer l. This raw gradient-based importance is then normalized:

$$I^l_{\text{grad}}(i) = \frac{\mathcal{I}^l_{\text{grad}}(i) - \min(\mathcal{I}^l_{\text{grad}})}{\max(\mathcal{I}^l_{\text{grad}}) - \min(\mathcal{I}^l_{\text{grad}}) + \epsilon} \in \mathbb{R},$$

where: ϵ is a small constant to prevent division by zero. The gradient-based approach provides a direct measure of how sensitive the model's predictions are to changes in each token's representations at layer l, highlighting tokens that most significantly influence the output. (d) The hybrid importance strategy (I_{hybrid}) combines the strengths of the previous approaches through a weighted combination of their respective importance scores. This strategy is formulated as follows:

$$I_{\text{hybrid}}(i) = w_{\text{attn}} \cdot I_{\text{attn}}(i) + w_{\text{entropy}} \cdot I_{\text{entropy}}(i) + w_{\text{grad}} \cdot I_{\text{grad}}(i),$$

where w_{attn}, w_{entropy}, and w_{grad} are configurable weights that sum to 1. This weighted sum is further normalized to ensure values fall within the range $[0, 1]$. The hybrid approach provides flexibility to customize the compression behavior based on specific model characteristics allowing implementers to balance the different aspects of token importance according to their needs. Following the computation of token importances using the hybrid strategy (I_{hybrid}), which integrates attention-based, entropy-based, and gradient-based measures, the framework determines the number of tokens to retain (n_c) in the Key-Value (KV) cache. It is designed to optimize memory usage while preserving model performance. The number of tokens to retain is calculated as:

$$n_c = \min(\max(m, \lfloor (1-r) \cdot n \rfloor), n-1),$$

where r is the compression ratio (typically between 0.1 and 0.5), and m is a minimum token count. It ensures that at least m tokens are retained while also preserving at least one token for potential removal, guaranteeing $n_c < n$. The minimum token count (m) prevents excessive compression that could degrade model performance, while the upper bound ($n-1$) ensures the integrity of the sequence by always leaving at least one token available for removal. Once n_c is determined, the framework selects the tokens with the highest importance scores for retention using a top-k operation:

$$\text{SelectedTokens} = \text{TopK}(I_{\text{hybrid}}, n_c),$$

where I_{hybrid} is the vector of hybrid importance scores for all tokens in the sequence, and $\text{TopK}(\cdot, n_c)$ selects the n_c tokens with the highest scores. This approach ensures that only the most critical tokens, which significantly influence model predictions, are retained, optimizing memory usage without compromising performance. To minimize computational overhead, the framework incorporates a delayed caching mechanism. Compression is initiated only after processing a minimum number of tokens (m), ensuring that shorter sequences (with fewer than m tokens) operate without compression. This threshold-based approach ensures that compression overhead is incurred only when the benefits of memory savings outweigh the computational costs, making the framework practical for sequences of varying lengths. Additionally, the framework dynamically adjusts the compression ratio based on current memory usage to balance memory savings and model performance. The adaptive compression ratio (r_{adaptive}) is computed as:

$$r_{\text{adaptive}} = \min(r_{\text{base}} + \alpha \cdot \frac{M_{\text{used}}}{M_{\text{total}}}, r_{\max}),$$

where M_{used} represents current memory consumption, M_{total} is the total available memory, α is a tunable parameter controlling adaptation sensitivity, r_{base} is the base compression ratio, and $r_{\max}$ is the maximum allowable compression ratio. This adaptive mechanism increases compression when memory pressure is high and relaxes it when resources are abundant, ensuring efficient memory utilization without exceeding hardware limits. In summary, the framework combines a hybrid importance calculation, token retention logic, delayed caching, and adaptive compression to achieve efficient memory usage while maintaining model performance in RAG contexts. This makes it particularly suitable for deployment in large language models, especially in long-context applications where memory demands are significant. During text generation, the framework implements a phased approach to adaptive KV cache compression. Initially, tokens are collected without compression until a minimum token threshold (m) is reached, ensuring that shorter sequences operate without compression to minimize unnecessary computational overhead. Once the threshold is exceeded, the framework performs a series of steps for each generated token: it extracts hidden states and computes query, key, and value projections; appends keys and values to an accumulation buffer while tracking the total number of processed tokens; concatenates all cached keys and values when the token count exceeds the threshold; computes attention scores between the current queries and the cached keys; calculates token importances using the selected strategy (e.g., the hybrid strategy I_{hybrid}); selects the top-k most important tokens based on their importance scores; reconstructs the KV cache with the selected tokens, discarding less important ones; and updates compression statistics to track memory savings and performance impact. CRITIC reconstructs the KV cache after importance-based compression, preserving sequence integrity. By retaining the most critical tokens and synchronizing their positional indices, it prevents token misalignment—essential for autoregressive text generation where self-attention relies on sequential dependencies. This reconstruction enables long-sequence processing while optimizing memory usage, ensuring model fluency and contextual coherence. This phased approach ensures that compression is applied only when necessary (after processing at least m tokens) and dynamically adapts to the importance of tokens in the sequence, optimizing memory usage while preserving model performance.

A.6.1 CRITIC Evaluation. The evaluation of the CRITIC module's impact on the PORAG+ATLAS framework reveals a modest performance trade-off that accompanies significant efficiency gains across all benchmark datasets. As shown in Table 11, the Qwen2.5-3B model with CRITIC integration experiences only slight decreases in HotpotQA metrics, with Joint EM dropping from 45.29% to 42.37% and Joint F1 declining from 71.32% to 67.95%. Similarly, Table 12 demonstrates minor reductions in Gorilla performance, where overall accuracy falls marginally from 76.38% to 73.85% while wrong API calls see a small increase from 4.98% to 6.77%. The PubMedQA results in Table 13 follow this pattern, showing slight dips in both accuracy (78.35% to 74.62%) and F1 score (74.56% to 69.83%). These minimal quality trade-offs are offset by substantial efficiency improvements, as evidenced in Table 14, where latency is nearly halved from 68.27 seconds to 34.19 seconds and throughput more than doubles from 120 to 242 tokens per second. The consistent but modest performance impact suggests that CRITIC's memory optimization strategy successfully balances computational benefits with acceptable quality preservation, making it particularly valuable for applications where efficiency is prioritized without significantly compromising output accuracy.

Table 11: HotpotQA Quality Metrics

Model	Joint EM (%)	Joint F1 (%)
PORAG+ATLAS (Baseline)	**45.29**	**71.32**
PORAG+ATLAS + CRITIC	42.37	67.95

Table 12: Gorilla Quality Metrics

Model	Overall Acc. (%)	Wrong API (%)
PORAG+ATLAS (Baseline)	**76.38**	**4.98**
PORAG+ATLAS + CRITIC	73.85	6.77

A.6.2 Computational Complexity. The computational complexity of our adaptive KV cache compression framework is dominated by token importance computation and token selection. Given a sequence of length n, with H attention heads, key/value dimension d, and batch size b, computing token importance requires $O(bHn^2d)$

Table 13: PubMedQA Quality Metrics

Model	Accuracy (%)	F1 (%)
PORAG+ATLAS (Baseline)	**78.35**	**74.56**
PORAG+ATLAS + CRITIC	74.62	69.83

Table 14: Efficiency Metrics

Model	Latency (sec)	Tokens/sec (↑)
PORAG+ATLAS (Baseline)	68.27	120
PORAG+ATLAS + CRITIC	34.19	242

operations for attention-based and entropy-based strategies, matching standard self-attention complexity. The gradient-based strategy adds backpropagation overhead but remains $O(bHn^2d)$ asymptotically, with gradient checkpointing minimizing memory overhead. Token selection, using a top-k operation, has a complexity of $O(bn \log n)$ with heap-based selection, where $k = n_c$. The number of retained tokens n_c is calculated as
$n_c = \min(\max(m, \lfloor (1-r) \cdot n \rfloor), n-1)$, ensuring at least m tokens are kept and one token is removed. This reduces the memory footprint from $O(bHnd)$ to $O(bHn_cd)$, achieving a reduction factor of $\frac{n_c}{n}$. Compression is triggered only when the sequence length exceeds m, minimizing overhead for short sequences, while the adaptive compression ratio dynamically adjusts r based on memory pressure, balancing efficiency and performance.

A.7 Comparing PORAG and RAFT Methodologies

Policy-Optimized Retrieval-Augmented Generation (PORAG) and Retrieval-Augmented Fine-Tuning (RAFT) [60] offer fundamentally different strategies for optimizing RAG systems. RAFT employs supervised fine-tuning (SFT) on static, curated datasets containing predefined question-response pairs accompanied by both relevant ("golden") and irrelevant ("distractor") documents. It optimizes indirectly by teaching the model to differentiate between useful and distracting documents through explicit training examples and incorporates logical reasoning via Chain-of-Thought (CoT) prompts. However, RAFT is inherently limited by its reliance on predefined data, single-objective cross-entropy optimization, and its inability to explicitly optimize retrieval fidelity and generation quality independently. In contrast, PORAG employs Group Relative Policy Optimization (GRPO), an advanced reinforcement learning method, to directly optimize multiple generation quality dimensions simultaneously through specialized reward models. PORAG dynamically generates policy-driven training samples, directly optimizing retrieval fidelity—how faithfully retrieved information is reflected—and response quality, including coherence, fluency, and helpfulness. Unlike RAFT, PORAG implicitly and dynamically handles distractors through reward modeling and advantage estimation rather than explicitly embedding distractors in supervised training sets. Additionally, PORAG incorporates explicit advantage estimation and KL-divergence regularization during policy updates to maintain controlled adaptation in retrieval-augmented generation. This stabilizes training, prevents drastic policy shifts, and balances retrieval fidelity with the model's inherent parametric knowledge, enhancing robustness and generalization across retrieval scenarios. In contrast, RAFT provides robustness primarily within domain-specific scenarios due to its explicit distractor-aware fine-tuning but lacks dynamic adaptability beyond its predefined training context. In summary, PORAG offers greater deployment flexibility, nuanced generation optimization, and dynamic adaptability, addressing key limitations of RAFT related to static supervision, single-strategy optimization, and the lack of direct optimization of retrieval fidelity and response quality.

A.8 Comparing DRAGIN and ATLAS Methodologies

Dynamic Retrieval Augmented Generation based on the Information Needs of Large Language Models (DRAGIN) [38] and Adaptive Token-Layer Attention Scoring for Selective Retrieval (ATLAS) both dynamically determine the optimal timing (when retrieval should occur) and the specific content to retrieve (query formulation) based on the internal states and immediate informational needs of the language model during text generation. DRAGIN primarily leverages final-layer self-attention to identify real-time information gaps. Conversely, ATLAS employs a sophisticated Multi-Layer Attention Gradient (MLAG) analysis, explicitly quantifying attention shifts across multiple transformer layers to capture nuanced transitions indicative of deeper knowledge gaps. For query formulation, DRAGIN constructs retrieval queries using attention patterns from the final layer, combined with token-level semantic filters. ATLAS, in contrast, integrates Layerwise Representation Pooling (LRP), combining semantic similarity and attention scores across layers, along with token-level semantic filters, to form retrieval queries, thereby enhancing semantic precision. In terms of resource management, ATLAS explicitly considers real-time computational load via a dynamic scaling factor, optimizing retrieval frequency relative to resource availability. DRAGIN utilizes a simpler exponential scaling factor, adjusting retrieval sensitivity based on resource usage, but without the fine-grained computational tracking featured in ATLAS. Overall, ATLAS's integrated, multi-layer attention and resource-aware approach offers superior adaptability and accuracy in dynamically identifying subtle retrieval needs, while DRAGIN presents a simpler final-layer attention-driven strategy, achieving computational simplicity at the potential cost of retrieval precision depth.

A.9 Test-Time Scaling of LLMs

Test-time scaling inference for Large Language Models (LLMs) leverages advanced algorithmic techniques designed to enhance model outputs without altering the underlying weights. These methods dynamically adjust reasoning depth, sampling strategies, and validation processes during inference, optimizing efficiency and output quality in real time. This approach is particularly valuable in resource-constrained environments where retraining or fine-tuning models is impractical. By strategically scaling complexity based on task demands, these techniques enable LLMs to navigate complex problem spaces more effectively, ensuring robust decision-making, improved accuracy, and reduced computational

costs. At its core, test-time scaling in LLMs can be mathematically modeled through a utility-cost optimization framework. By defining $U(q, c)$ as the utility function where q represents output quality and c represents computational cost, and $f_\theta(x, s)$ as the LLM function with parameters θ, input x, and scaling strategy s, we can formulate the fundamental objective as maximizing utility while managing resource constraints: $\max_{s \in S} U(q(f_\theta(x, s)), c(s))$ subject to $c(s) \leq C_{max}$, where S represents the set of all possible test-time scaling strategies, $q(f_\theta(x, s))$ measures the quality of model outputs, $c(s)$ represents the computational cost of strategy s, and C_{max} is the maximum allowable computational budget. This mathematical formulation captures the essential trade-off that underlies all test-time scaling approaches. A form of Weak-to-Strong Distillation serves as a foundational strategy for test-time scaling inference techniques, where diverse preliminary outputs are generated and iteratively refined to enhance reasoning and accuracy. This approach improves robustness by progressively strengthening outputs through evaluation and refinement, ensuring accurate and consistent results. These inference techniques represent advanced strategies for test-time scaling in LLMs, significantly enhancing language model capabilities by implementing metacognitive processes such as decomposing problems, evaluating intermediate results, and refining solutions—effectively mimicking human deliberative reasoning while maintaining inference efficiency. By dynamically adjusting computational resources during inference and scaling complexity only when necessary, these methods optimize both efficiency and output quality. This adaptive approach boosts accuracy, minimizes hallucinations and logical errors, and enhances the suitability of LLMs for high-stakes decision-making scenarios.

A.9.1 ***Self-Consistency Algorithm.*** : Self-Consistency [22, 44] enhances model reliability by generating multiple independent reasoning trajectories and selecting the most consistent answer through stochastic decoding. Let $\mathcal{M}$ be a language model with parameters θ and x be an input query. The Self-Consistency framework can be formalized as follows:

$$y^* = \underset{y \in \mathcal{Y}}{\text{argmax}} \sum_{i=1}^{k} \mathbb{1}[y = y_i]$$

where $\mathcal{Y} = \{y_1, y_2, \ldots, y_k\}$ is the set of k sampled responses, generated as $y_i \sim p_{\mathcal{M}_\theta}(y|x, T)$ with temperature $T > 0$. Here, $\mathbb{1}[\cdot]$ is the indicator function used to identify the frequency of each response y^* within the sampled responses. The goal is to select the most frequently occurring response, which is considered the most consistent answer. Specifically, argmax finds the response y that maximizes the count of identical responses among the samples. To achieve this, the Self-Consistency algorithm first creates diverse solution attempts using temperature-controlled sampling. Then, it computes a similarity matrix $S \in \mathbb{R}^{k \times k}$, where each element S_{ij} represents the semantic similarity between responses y_i and y_j:

$$S_{ij} = \text{sim}(y_i, y_j)$$

This similarity can be quantified using various metrics, including string similarity, Levenshtein distance, or embedding-based cosine similarity, allowing for the identification of conceptually equivalent answers despite surface-level variations. Next, the framework employs a clustering algorithm with a predefined similarity threshold τ to group responses into clusters $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$, where $m \leq k$:

$$C_i = \{y_j \in \mathcal{Y} \mid \forall y_j, y_l \in C_i, S_{jl} \geq \tau\} \quad (1)$$

where C_i represents a cluster of responses, a subset of the sampled responses $\mathcal{Y}$, such that every pair of responses within C_i has a similarity score of τ or higher. To assess these clusters, the framework analyzes their statistical distribution by examining: (1) Cluster size: The number of responses in each cluster, $|C_i|$, which serves as the primary factor in determining the most frequent answer pattern. (2) Intra-cluster coherence: $\text{coh}(C_i) = \frac{1}{|C_i|(|C_i|-1)} \sum_{y_j, y_l \in C_i, j \neq l} S_{jl}$, measuring the internal consistency within each cluster and indicating the semantic closeness of responses beyond the similarity threshold. (3) Response quality metrics: Metrics like perplexity, entropy, and response length, which offer additional insights into the confidence and quality of individual responses within each cluster, contributing to a broader understanding of cluster reliability. While the final output selection in this basic formulation is determined by identifying the largest cluster based on cluster size, as formalized below:

$$y^* = \underset{C_i \in C}{\text{argmax}} \, (|C_i|)$$

the intra-cluster coherence and response quality metrics provide valuable supplementary information for analyzing the clusters and potentially refining the answer selection process in more advanced implementations. The overall process follows a pipeline of: (a) Stochastic sampling: $\mathcal{Y} = \{y_i \sim p_{\mathcal{M}_\theta}(y|x, T) \mid i \in \{1, 2, \ldots, k\}\}$, (b) Similarity computation: $S_{ij} = \text{sim}(y_i, y_j), \forall i, j \in \{1, 2, \ldots, k\}$, (c) Clustering: $\mathcal{C} = \text{cluster}(\mathcal{Y}, S, \tau)$, and (d) Statistical analysis: $y^* = \underset{C_i \in C}{\text{argmax}} |C_i|$. By emphasizing high-probability reasoning paths and de-emphasizing less common trajectories susceptible to errors, Self-Consistency effectively achieves a form of implicit ensemble learning within a single model's parameter space. This method leverages Shannon entropy minimization to filter out stochastic noise and converge on consistently correct answers. The entropy of the final distribution $H(p_{\mathcal{M}_\theta}(y|x, C))$, which represents the uncertainty in the model's output after applying Self-Consistency, is typically lower than the entropy of individual samples $H(p_{\mathcal{M}_\theta}(y|x))$. This reduction in entropy indicates that the probability distribution is more focused, ideally concentrating around the most consistent and correct answer, y^*. Furthermore, this technique inherently employs Weak-to-Strong Distillation by generating diverse outputs that represent different regions of the model's probability distribution, and subsequently refining the answer through consistency checks and majority voting to attain robust convergence on the most globally reliable solution.

A.9.2 Computational Time Complexity. : Self-consistency increases computational cost compared to standard language model inference, shifting from $O(n)$ to $O(k \times n + 2k^2)$. This complexity arises from:

$$\text{Time Complexity} = \underbrace{O(k \times n)}_{\text{Response Generation}} + \underbrace{O(k^2)}_{\text{Similarity Computation}} + \underbrace{O(\text{Clustering Algorithm Complexity})}_{\text{Clustering}}$$

Generating k responses contributes $O(k \times n)$, while pairwise similarity computation requires $O(k^2)$. The clustering complexity, denoted as O(Clustering Algorithm Complexity), depends on the specific algorithm used; a simplified approximation also yields $O(k^2)$. Thus, considering both similarity computation and clustering as potentially $O(k^2)$ operations, the overall time complexity is $O(k \times n + 2k^2)$. While in asymptotic notation $O(2k^2) = O(k^2)$, the final complexity of $O(k \times n + k^2)$ results in an increased computational cost compared to the $O(n)$ complexity of standard inference. This highlights the trade-off between computational cost and enhanced answer consistency.

A.9.3 ***Best-of-N Sampling Algorithm.*** : Best-of-N sampling [8] improves output quality by generating several candidate responses and selecting the highest-rated response using explicit quality assessment. This method creates diverse solution attempts via stochastic decoding with temperature-controlled sampling, then employs a systematic rating mechanism where the model evaluates each candidate on a numerical scale (0-10) based on specific quality criteria including clarity, accuracy, and helpfulness. Let $\mathcal{M}$ represent the language model, s be the system prompt, and x be the user query. The Best-of-N sampling procedure can be formalized as follows:

$$C = \{y_1, y_2, \ldots, y_k\} \quad \text{where} \quad y_i \sim \mathcal{M}(y|s, x, \tau_g)$$

Where, $C = \{y_1, y_2, \ldots, y_k\}$ is the set of k generated candidate responses. y_i represents the i-th candidate response, which is sampled from the language model $\mathcal{M}$. The sampling is conditioned on the system prompt s, the user query x, and the generation temperature τ_g.

$$r_i = \mathcal{M}(r|s_r, x, y_i, \tau_r) \quad \forall i \in \{1, 2, \ldots, k\}$$

Where, r_i is the rating assigned to the i-th candidate response y_i. This rating is generated by the same language model $\mathcal{M}$, but now acting as a rater. The rating is based on a specialized system prompt for rating s_r ("Rate the following response from 0-10 based on clarity, accuracy, and helpfulness. Respond with ONLY a number)"), the user query x, the candidate response y_i, and the rating temperature τ_r. The rating temperature τ_r is typically set to low values to ensure consistent evaluations.

$$y^* = \arg\max_{y_i \in C} r_i$$

y^* is the final selected response. It is chosen by finding the candidate response y_i from the set C that has the highest rating r_i. The framework implements a dual-role architecture where the model first functions as a generator producing multiple completions, then transitions to an evaluator by processing each completion with a specialized rating prompt. By filtering through multiple solution trajectories, Best-of-N sampling enhances output reliability and accuracy, reducing logical inconsistencies and factual errors that might appear in any single response. By leveraging the model's ability to generate and evaluate responses, the algorithm creates a robust internal quality control mechanism that enhances the reliability and accuracy of the final output. The approach leverages Weak-to-Strong Distillation principles by first generating multiple outputs of varying quality (the "weak" learning phase) and then using the model's own evaluation capabilities to identify and select the strongest output (the "strong" distillation phase). This creates a knowledge transfer process where weaker outputs inform the selection of the optimal solution.

A.9.4 Computational Time Complexity. Best-of-N sampling increases computational cost compared to standard language model inference, shifting from $O(n)$ to $O(k \times n)$. This complexity arises from the need to generate and evaluate k candidate responses. The time complexity can be broken down into the following components:

$$\text{Time Complexity} = \underbrace{O(k \times n)}_{\text{Response Generation}} + \underbrace{O(k \times n)}_{\text{Response Rating}} + \underbrace{O(k)}_{\text{Response Selection}}$$

Generating k candidate responses, each of average length n, contributes $O(k \times n)$. Subsequently, rating each of these k responses, which also involves a forward pass through the language model, adds another $O(k \times n)$ component. Finally, selecting the best response from the k rated responses based on their scores takes $O(k)$ time. Summing these components, the overall time complexity is $O(k \times n + k \times n + k) = O(2kn + k)$. In asymptotic notation, this simplifies to $O(k \times n)$, as the term k becomes less significant compared to kn when n is sufficiently large. This complexity highlights that the computational cost of Best-of-N sampling scales linearly with the number of candidate responses k, representing a trade-off for the enhanced output quality achieved through explicit response evaluation, yet remaining more computationally efficient in terms of asymptotic complexity compared to Self-Consistency which includes a quadratic component.

A.9.5 Comparing Best-of-N Sampling and Self-Consistency. While both Best-of-N Sampling and Self-Consistency enhance output quality by generating multiple responses, their core distinction lies in the answer selection mechanism. Best-of-N Sampling employs an explicit quality assessment: it leverages the language model itself to rate each generated candidate response based on defined criteria such as clarity, accuracy, and helpfulness. The response with the highest rating is then chosen as the final output. In contrast, Self-Consistency utilizes an implicit evaluation approach. It focuses on identifying the most consistent reasoning pattern across the generated responses through similarity clustering. By grouping semantically similar outputs and selecting the most frequent cluster, Self-Consistency implicitly evaluates responses based on their agreement with each other, without requiring explicit quality ratings for each individual response. Thus, Self-Consistency measures conceptual consensus among multiple reasoning paths, whereas Best-of-N directly assesses the quality of each individual output. This fundamental difference underscores two distinct strategies for enhancing LLM output quality: direct, model-driven quality evaluation of individual responses versus statistical validation through inter-response agreement.

A.9.6 ***Chain-of-Thought with Reflection.*** : Chain-of-Thought with Reflection [45, 62] enhances reasoning capabilities by structuring the problem-solving process into distinct conceptual phases

Feature	Self-Consistency	Best-of-N Sampling
Selection Method	Majority clustering + statistical analysis	Explicit self-evaluation
Quality Assessment	Implicit through similarity & frequency	Direct scoring system (0-10)
Computational Overhead	$O(k \times n + k^2)$ (clustering is costly)	$O(k \times n)$ (single pass rating)
Weak-to-Strong Distillation	Yes (reinforces high-probability reasoning paths)	Yes (filters weak outputs via scoring)
Error Handling	Reduces stochastic noise via statistical convergence	Mitigates low-quality outputs with explicit filtering

Table 15: Comparison of Self-Consistency and Best-of-N Sampling

that emulate human cognitive processes. This approach decomposes the reasoning task into three sequential components within a single generative process. Let $\mathcal{M}_\theta$ denote a language model with parameters θ, and let q represent an input query. We formalize the Chain-of-Thought with Reflection process as follows:

$$R = \mathcal{M}_\theta(P(q)),$$

where R is the model's response generated using a structured prompt $P(q)$. While the response is generated in a single forward pass, it can be conceptually decomposed into three functional components:

$$R = [R_\mathcal{T}, R_\mathcal{R}, R_O],$$

where: $R_\mathcal{T}$ represents the systematic decomposition of the problem (thinking phase), $R_\mathcal{R}$ denotes the critical assessment of the initial analysis (reflection phase), and R_O is the integration of reasoning into a cohesive solution (output phase). The structured prompt $P(q)$ is constructed to guide this decomposition:

$$P(q) = \Phi(q, \tau),$$

where Φ is the prompt engineering function, and τ is a template specifying the expected structure. This template encodes phase-specific instructional priors that guide the model to produce each component with distinct reasoning objectives. Though generated in a single forward pass, each component can be conceptually viewed as being influenced by the preceding components, which we represent as conditional distributions:

$$\begin{aligned} p(R_\mathcal{T}|q) &\approx p(R_\mathcal{T}|q, \tau_\mathcal{T}), \\ p(R_\mathcal{R}|q, R_\mathcal{T}) &\approx p(R_\mathcal{R}|q, R_\mathcal{T}, \tau_\mathcal{R}), \\ p(R_O|q, R_\mathcal{T}, R_\mathcal{R}) &\approx p(R_O|q, R_\mathcal{T}, R_\mathcal{R}, \tau_O), \end{aligned}$$

where $\tau_\mathcal{T}$, $\tau_\mathcal{R}$, and τ_O are the phase-specific instructional priors embedded in the template. The probability of generating the full response can be expressed as:

$$p(R|q) = p(R_\mathcal{T}|q) \cdot p(R_\mathcal{R}|q, R_\mathcal{T}) \cdot p(R_O|q, R_\mathcal{T}, R_\mathcal{R})$$

This structured decomposition implements a form of guided reasoning through explicit metacognitive phases. The key insight is that while $\mathcal{M}_\theta$ remains fixed, the structured prompt effectively guides the model's reasoning process by encouraging it to follow distinct cognitive phases within a single generation. See Algorithm 3 for details.

A.9.7 Computational Time Complexity. Chain-of-Thought with Reflection achieves enhanced reasoning with minimal computational overhead. Since the entire process—including structured thinking, reflection, and output—is generated in a single forward pass through the language model, the dominant computational cost remains that of standard inference. This results in a complexity of $O(n)$, where n is the length of the generated response. However, if reflection introduces an iterative refinement mechanism (e.g., regenerating based on self-evaluation), the complexity could increase depending on the number of iterations. In such cases, the worst-case complexity becomes $O(r \cdot n)$, where r is the number of refinement steps. The trade-off is that additional refinement may improve output quality at the cost of higher computational demand. Therefore, in its simplest form, the overall computational complexity remains $O(n)$, comparable to standard inference, while providing enhanced reasoning capabilities. In iterative settings, complexity scales proportionally to the number of refinement steps, requiring careful tuning to balance reasoning depth and efficiency.

A.9.8 ***Entropy-Guided Decoding.*** : Entropy-Guided Decoding [12, 37, 59] enhances language model outputs by dynamically adjusting sampling parameters based on uncertainty metrics. Traditional approaches use fixed parameters throughout generation, but our method adapts in real-time to each token's context. In our notation, we represent the sequence of tokens generated up to the current generation step t as $\mathbf{x} = (x_1, x_2, \ldots, x_t)$, where each token belongs to a vocabulary of size V. At each generation step, the language model produces logits $\mathbf{l}_t \in \mathbb{R}^V$, which are the unnormalized prediction scores for the next token, and attention weights $A_t \in \mathbb{R}^{L \times H \times S \times S}$, where L is the number of transformer layers, H is the number of attention heads per layer, and S is the sequence length. These attention weights represent how much each token attends to other tokens in the sequence, with $A_t^{l,h,i,j}$ indicating how much token i attends to token j in head h of layer l. We first compute token probabilities from the logits using the softmax function:

$$\begin{aligned} p_t &= \text{softmax}(\mathbf{l}_t) \\ \log p_t &= \log \text{softmax}(\mathbf{l}_t) \end{aligned}$$

Here, $p_t \in \mathbb{R}^V$ represents the probability distribution over all tokens in the vocabulary, with $p_t(v)$ indicating the probability of token v. (a) The Shannon entropy of this token distribution quantifies uncertainty in next-token selection, which we normalize by $\ln(2)$ to express entropy in bits, providing a more interpretable scale:

$$\mathcal{H}(p_t) = -\sum_{v=1}^{V} p_t(v) \log_2 p_t(v)$$

Entropy is a fundamental measure of uncertainty; higher entropy values (approaching $\log_2 V$) indicate that the model is uncertain about which token to generate next, distributing probability more evenly across many tokens. Conversely, values near zero suggest the model is highly confident, concentrating probability on one or

Algorithm 3 Chain-of-Thought(CoT) with Reflection

procedure CoT-Reflection($q, \mathcal{M}_\theta$)
$\tau \leftarrow$ ConstructTemplate() ▷ Create structured reasoning template with phase markers for thinking, reflection, and output
$P(q) \leftarrow \Phi(q, \tau)$ ▷ Construct prompt with query q and template τ
$R \leftarrow \mathcal{M}_\theta(P(q))$ ▷ Generate complete response in a single forward pass
$R_O \leftarrow$ ExtractOutput(R) ▷ Extract final output component R_O

return R_O ▷ Return the final output
end procedure

few tokens. The variance entropy (varentropy) is a complementary metric that captures the spread of log-probabilities around the mean entropy:

$$\mathcal{V}(p_t) = \sum_{v=1}^{V} p_t(v) \left(\log_2 p_t(v) + \mathcal{H}(p_t)\right)^2$$

(b) Varentropy helps distinguish between distributions with similar entropy but different shapes; higher varentropy indicates a "peakier" distribution with a few high-probability tokens amidst many low-probability ones, which can suggest that the model is considering multiple distinct possibilities rather than being genuinely uncertain across the entire vocabulary. We derive attention-based uncertainty metrics from the refined attention patterns encoded in $\mathbf{A}_t^L \in \mathbb{R}^{H \times S \times S}$, the final layer's attention weights. (c) The attention entropy measures how uniformly attention is distributed across the sequence:

$$\mathcal{H}_{\text{attn}}(A_t^L) = -\sum_{h=1}^{H}\sum_{i=1}^{S}\sum_{j=1}^{S} A_t^{L,h,i,j} \log_2 A_t^{L,h,i,j}$$

High attention entropy indicates diffuse attention patterns, suggesting the model is uncertain about which parts of the context are relevant for generating the next token. Low values suggest focused attention on specific context tokens, indicating higher confidence in the relevance of those tokens. (d) The attention variance entropy quantifies how consistently different attention heads focus on the same parts of the input:

$$\mathcal{V}_{\text{attn}}(A_t^L) = \text{Var}_{h \in [1,H]}(\mathcal{H}_{\text{attn}}(A_t^{L,h}))$$

Here, $\mathcal{H}_{\text{attn}}(A_t^{L,h})$ is the entropy of attention weights for head h, and Var denotes variance. This metric captures disagreement between attention heads, with higher values indicating that different heads are focusing on different aspects of the input, suggesting multi-faceted uncertainty. We also introduce two consistency metrics to capture attention patterns more comprehensively. (e) The agreement metric α_t measures how consistently different attention heads focus on the same tokens:

$$\bar{A}_t^L = \frac{1}{H}\sum_{h=1}^{H} A_t^{L,h}$$

$$\alpha_t = \mathbb{E}_{h \in [1,H]}\left[\|A_t^{L,h} - \bar{A}_t^L\|_1\right]$$

where $\bar{A}_t^L$ is the mean attention pattern across all heads, and $\|\cdot\|_1$ denotes the L1 norm (sum of absolute differences). Lower α_t values indicate high agreement among attention heads, suggesting model confidence in its understanding of the relevant context. Higher values suggest disagreement, indicating uncertainty about which contextual elements are most important. (f) The interaction strength γ_t quantifies the intensity of attention activations:

$$\gamma_t = \mathbb{E}_{h,i,j}\left[\left|\log A_t^{L,h,i,j}\right|\right]$$

where $\mathbb{E}_{h,i,j}[\cdot]$ denotes the expectation (average) over all heads, query positions, and key positions. Higher γ_t values indicate stronger, more defined attention patterns, suggesting the model has formed clearer associations between tokens. These metrics collectively inform our adaptive parameter selection function Φ, which adjusts four key sampling parameters based on observed uncertainty:

$$(\tau_t, p_t^{\text{top}}, k_t, p_t^{\text{min}}) = \Phi\big(\mathcal{H}(p_t), \mathcal{V}(p_t), \mathcal{H}_{\text{attn}}(A_t^L), \mathcal{V}_{\text{attn}}(A_t^L), \alpha_t, \gamma_t\big)$$

(i) The temperature parameter τ_t controls the sharpness of the probability distribution before sampling; higher temperatures make the distribution more uniform (increasing randomness), while lower temperatures make it more peaked (increasing determinism). We adapt it based on token and attention uncertainties:

$$\tau_t = \tau_0 \cdot \text{clip}\Big(1 + \beta_1(\mathcal{H}(p_t) + \mathcal{V}(p_t)) + \beta_2 \mathcal{H}_{\text{attn}}(A_t^L) - \beta_3 \alpha_t, \tau_{\min}, \tau_{\max}\Big)$$

(ii) The top-p (nucleus sampling) threshold p_t^{top} restricts sampling to the smallest set of tokens whose cumulative probability exceeds this threshold, effectively removing unlikely tokens from consideration. We adapt it primarily based on attention head disagreement:

$$p_t^{\text{top}} = p_0^{\text{top}} \cdot \text{clip}\left(1 + \beta_4 \mathcal{V}_{\text{attn}}(A_t^L), p_{\min}^{\text{top}}, 1.0\right)$$

(iii) The top-k filtering parameter k_t restricts sampling to the k_t most probable tokens, providing a hard limit on the token candidates. We adjust it based on attention consistency and strength:

$$k_t = \text{clip}\left(\lfloor k_0 \cdot (1 + \beta_5 \gamma_t - \beta_6 \alpha_t) \rceil, 1, k_{\max}\right)$$

(iv) The minimum probability threshold p_t^{min} filters out tokens with probability below $p_t^{\text{min}} \cdot \max_v p_t(v)$ relative to the most probable token, providing another way to eliminate unlikely candidates. We adapt it based on token uncertainty:

$$p_t^{\text{min}} = p_0^{\text{min}} \cdot \text{clip}\left(1 - \beta_7(\mathcal{H}(p_t) + \mathcal{V}(p_t)), p_{\min}^{\text{min}}, p_{\max}^{\text{min}}\right)$$

where $\tau_0, p_0^{\text{top}}, k_0, p_0^{\text{min}}$ are the base parameter values used when uncertainty metrics are neutral (default sampling behavior), $\beta_{1\ldots7}$ are hyperparameters controlling the influence of each uncertainty metric, $\text{clip}(x, \min, \max)$ constrains value x to the range $[\min, \max]$,

and $\lfloor x \rceil$ represents rounding to the nearest integer (for k_t). The intuition behind our parameter adjustments is rooted in uncertainty: high token distribution or attention entropy (uncertainty) prompts increased temperature for broader exploration. Attention head disagreement (high attention varentropy) leads to a wider top-p sampling to include more candidates. Strong attention patterns with moderate agreement (high interaction strength) expand top-k selection for a more diverse set of top tokens. Elevated token uncertainty lowers the minimum probability threshold, preventing exclusion of potentially valid but less probable tokens. This dynamic adaptation enhances generation quality across contexts without specialized tuning. In precision-demanding contexts, uncertainty metrics naturally guide conservative sampling; in creative settings, they enable greater exploration. By linking sampling parameters to the model's uncertainty assessment, we achieve a principled balance between diversity and coherence, surpassing static parameter approaches. Entropy-guided decoding thus refines language model outputs by dynamically adjusting sampling parameters based on real-time uncertainty. This method calculates token and attention-based metrics during generation, adapting temperature, top-p, top-k, and minimum probability threshold. This allows for exploration when uncertain and precision when confident, all with minimal inference overhead.

A.9.9 Computational Time Complexity Analysis. The computational complexity of entropy-guided decoding per token generation step is determined by several key operations. Calculating token distribution uncertainty metrics (entropy and varentropy) from the vocabulary logits requires $O(V)$ operations, where V is the vocabulary size. The computation of attention-based uncertainty metrics, which analyze the model's attention patterns, contributes $O(L \cdot H \cdot S^2)$ complexity. This arises from processing the attention weights across L transformer layers, H attention heads, and sequence length S. Adapting the sampling parameters based on these metrics involves simple arithmetic and has a negligible $O(1)$ time cost. The token sampling process, including steps like top-k or top-p filtering, adds $O(V \log V)$ complexity due to sorting operations required to filter the vocabulary distribution. Therefore, the overall per-token computational complexity is dominated by the sum of these factors, approximately $O(V \log V + L \cdot H \cdot S^2)$. Consequently, for generating a text sequence of length T, the total computational complexity becomes $O(T \cdot (V \log V + L \cdot H \cdot S^2))$. For typical Large Language Models and longer text sequences, the term $O(L \cdot H \cdot S^2)$ associated with attention processing and uncertainty metric calculations often represents the most significant portion of the computational cost per token.

A.9.10 ***Chain-of-Thought (CoT) Decoding.*** : Chain-of-Thought (CoT) Decoding [45, 47] is a multi-path inference technique designed to enhance the reliability and logical coherence of language model outputs. Unlike conventional decoding methods that generate a single response, CoT Decoding explores a set of potential reasoning trajectories in parallel. This approach leverages a path management framework to generate, evaluate, and select from a diverse set of candidate responses, ultimately aiming for outputs grounded in more robust reasoning processes. The CoT Decoding process begins with the initiation of multiple reasoning paths. Given an input context c, the language model $\mathcal{M}$ first computes the probability distribution over the vocabulary $\mathcal{V}$ for the first token position. This distribution, $P(x_1|c)$, is derived from the logits (pre-softmax scores) $\mathbf{l}_1 \in \mathbb{R}^{|\mathcal{V}|}$ produced by the model for the first token position. The probability distribution is typically obtained via a softmax function with a temperature parameter T:

$$P(x_1|c) = \text{softmax}(\mathbf{l}_1/T)$$

Here, $x_1 \in \mathcal{V}$ represents a token from the vocabulary, and $P(x_1|c)$ denotes the probability of x_1 being the first token in the response, conditioned on the input context c. To initiate diverse reasoning paths, the system samples the top-k tokens with the highest probabilities from $P(x_1|c)$. Let $\mathcal{T} = \{t_1, t_2, \ldots, t_k\}$ be the set of these top-k tokens. For each initial token $t_i \in \mathcal{T}$, the model generates a complete response sequence, resulting in a set of k candidate paths $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$. Each path $P_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,n_i})$ represents a complete sequence of tokens, where $x_{i,1} = t_i$ and n_i is the length of path P_i. A core component of CoT Decoding is the reliability scoring mechanism. This mechanism evaluates the confidence in token selections within each path. For each token $x_{i,j}$ at position j in path P_i, with corresponding logits $\mathbf{l}_{i,j}$, a token-level reliability score $r(x_{i,j})$ is computed. Let $p_{i,j}^{(1)}$ and $p_{i,j}^{(2)}$ be the probabilities of the most and second most likely tokens at position j in path P_i, respectively, obtained after applying the softmax function to $\mathbf{l}_{i,j}$. The token reliability score is defined as:

$$r(x_{i,j}) = (p_{i,j}^{(1)} - p_{i,j}^{(2)}) \cdot f(j)$$

where $f(j)$ is a position-based damping function designed to emphasize the reliability of earlier tokens in the sequence. A common form for $f(j)$ is a linearly decreasing function:

$$f(j) = 1 - \alpha \cdot \frac{j}{L_i}$$

Here, L_i is the maximum sequence length considered for path P_i, and $\alpha \in [0, 1]$ is a damping coefficient that controls the rate of decrease in reliability weight with position. The overall reliability $R(P_i)$ of a path P_i is calculated as a weighted average of its token-level reliability scores. Let w_j be position-dependent weights that further emphasize earlier tokens. The path reliability is given by:

$$R(P_i) = \frac{\sum_{j=1}^{n_i} r(x_{i,j}) \cdot w_j}{\sum_{j=1}^{n_i} w_j}$$

In scenarios where multiple reasoning paths may lead to semantically similar responses, CoT Decoding can incorporate a path consolidation mechanism. This process groups paths that exhibit high textual similarity, typically measured using sequence comparison techniques. For each group of similar paths, the path with the highest reliability score is selected as a representative of that group. Finally, the system selects the output response. In scenarios without path consolidation, the path with the highest overall reliability is chosen as the final output:

$$P^* = \arg\max_{P_i \in \mathcal{P}} R(P_i)$$

When path consolidation is enabled, the selection is performed among the representatives of the consolidated path groups, again choosing the one with the highest reliability. By exploring multiple reasoning paths and employing a reliability-based selection process, Chain-of-Thought Decoding aims to generate responses that are not only probable but also more logically consistent and reliably

reasoned. This method effectively addresses uncertainty by systematically exploring and evaluating different reasoning trajectories, ensuring that the final output is grounded in a well-supported and coherent line of reasoning.

A.9.11 Computational Time Complexity Analysis. : CoT Decoding's complexity is primarily determined by k (initial paths) and L (sequence length). Initial path expansion via a forward pass on input context c (length n) to compute $P(x_1|c)$ contributes $O(n \cdot h)$, where h is the hidden dimension. Top-k token selection $\mathcal{T} \subset \mathcal{V}$ (vocabulary size V) adds $O(V \log k)$. Sequence generation for k paths $P_i \in \mathcal{P}$ up to length L incurs $O(k \cdot L \cdot h)$, considering $O(h)$ per-token cost. Reliability scoring for $k \cdot L$ tokens adds $O(k \cdot L)$ overhead. Path consolidation, involving pairwise comparisons of k paths $\mathcal{P}$, requires $O(k^2 \cdot \text{sim}(L)) \approx O(k^2 \cdot L)$. Thus, CoT Decoding's overall time complexity, dominated by generation and consolidation, is approximately $O(n \cdot h + V \log k + k \cdot L \cdot h + k^2 \cdot L)$, simplifying to $O(k \cdot L \cdot h + k^2 \cdot L)$ for large k and L. This highlights the computational cost for enhanced reasoning via multi-path exploration.

A.9.12 ***RE^2 (Re-Reading and Re-Analyzing).*** : The RE^2 framework is an advanced reasoning methodology designed to enhance the performance of language models on complex tasks. Drawing inspiration from human cognitive processes, this framework structures reasoning into explicit phases, facilitating a more thorough analysis of input queries. Unlike traditional language model inference, where a model $\mathcal{M}$ with parameters θ directly processes an input query x to generate a response y, expressed as: $y = \mathcal{M}_\theta(x)$, the RE^2 framework introduces a structured approach. It refines the generation process by decomposing reasoning into three distinct steps, transforming the input query x into a composite prompt structure, P_{RE^2}. The response generation in RE^2 is then formulated as: $y_{RE^2} = \mathcal{M}_\theta(P_{RE^2})$, where P_{RE^2} is constructed by concatenating several components:

$$P_{RE^2} = P_{sys} \oplus P_{init}(x) \oplus P_{reread}(x) \oplus P_{synth}$$

Here, P_{sys} represents optional system instructions, and $\oplus$ denotes concatenation. The framework incorporates three key reasoning phases, represented by $P_{init}(x)$, $P_{reread}(x)$, and $P_{synth}(x)$. The first step, $P_{init}(x)$, prompts the model to carefully comprehend the input query:

$$\begin{aligned} P_{init}(x) = &\text{"Step 1 - Initial Reading: Let's first} \\ &\text{read and understand the question carefully."} \\ &\oplus \text{"Original Question: "} \oplus x \end{aligned}$$

The next step, $P_{reread}(x)$, instructs the model to revisit the query for structured decomposition and analysis:

$$\begin{aligned} P_{reread}(x) = &\text{"Step 2 - Re-reading and Analysis:} \\ &\text{Let's read the question again: } \oplus x \\ &\oplus \text{"Now, let's break down what the question} \\ &\text{is asking and analyze its key components."} \end{aligned}$$

Finally, P_{synth} guides the model to synthesize a response based on insights from the previous steps:

$$\begin{aligned} P_{synth} = &\text{"Step 3 - Final Answer: Based on our analysis,} \\ &\text{here is the complete answer:"} \end{aligned}$$

The RE^2 framework incorporates parameters to regulate the response generation process. The temperature parameter, T, modifies the output probability distribution, given by:

$$P_T(y|P_{RE^2}) = \frac{\exp(\text{logit}(y)/T)}{\sum_{y' \in V} \exp(\text{logit}(y')/T)}$$

where y represents output tokens, V is the vocabulary space, and $\text{logit}(y)$ is the unnormalized score for token y. To refine token selection, nucleus sampling (top-p sampling) is applied. It limits the vocabulary to a subset V_p (the nucleus), defined as:

$$V_p = \min\{V' \subseteq V \mid \sum_{y \in V'} P_T(y|P_{RE^2}) \geq p\}$$

such that the cumulative probability of selected tokens exceeds a predefined threshold p. The final sampling distribution is then computed as:

$$P_{final}(y|P_{RE^2}) = \begin{cases} \frac{P_T(y|P_{RE^2})}{\sum_{y' \in V_p} P_T(y'|P_{RE^2})}, & \text{if } y \in V_p \\ 0, & \text{otherwise} \end{cases}$$

ensuring that tokens are sampled only from within the nucleus V_p, with their probabilities rescaled to sum to one, thereby eliminating low-probability tokens. By integrating temperature scaling and nucleus sampling, the RE^2 framework balances determinism and diversity in text generation. Its structured approach mirrors deliberate human analysis, fostering a more comprehensive exploration of the problem before generating a response. This makes RE^2 particularly advantageous for complex reasoning tasks.

A.9.13 Computational Time Complexity Analysis. The computational complexity of the RE^2 framework is primarily dictated by the transformer's self-attention mechanism operating over the constructed prompt P_{RE^2}, which has length m (linearly related to the original query length n). This self-attention mechanism imposes a quadratic cost, specifically $O(m^2 \cdot d)$, where d represents the model's hidden dimension. Although the process of constructing the prompt and the subsequent token sampling (which includes techniques like temperature scaling and nucleus sampling) introduce some additional computational overhead, these factors are relatively minor compared to the dominant quadratic cost. Thus, while RE^2 maintains the single forward pass characteristic of standard transformer-based inference, it does so at the expense of processing a longer, more structured prompt, resulting in a higher constant factor in runtime.

A.9.14 ***Mixture of Agents.*** : The Mixture of Agents (MoA)[1, 42] framework enhances the quality of language model responses through candidate generation, critique, and synthesis. Let M denote a pre-trained language model with trainable parameters θ. Given an input query q and system context s, the MoA process consists of the following stages. In the initial stage, a set of n diverse candidate responses, denoted as $Y = y_1, y_2, \ldots, y_n$, is generated. Each response y_i is sampled from the conditional probability distribution of the language model M, parameterized by θ, given the query q, system context s, and a generation temperature T_1:

$$\begin{aligned} &Y = \{y_1, y_2, \ldots, y_n\}, \\ &\text{where} \quad y_i \sim p_M(y|q, s; \theta, T_1), \quad \forall i \in \{1, 2, \ldots, n\} \end{aligned}$$

Feature	Entropy-Guided Decoding	Chain-of-Thought Decoding
Approach	Dynamically adjusts token sampling based on uncertainty metrics from logits and attention.	Generates multiple reasoning paths from diverse initial tokens, then scores and consolidates for best output.
Core Mechanism	Adapts parameters (temperature, top-p, top-k, min probability) using logits entropy/varentropy and attention entropy/varentropy, agreement, and interaction strength.	Scores reliability using top probability differences and position damping to assess path quality, optionally merges paths before selection.
Focus	Adaptive sampling balancing exploration and precision by reducing uncertainty.	Multi-path exploration to enhance logical coherence and output reliability.
Strength	Dynamically modulates parameters based on context confidence, for flexible application.	Synthesizes multiple paths to overcome errors and produce robust and coherent output.
Primary Goal	Minimize generation uncertainty while balancing diversity and determinism.	Maximize reasoning quality and consistency by selecting the best path.

Table 16: Comparison of Entropy-Guided Decoding and Chain-of-Thought Decoding

where Y is the set of candidate responses, y_i is the i-th candidate response, n is the number of generated responses (a hyperparameter), $p_M(y|q,s;\theta,T)$ represents the conditional probability distribution of the language model, and T_1 controls the stochasticity and diversity of responses, with higher values promoting greater diversity. A critique function C evaluates the candidate responses Y in the context of the original query q and system context s. For this, we utilize the same language model M to generate a critique c based on a conditional probability distribution with temperature T_2:

$$c = C(Y, q, s; \theta) \sim p_M(c|Y, q, s; \theta, T_2)$$

where $C(Y, q, s; \theta)$ is the critique function evaluating Y, c represents the generated critique, and T_2 is set lower than T_1 to ensure a more discerning evaluation. The final response y^* is synthesized using the critique c, query q, and system context s. A synthesis function S, also utilizing the language model M, generates y^* under a temperature T_3:

$$y^* = S(c, q, s; \theta) \sim p_M(y|c, q, s; \theta, T_3)$$

where $S(c, q, s; \theta)$ generates the refined response, y^* is the synthesized response, and T_3 is set lower than T_2 to encourage precise and focused refinement. A post-processing function Φ further refines the synthesized response to remove meta-content, artifacts, and formatting inconsistencies. The final output is denoted as y_{final}:

$$y_{final} = \Phi(y^*) = \Phi(S(C(y_{i}{}_{i=1}^{n}, q, s; \theta), q, s; \theta))$$

where $\Phi(y^*)$ processes the synthesized response, and y_{final} is the final enhanced response. The MoA framework employs a temperature scheduling strategy to control the refinement process:

$$T_1 > T_2 > T_3$$

This descending order encourages diversity in generation (T_1), balanced critique evaluation (T_2), and precise synthesis (T_3). Regularization techniques improve response quality by penalizing redundancy during generation:

$$p_M(y|x;\theta,T,\lambda) \propto p_M(y|x;\theta,T) \cdot R(y,\lambda)$$

where x represents either the query q or a combination of inputs depending on the stage, $\propto$ denotes proportionality, and $R(y,\lambda)$ is a regularization function controlling repetition, ensuring varied and high-quality responses. For practical implementation, parameters that apply a penalty for token repetition and prevent n-gram sequence repetition implicitly implement the regularization function $R(y,\lambda)$ during text generation by modifying the language model's probability distribution to reduce repetitive token and n-gram sequences, and effectively control the strength and type of regularization applied In summary, the MoA framework iteratively refines responses by first generating diverse candidate responses, critically evaluating them, and synthesizing an improved output. The structured use of temperature cascade and regularization enhances response quality beyond single-pass generation approaches.

A.9.15 Computational Time Complexity Analysis. The computational complexity of the Mixture of Agents (MoA) framework is substantially higher than standard single-pass generation due to its multi-stage process. The dominant computational cost arises from the transformer model's self-attention mechanism, leading to a per-token complexity that scales at least linearly, and potentially quadratically, with the generated sequence lengths: L (average length of candidate responses), L_c (length of the critique), and L^* (length of the final synthesized response). The complexity is also directly proportional to the model's hidden dimension (d). Generating n candidate responses increases this cost, making candidate generation the most computationally intensive stage, with an approximate complexity of $O(n \cdot L^2 \cdot d)$ or $O(n \cdot L \cdot S_{max} \cdot d)$, where S_{max} represents the maximum sequence length. The critique and synthesis stages further contribute to the total computational demand, making MoA significantly more resource-intensive compared to single-pass inference. However, parallelization, such as distributed GPU inference, can mitigate latency in candidate generation while maintaining the overall computational workload.

A.9.16 ***Reimplementation Then Optimize (RTO).*** : We introduce Reimplementation Then Optimize (RTO), a novel multi-stage framework designed to enhance the quality of solutions generated

by large language models (LLMs). By decomposing the generation process into discrete stages—implementation, analysis, reimplementation, and synthesis—RTO achieves significant improvements in correctness, consistency, and optimization compared to single-pass generation methods. The framework leverages iterative refinement to progressively improve solution quality through multiple generative passes. Let $\mathcal{M}$ denote the language model and q represent the initial problem specification. The RTO process is formalized as follows:

$$c_1 = \mathcal{M}(s, q_{\text{augmented}}) \tag{2}$$

$$r = \mathcal{M}(s, c_1, q_{\text{analysis}}) \tag{3}$$

$$c_2 = \mathcal{M}(s, r) \tag{4}$$

$$c_{\text{opt}} = \begin{cases} c_1 & \text{if } \delta(c_1, c_2) \geq \tau \\ \mathcal{M}(s, c_1, c_2, q) & \text{otherwise} \end{cases} \tag{5}$$

In Stage 1 (Equation 2), the language model $\mathcal{M}$ generates an initial solution c_1 based on a system prompt s (which provides instructions to guide the model's behavior) and an augmented query $q_{\text{augmented}}$ (the initial query q augmented with instructions for generating high-quality output). Stage 2 (Equation 3) involves the model $\mathcal{M}$ analyzing the initial solution c_1 along with the system prompt s and an analysis query q_{analysis} (a prompt designed to extract requirements), resulting in the extracted specification r. In Stage 3 (Equation 4), the model $\mathcal{M}$ produces an independent solution c_2 based on the extracted specification r and the system prompt s. Finally, in Stage 4 (Equation 5), the framework determines the optimized solution c_{opt}. This is achieved by comparing the initial solution c_1 and the reimplemented solution c_2 using a similarity function $\delta(c_1, c_2)$ and a consistency threshold τ. If the similarity exceeds the threshold, c_{opt} is set to c_1; otherwise, $\mathcal{M}$ synthesizes a new optimized solution c_{opt} from s, c_1, c_2, and q. The effectiveness of RTO is quantified by the quality improvement ΔQ, defined as:

$$\Delta Q = Q(c_{\text{opt}}) - Q(c_1) \tag{6}$$

Equation 6 measures the improvement in quality ΔQ as the difference between the quality metric Q of the optimized solution c_{opt} and the initial solution c_1. Here, Q represents a domain-specific quality metric that encompasses aspects such as correctness, efficiency, and other relevant criteria.

A.9.17 Computational Time Complexity Analysis. The computational complexity of RTO is given by: $T_{\text{RTO}} = \sum_{i=1}^{n} (\mathcal{M}, l_i)$, where $T(\mathcal{M}, l_i)$ denotes the time complexity for the language model $\mathcal{M}$ to generate a sequence of length l_i in the i-th step. For Transformer-based LLMs, the per-step complexity $T(\mathcal{M}, l_i)$ is dominated by the self-attention mechanism and scales approximately as $O(l_i^2 \cdot d)$, where d represents the model dimension. Consequently, the total complexity of RTO, T_{RTO}, is the sum of these per-step costs across its n stages.

A.9.18 ***PlanSearch***. : We present a novel multi-step planning and search (PlanSearch [41]) framework for general language tasks that leverages LLMs to decompose complex queries through iterative abstraction and refinement. Our approach formalizes the response generation as a structured sequence of transformations that progressively refine the understanding of the query before producing a final response. Let us define a query as $Q \in \mathcal{Q}$, where $\mathcal{Q}$ represents the space of all possible queries, each encapsulating the query, contextual requirements, and constraints. We aim to find an optimal answer $a^* \in \mathcal{A}$, where $\mathcal{A}$ is the answer space. The process is decomposed into intermediate representations through multiple transformation phases, mediated by a system prompt Ψ that provides high-level guidance to the model. Given a question Q and system prompt Ψ, we define the following transformation sequence:

$$O_1 = f_{\text{obs}}(Q, \Psi, n_1) \tag{7}$$

$$O_2 = f_{\text{derive}}(Q, \Psi, O_1, n_2) \tag{8}$$

$$O = O_1 \cup O_2 \tag{9}$$

$$\sigma = f_{\text{strategy}}(Q, \Psi, O) \tag{10}$$

$$a = f_{\text{answer}}(Q, \Psi, \sigma) \tag{11}$$

Here, $O_1 = \{o_1, o_2, \ldots, o_{n_1}\}$ comprises n_1 initial observations about the question Q, while $O_2 = \{o_{n_1+1}, o_{n_1+2}, \ldots, o_{n_1+n_2}\}$ represents n_2 derived observations. The union of these sets is denoted as O. The symbol σ represents the reasoning strategy derived from Q and O, while a denotes the final answer derived from Q and σ. The transformation functions f_{obs}, f_{derive}, f_{strategy}, and f_{answer} play distinct roles: f_{obs} generates initial insights by identifying key components of the question, such as entities, relationships, and constraints; f_{derive} synthesizes deeper observations by connecting these components and inferring implicit knowledge; f_{strategy} formulates a reasoning strategy to address the question systematically; and f_{answer} produces a final, well-structured answer based on the reasoning strategy. Each transformation function f_i is realized through a pretrained language model $\mathcal{M}$ with parameters θ and a task-specific prompt template τ_i:

$$f_i(Q, \Psi, x_1, x_2, \ldots, x_n) = \mathcal{M}(\Psi \oplus \tau_i(Q, x_1, x_2, \ldots, x_n); \theta)$$

where $\mathcal{M}$ represents the pretrained language model, θ denotes its parameters, τ_i is a task-specific prompt template, and $\oplus$ represents the concatenation operation. The variables $x_1, x_2, \ldots, x_n$ represent function-specific inputs, such as the question or previously generated observations. To enhance answer diversity and quality, we generate multiple candidate answers by introducing stochasticity through temperature sampling:

$$A = \{a_1, a_2, \ldots, a_N\} = \{f_{\text{solve}}(Q, \Psi; T)\}_{i=1}^{N} \tag{12}$$

Here, T represents the temperature parameter controlling generation diversity, N denotes the number of answers generated, and f_{solve} is the complete solution pipeline executing all transformation phases. This approach allows exploration of different reasoning paths and answer formulations for a given question. The decomposition offers several advantages: it activates relevant parametric knowledge by identifying key components and relationships in the question, enables compositional reasoning through derived observations, provides guided answer generation via explicit reasoning strategies, and enhances explainability through a traceable reasoning chain from question to answer. The multi-stage process mirrors human-like reasoning strategies, systematically breaking down complex questions before generating answers, resulting in responses that are both accurate and interpretable.

A.9.19 Time Complexity Analysis. The time complexity of PlanSearch is determined by the sequential execution of its transformation functions through a transformer-based language model $\mathcal{M}$ with parameters θ. For transformer architectures, processing inputs requires $O(L_i^2)$ complexity due to self-attention, while generating outputs adds $O(L_o \cdot L_i)$ complexity, where L_i and L_o represent input and output lengths respectively. For each transformation function, the time complexity can be expressed as:

$$\begin{aligned} f_{\text{obs}} &: O\Big((|\Psi| + |Q|)^2 \cdot |\theta| + \\ &\quad |O_1| \cdot (|\Psi| + |Q|) \cdot |\theta|\Big) \\ f_{\text{derive}} &: O\Big((|\Psi| + |Q| + |O_1|)^2 \cdot |\theta| + \\ &\quad |O_2| \cdot (|\Psi| + |Q| + |O_1|) \cdot |\theta|\Big) \\ f_{\text{strategy}} &: O\Big((|\Psi| + |Q| + |O|)^2 \cdot |\theta| + \\ &\quad |\sigma| \cdot (|\Psi| + |Q| + |O|) \cdot |\theta|\Big) \\ f_{\text{answer}} &: O\Big((|\Psi| + |Q| + |\sigma|)^2 \cdot |\theta| + \\ &\quad |a| \cdot (|\Psi| + |Q| + |\sigma|) \cdot |\theta|\Big) \end{aligned}$$

where $|O| = |O_1| + |O_2|$ represents the total length of all observations. The overall time complexity for generating N solutions can be summarized as:

$$O\left(N \cdot \sum_{i \in \{\text{obs,derive,strategy,answer}\}} \left(L_i^2 + L_o^i \cdot L_i\right) \cdot |\theta|\right)$$

where L_i represents the input context length and L_o^i represents the output length for each transformation function i. As the context grows through the pipeline, complexity is dominated by later stages with larger contexts. The framework achieves efficiency through prompt engineering and early termination of unpromising reasoning paths.

A.9.20 ***Monte Carlo Tree Search Algorithm.*** : We utilize Monte Carlo Tree Search (MCTS)[14, 17, 40, 50, 58] for improved reasoning-driven response generation in large language models (LLMs), especially for complex, multi-step language tasks where traditional methods often fall short. MCTS offers a framework for language models to engage in structured thinking, logical inference, and multi-step problem-solving, enabling capabilities such as hypothetical and counterfactual reasoning, commonsense and causal reasoning, and multi-source, multi-hop question answering with RAG. By formulating reasoning-driven response generation as a sequential decision-making problem, we demonstrate how MCTS can systematically explore the vast space of potential responses to identify optimal outputs for a given end-user query. This systematic exploration is particularly crucial when dealing with complex queries that require intricate reasoning and planning over multiple steps. Our methodology leverages the inherent uncertainty in language generation and provides a principled way to balance exploration of diverse responses with exploitation of high-quality language patterns. MCTS demonstrates significant improvements in response quality, coherence, and relevance compared to traditional sampling and beam search methods, which are often inadequate for navigating the complexities of multi-step reasoning. We formulate reasoning-driven response generation as a search problem within a state space that evolves with the generation process. Let $s \in \mathcal{S}$ denote a state in the generation process, where $\mathcal{S}$ represents the set of all possible states the generation process can assume. Each state s is formally defined as:

$$s = (p, q, h) \tag{13}$$

Here, $p \in \mathcal{P}$ is the system prompt, which serves to guide and condition the language model's behavior. $\mathcal{P}$ represents the entire set of possible system prompts that can be used. Next, $q \in \mathcal{Q}$ denotes the current user query, which is the latest input to the language model. $\mathcal{Q}$ is the set encompassing all possible queries a user might pose. Finally, $h = ((r_1, c_1), (r_2, c_2), \ldots, (r_n, c_n)) \in \mathcal{H}$ represents the generation history up to the current point. In this history, each element (r_i, c_i) is a message, where $r_i \in \{\text{user, assistant}\}$ specifies the role of the message sender, and $c_i \in C$ is the content of the message. $\mathcal{H}$ is the collection of all possible generation histories. The state space $\mathcal{S}$ grows exponentially with the length of the generation sequence, rendering an exhaustive search for the best response computationally impractical, especially in complex tasks where the sequence of necessary steps can be long and branching. At each state s, the action space $\mathcal{A}(s)$ is defined as the set of all potential responses that the language model can generate from that state:

$$\mathcal{A}(s) = \{a_1, a_2, \ldots, a_k\} \tag{14}$$

Each $a_i \in C$ in this set represents a possible response, which is a content from the language model's output space C. Given a state $s = (p, q, h)$ and an action $a \in \mathcal{A}(s)$, the state transition function $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ determines the next state based on the current state and the chosen action, and is defined as:

$$T(s, a) = (p, q, h \oplus (\text{assistant}, a)) \tag{15}$$

Here, a signifies the action taken, which is the content of the newly generated message by the assistant. The symbol $\oplus$ represents the operation of concatenation, which in this context appends the new assistant message to the existing generation history. Monte Carlo Tree Search (MCTS) iteratively constructs a search tree to discover optimal responses through a sequence of four critical phases, enabling effective planning and decision-making even in complex scenarios: (a) The selection phase is the first step, where the algorithm navigates from the root of the search tree down to a leaf node. This traversal uses the Upper Confidence Bound for Trees (UCT) method, which is essential for balancing the exploration of less-visited branches of the tree against the exploitation of branches that have thus far shown promise. This balance is vital for complex queries where the optimal solution might not be immediately obvious and requires exploration of diverse reasoning paths. The UCT is defined as follows:

$$\text{UCT}(s, a) = \frac{V(s, a)}{N(s, a)} + c \cdot \sqrt{\frac{\ln(N_{\text{parent}}(s))}{N(s, a)}} \tag{16}$$

where $V(s, a)$ represents the cumulative value associated with taking action a from state s, accumulating the evaluations from all simulations that passed through this state-action pair. $N(s, a)$ is the

number of times the action a has been selected from state s, serving as a visit count for this specific state-action pair. $N_{\text{parent}}(s)$ is the total number of visits to the parent node of state s, representing the overall exploration effort from the preceding state. The term c is the exploration weight, a constant that tunes the balance between exploration and exploitation; a higher value encourages more exploration. At each node in the tree during selection, the algorithm calculates the UCT value for each possible action and chooses the action a^* that maximizes this value, guiding the search towards potentially optimal paths.

$$a^* = \arg\max_{a \in \mathcal{A}(s)} \text{UCT}(s, a) \tag{17}$$

(b) Once the selection phase reaches a leaf node s_{leaf}, the expansion phase begins. Here, the tree is expanded by generating k candidate responses from the language model. These responses represent possible actions that can be taken from the leaf state, effectively broadening the search space. For complex tasks, generating diverse candidates is crucial to uncover potentially effective, yet non-obvious, steps towards a solution, supporting hypothetical reasoning by considering multiple potential continuations.

$$\mathcal{A}(s_{\text{leaf}}) = \{a_1, a_2, \ldots, a_k\} \sim f_{\text{LM}}(s_{\text{leaf}}) \tag{18}$$

In this step, f_{LM} denotes the language model generation function, which takes the current state s_{leaf} as input and produces k diverse responses, each representing a potential next step in the response generation. Each candidate response a_i generated in this phase leads to the creation of a new child node in the search tree, with an updated state $s'_i = T(s_{\text{leaf}}, a_i)$ reflecting the addition of the new response to the generation history. (c) Following expansion, the simulation phase, also known as rollout, is initiated from each of the newly created child nodes s'. In this phase, the algorithm simulates future generation steps by proceeding from the child node down to a certain depth or until a terminal state is reached. This lookahead capability is particularly beneficial for complex tasks, allowing the algorithm to assess the longer-term consequences of early decisions and perform multi-step problem-solving by exploring sequences of actions. This simulation is carried out according to the following process:

$$s^{(0)} = s' \tag{19}$$

$$\text{depth} = 0 \tag{20}$$

$$\text{while depth} < d \text{ and not } \tau(s^{(\text{depth})}) : \tag{21}$$

$$A^{(\text{depth})} = \{a_1, a_2, \ldots, a_k\} \sim f_{\text{LM}}(s^{(\text{depth})}) \tag{22}$$

$$a^{(\text{depth})} = \text{Random}(A^{(\text{depth})}) \tag{23}$$

$$s^{(\text{depth+1})} = T(s^{(\text{depth})}, a^{(\text{depth})}) \tag{24}$$

$$\text{depth} = \text{depth} + 1 \tag{25}$$

Here, $s^{(0)} = s'$ sets the starting state for the simulation as the newly created child node. The simulation continues iteratively as long as the current simulation depth is less than a predefined maximum depth d, and the current state $s^{(\text{depth})}$ is not a terminal state, as determined by the terminal state function $\tau(s)$(discussed later). In each step of the simulation, the language model generation function f_{LM} is used to generate a set of possible actions $A^{(\text{depth})}$ from the current state $s^{(\text{depth})}$. Then, an action $a^{(\text{depth})}$ is selected randomly from $A^{(\text{depth})}$ using the Random() function, which chooses uniformly at random from the available actions. The state is then transitioned to the next state $s^{(\text{depth+1})}$ using the state transition function T, and the depth counter is incremented. (d) After the simulation phase completes, reaching either the maximum simulation depth d or a terminal state, the backpropagation phase is executed. In this step, the terminal state $s^{(d)}$ is evaluated using a quality function $Q : \mathcal{S} \to [0, 1]$, which assigns a score reflecting the quality of the simulated generation trajectory. This evaluation step is critical for complex queries, as it allows the algorithm to judge the overall coherence and quality of a multi-step reasoning process, rather than just focusing on immediate next-token probabilities. Furthermore, by evaluating different generation trajectories, MCTS implicitly performs counterfactual reasoning, assessing the impact of different choices made during the generation process. This value is then propagated back up through the search tree, from the node where the rollout began all the way back to the root. The update process is as follows:

$$Q(s) = f_{\text{LM}}^{\text{eval}}(s) \tag{26}$$

$$N(s, a) \leftarrow N(s, a) + 1 \tag{27}$$

$$V(s, a) \leftarrow V(s, a) + Q(s^{(d)}) \tag{28}$$

Here, $f_{\text{LM}}^{\text{eval}}(s)$ is the function that performs the evaluation of a state, providing a quality score. For each state-action pair (s, a) along the path from the rollout start node back to the root, the visit count $N(s, a)$ is incremented by one, and the cumulative value $V(s, a)$ is updated by adding the quality score $Q(s^{(d)})$ obtained from the terminal state of the simulation. Quality evaluation is crucial for MCTS success, and a primary method is using the LLM for self-evaluation. The LLM assesses its own generated responses by being prompted to rate their quality on a scale of 0 to 1. This leverages the LLM's inherent understanding of language, making it effective for nuanced and complex queries, including those requiring commonsense and causal reasoning to judge coherence and relevance. This self-evaluation is represented by $Q(s) = f_{\text{LM}}^{\text{eval}}(M(s) \oplus m_{\text{eval}})$, where the LLM ($f_{\text{LM}}$) evaluates a formatted state ($M(s)$) combined with an evaluation prompt (m_{eval}) to produce a quality score. A terminal state function (τ) is used to manage MCTS computational cost by identifying states for early simulation termination. This is crucial for complex tasks to ensure efficient exploration and prevent unbounded computation, especially in tasks like multi-hop question answering with potentially lengthy reasoning chains. The terminal state function is defined as:

$$\tau(s = (p, q, h_{\text{conv}})) = \begin{cases} 1 & \text{if } |h_{\text{conv}}| > h_{\max} \\ 0 & \text{otherwise} \end{cases}$$

where simulations terminate if the generation history length ($|h_{\text{conv}}|$) exceeds a predefined maximum length ($h_{\max}$). In summary, Monte Carlo Tree Search enhances reasoning-driven response generation in large language models, particularly for complex, multi-step queries. MCTS excels at structured thinking, logical inference, and multi-step problem-solving, enabling capabilities like hypothetical, counterfactual, commonsense, and causal reasoning, as well as multi-hop question answering in RAG settings. By systematically exploring potential responses, MCTS provides a more reasoned and

higher-quality approach to language generation, overcoming limitations of traditional methods through integrated forward planning and evaluation. This multi-step planning and evaluation makes MCTS especially effective for complex tasks demanding intricate reasoning and coherent multi-turn interactions, offering a significant advantage over simpler generation techniques.

A.9.21 ***R** *Algorithm***. : The R*[34] algorithm is a principled approach to improving language model response generation through Monte Carlo Tree Search (MCTS). When presented with a user query, R* systematically explores diverse reasoning pathways to generate high-quality, well-reasoned responses by leveraging specialized reasoning strategies. This framework empowers language models to engage in structured thinking, logical inference, and multi-step problem-solving, enhancing capabilities such as counterfactual and causal reasoning, and multi-step question answering within RAG settings. We formulate response generation as a search process through a tree of reasoning states. In this formulation, let $\mathcal{Q}$ be the set of all possible user queries (input questions), $\mathcal{S}$ be the set of intermediate reasoning states (natural language reasoning steps), $\mathcal{A}$ be the finite set of predefined reasoning actions $\{A_1, A_2, A_3, A_4, A_5\}$ (reasoning strategies), and $\mathcal{N}$ be the set of nodes in the MCTS tree, where each node $n \in \mathcal{N}$ corresponds to a state $s \in \mathcal{S}$. Given a user query $q \in \mathcal{Q}$, R* generates a response by performing multiple rollouts through a dynamically constructed reasoning tree. The process begins with a selection phase where, at each decision point, actions are selected using the Upper Confidence bound for Trees (UCT) to balance exploration and exploitation:

$$a^*(n) = \arg\max_{a \in \mathcal{A}} [\mathrm{UCT}(n, a)]$$

$$\mathrm{UCT}(n, a) = \underbrace{\frac{V(\mathrm{child}(n, a))}{N(\mathrm{child}(n, a))}}_{\text{Exploitation}} + \underbrace{c \cdot \sqrt{\frac{\ln N(n)}{N(\mathrm{child}(n, a))}}}_{\text{Exploration}}$$

where n denotes the current node in the MCTS tree being considered for action selection. Here, $\arg\max_{a \in \mathcal{A}}[f(a)]$ denotes the action a that maximizes the function $f(a)$. In the R* algorithm, an **action** $a \in \mathcal{A}$ represents a predefined reasoning strategy from a finite set $\mathcal{A}$. Each action guides the LLM towards a specific problem-solving approach. For example, action A_1 directs the LLM to identify the immediate next step, while A_2 prompts the development of a comprehensive solution pathway. By strategically selecting and applying these diverse actions during the search, R* orchestrates the LLM's reasoning, encouraging exploration of various tactics to enhance the quality and effectiveness of generated responses. The UCT balances exploitation, represented by $\frac{V(\mathrm{child}(n,a))}{N(\mathrm{child}(n,a))}$, which favors actions that have historically led to higher values, with exploration, represented by $c \cdot \sqrt{\frac{\ln N(n)}{N(\mathrm{child}(n,a))}}$, which encourages the investigation of less-visited actions, controlled by the exploration parameter $c \approx 1.4$. When encountering a node with unexplored actions or during initial rollout, the algorithm expands. For a chosen reasoning action $a \in \mathcal{A}$ applicable to the current state s, a prompt is generated to guide the language model. The language model then generates the subsequent reasoning state s' from this prompt, representing the next step in natural language reasoning, guided by the selected strategy. The LLM functions as a natural language reasoning engine, generating logically progressive states guided by these actions. Following expansion, simulations are performed from the newly expanded nodes to a maximum depth d (typically 5). Specifically, after expanding a node and creating a new child node representing the subsequent reasoning state, the simulation process begins from this child node. It is from this newly created node, which we will now refer to as n for clarity in the following equations, that the simulation initiates:

$$v = \mathrm{Sim}(n)$$

$$\mathrm{Sim}(n) \approx \begin{cases} \mathrm{Eval}(n), & \text{if } \mathrm{depth}(n) \geq d \\ \mathrm{Sim}(\mathrm{RandChild}(n)), & \text{otherwise} \end{cases}$$

In simulation, the process starts from this newly expanded child node n and proceeds by repeatedly selecting random actions (if no children exist, a random action is chosen for expansion from n; if children exist, a random child of n is chosen) until the maximum depth d is reached. At the maximum depth, the evaluate function is called on the final node to estimate its value. This simulation estimates the long-term value of different reasoning approaches without fully exploring all possible paths. After simulation, the estimated value v is propagated backward through the tree in the backpropagation phase:

$$N(n) \leftarrow N(n) + 1$$
$$V(n) \leftarrow V(n) + v$$

This backpropagation updates the visit counts and cumulative values of the current node n and its parent nodes, ensuring that promising reasoning paths receive more exploration in subsequent MCTS iterations. For any reasoning state (represented by a node), we evaluate the quality of the potential response it contains:

$$\mathrm{Eval}(n) = \begin{cases} \mathrm{Conf}(s), & \text{if response in state } s \\ & \text{contains valid answer information} \\ 0, & \text{otherwise} \end{cases}$$

The $\mathrm{Conf}(s)$ function estimates the reliability of the answer extracted from state s, assigning higher confidence to responses that align with expected answer patterns. A critical component of R* is the mutual consistency check, $\mathrm{Consistent}(\tau)$, which validates reasoning trajectories $\tau = (n_0, a_0, n_1, ..., n_k)$:

$$\mathrm{Consistent}(\tau) = \begin{cases} \mathrm{True}, & \text{if } \mathrm{Overlap}(\tau'_{split:k}, \tau_{split:k}) > \theta \\ \mathrm{False}, & \text{otherwise} \end{cases}$$

Here, we split a reasoning trajectory τ into a partial trajectory $\tau_{0:\mathrm{split}}$ and a remaining trajectory $\tau_{\mathrm{split}:k}$. We prompt the LLM with the partial trajectory $\tau_{0:\mathrm{split}}$ and ask it to complete the reasoning, resulting in the predicted continuation $\tau'_{\mathrm{split}:k}$. The $\mathrm{Overlap}(A, B)$ function calculates the normalized word overlap between texts A and B:

$$\mathrm{Overlap}(A, B) = \frac{|\mathrm{Words}(A) \cap \mathrm{Words}(B)|}{|\mathrm{Words}(A) \cup \mathrm{Words}(B)|}$$

where $\mathrm{Words}(X)$ represents the set of normalized words in text X, and θ is a threshold for consistency (e.g., $\theta = 0.7$). The consistency check ensures that reasoning trajectories maintain logical coherence. After performing MCTS and extracting all possible reasoning trajectories, we select the final trajectory τ^* as the optimal trajectory based on a combination of consistency and quality scores:

$$\tau^* = \arg\max_{\tau \in \mathcal{T}} [\text{ValidTraj}(\tau) \cdot \text{Score}(\tau)]$$

where $\mathcal{T}$ is the set of all extracted trajectories, $\text{ValidTraj}(\tau)$ ensures only consistent trajectories are considered, and the $\text{Score}(\tau) = \frac{V(n_{\text{terminal}})}{N(n_{\text{terminal}})}$ evaluates trajectory quality based on the terminal node n_{terminal}. The final response r^* is then derived from the optimal trajectory τ^* using SelectAns:

$$r^* = \text{SelectAns}(\{\text{answer from state } s \mid s \in \tau^*\})$$

$$\text{SelectAns}(\{a_1, a_2, ...\}) = \arg\max_{a_i} [\text{frequency}(a_i) \cdot \text{Conf}(a_i)]$$

This architecture enables R* to address a wide range of language tasks, from factual queries to complex reasoning and creative generation, by systematically exploring and validating diverse reasoning pathways, thus enhancing the quality and reliability of language model responses. The approach is particularly effective for tasks requiring structured reasoning, clarification of ambiguities, and exploration of multiple solution approaches, making R* a versatile framework for improving response generation in various language-based applications.

A.10 Test-Time Inference Techniques Evaluation

Our experiments (see Table 17) demonstrate that all test-time scaling techniques yield improvements over the PORAG+ATLAS baseline. Notably, methods leveraging structured multi-path reasoning—such as Monte Carlo Tree Search and the R* Algorithm—achieve the most substantial gains, improving HotpotQA by up to 23.8% (EM) and 14.5% (F1), and Gorilla accuracy by up to 7.8%. Techniques like Self-Consistency, Best-of-N Sampling, and Chain-of-Thought with Reflection also contribute consistent and meaningful improvements across benchmarks. These findings confirm that dynamic, reasoning-driven inference strategies significantly boost the effectiveness of retrieval-augmented generation across diverse QA tasks.

A.10.1 ***Low-Latency LLM Decoding Strategies.*** : Optimizing inference latency and throughput is critical for RAG systems using LLMs in real-world applications. Inference latency refers to the time taken for a language model to generate a response, while throughput measures the number of tokens or requests processed per unit of time. Lower latency is essential for real-time applications, such as chatbots or virtual assistants, that may leverage RAG systems. Higher throughput is desirable for efficiently handling multiple tasks or serving many users concurrently, as in batch processing or cloud-based services, which can also benefit from RAG architectures. To address latency challenges in RAG systems, various decoding optimization techniques have been developed. Traditional methods like beam search and sampling strategies offer some improvements, but recent algorithmic innovations have shown even greater promise for accelerating inference without sacrificing output quality. (a) FlashAttention-2[10] significantly improves attention computation speed and latency by reengineering the original FlashAttention algorithm[11] to better utilize GPU parallelism and reduce memory inefficiencies, and is effective for low-latency inference and training in long-context Transformer models. Building on its predecessor—which reduced memory I/O via tiling and online softmax—FlashAttention-2 tackles remaining bottlenecks in GPU resource utilization, crucial for scaling Transformers to longer sequences. It introduces three key optimizations: (1) Reducing non-matrix multiplication FLOPs by modifying online softmax to favor GPU-optimized matmul operations and better exploit high-throughput compute units. (2) Increasing thread block occupancy through fine-grained parallelism across the sequence length, in addition to batch and head dimensions, which benefits long sequences and small batch sizes. (3) Improving intra-thread block work partitioning by assigning each warp a slice of the query matrix instead of the key, minimizing shared memory communication. (b) Lookahead Decoding[16] is a parallel decoding algorithm specifically designed to accelerate LLM inference by dramatically reducing sequential decoding steps. Unlike traditional autoregressive methods that generate tokens sequentially, Lookahead Decoding innovatively predicts multiple non-contiguous n-grams concurrently within a "lookahead branch", drawing inspiration from Jacobi iteration techniques. A dedicated "verification branch" then meticulously checks these potential tokens, acting as a quality control mechanism to validate the n-grams as correct continuations that preserve the LLM's intended output distribution, ensuring accuracy and fidelity to the base model's intended output. This method not only surpasses Speculative Decoding[3, 25, 31, 52] by eliminating the need for auxiliary draft models—enhancing efficiency and simplifying implementation—but also incorporates an n-gram pool. This pool caches and reuses promising token sequences, further accelerating performance while maintaining the high quality of generated text. For enhanced efficiency in our ATLAS-augmented RAG framework, we integrate low-latency LLM decoding strategies such as FlashAttention-2 and Lookahead Decoding. FlashAttention-2 directly accelerates the attention computations critical to ATLAS's Multi-Layer Attention Gradient (MLAG) and Layerwise Representation Pooling (LRP) mechanisms, as well as the subsequent token generation within the LLM. Complementarily, Lookahead Decoding reduces the sequential bottleneck of autoregressive generation by enabling parallel token prediction. This synergistic combination promises to significantly reduce the overall latency of our RAG system, resulting in faster dynamic retrieval triggering, quicker query formulation, and accelerated response generation, ultimately leading to a more efficient and responsive user experience for knowledge-intensive tasks. We implement these existing techniques to verify that these latency optimizations do not hinder the performance of our proposed framework.

A.10.2 ***LLM Decoding Efficiency Evaluation.*** : We evaluated the impact of low-latency decoding techniques on the efficiency of our PORAG+ATLAS framework (Qwen2.5-3B). As shown in Table 18, both FlashAttention-2 and Lookahead Decoding offer substantial improvements over the baseline (68.27s latency, 120 tokens/sec). FlashAttention-2, by accelerating attention computations crucial for ATLAS, reduced latency to 29.55s (↓ **56.7%**) and increased throughput to 208 tokens/sec (↑ **73.3%**). Lookahead Decoding achieved further gains through parallel token prediction, decreasing latency to 23.15s (↓ **66.1%**) and boosting throughput to 255 tokens/sec (↑ **112.5%**). These results confirm that incorporating optimized decoding methods significantly enhances the responsiveness of our RAG system by speeding up both retrieval

Table 17: Performance Comparison: PORAG+ATLAS Baseline Enhanced by Test-Time Scaling

Method	HotpotQA (Joint EM / F1)	Gorilla (Overall Acc.)	PubMedQA (Acc / F1)
PORAG+ATLAS (Baseline)	45.29 / 71.32	76.38	78.35 / 74.56
Self-Consistency	48.31 / 74.35 (+6.7%/+4.2%)	77.91 (+2.0%)	80.80 / 77.59 (+3.1%/+4.1%)
Best-of-N Sampling	48.85 / 74.90 (+7.9%/+5.0%)	78.34 (+2.6%)	81.24 / 78.11 (+3.7%/+4.8%)
Chain-of-Thought with Reflection	50.52 / 76.41 (+11.5%/+7.1%)	79.20 (+3.7%)	82.13 / 79.03 (+4.8%/+6.0%)
Entropy-Guided Decoding	49.95 / 75.88 (+10.3%/+6.4%)	78.85 (+3.2%)	81.76 / 78.65 (+4.4%/+5.5%)
CoT Decoding	50.91 / 76.80 (+12.4%/+7.7%)	79.50 (+4.1%)	82.45 / 79.38 (+5.2%/+6.5%)
RE^2	51.87 / 77.75 (+14.5%/+9.0%)	80.01 (+4.8%)	83.05 / 80.01 (+6.0%/+7.3%)
Mixture of Agents	52.55 / 78.47 (+16.0%/+10.0%)	80.41 (+5.3%)	83.50 / 80.55 (+6.6%/+8.0%)
RTO (Reimpl. Then Optimize)	53.10 / 79.02 (+17.3%/+10.8%)	80.78 (+5.8%)	83.89 / 80.98 (+7.1%/+8.6%)
PlanSearch	53.88 / 79.75 (+18.9%/+11.8%)	81.22 (+6.3%)	84.34 / 81.50 (+7.6%/+9.3%)
Monte Carlo Tree Search	54.95 / 80.83 (+21.3%/+13.3%)	81.85 (+7.2%)	85.01 / 82.31 (+8.5%/+10.4%)
R^* Algorithm	**56.05** / **81.68** (+23.8%/+14.5%)	**82.36** (+7.8%)	**85.55** / **82.90** (+9.2%/+11.2%)

and generation phases, complementing the quality enhancements provided by PORAG+ATLAS.

A.11 Related Work

A.11.1 ***Retrieval-Augmented Generation (RAG).*** : Advances in Retrieval-Augmented Generation (RAG) continue to extend the capabilities of Large Language Models (LLMs) in domain adaptation, efficiency, and long-context reasoning. RAFT [60] improves factual accuracy by fine-tuning models to ignore irrelevant retrievals and cite only the most pertinent sources. CoRAG [43] enhances multi-hop reasoning through iterative retrieval, refining queries based on intermediate results rather than relying on a single retrieval step. DRAGIN [38] introduces dynamic retrieval by detecting real-time information needs using model uncertainty and self-attention cues, enabling context-sensitive query formulation during generation. RAPID [4] accelerates long-context inference by combining RAG with speculative decoding, where a draft model predicts outputs for a larger model, balancing speed and accuracy through self- or upward-speculation. MemoRAG [35] integrates external retrieval with a cognitive memory system, recording episodic interactions and distilling them into semantic memory to improve retrieval relevance and consistency. Speculative RAG [46] reduces latency and enhances comprehension by generating draft responses using a small model and verifying them with a larger model. CAG [2] addresses retrieval latency by preloading cached documents into extended context windows, bypassing real-time retrieval altogether. Parametric RAG [39] replaces input-context retrieval with document parameterization, temporarily updating LLM weights during inference to embed external knowledge directly, thereby streamlining the retrieve-update-generate process.

A.11.2 ***Test-Time or Inference-Time Compute.*** : Recent research has significantly advanced the reasoning capabilities of Large Language Models (LLMs) through innovative test-time computation scaling strategies. S1 [32] introduces budget forcing, a prompting strategy that delays early conclusions by inserting "Wait" tokens, encouraging longer and more deliberate reasoning. SETS [5] improves output quality through a cycle of sampling, self-verification, and self-correction, iteratively refining responses until correctness or a termination condition is met. Test-Time Computing (TTC) [22] enables adaptive reasoning by combining a fast initial response with conditionally triggered refinement, emulating a shift from intuitive to deliberative thinking. Knockout and League [6] propose decision-time algorithms that reduce failure rates by comparing or averaging multiple candidate solutions. Marco-o1 [63] combines Chain-of-Thought fine-tuning with Monte Carlo Tree Search (MCTS) to explore diverse reasoning paths for complex problem-solving, while STILL-1 [23] integrates a policy and reward model to guide reasoning through a dynamically expanding tree. The Shortest Majority Vote [57] leverages parallel CoT sampling with CoT-length-aware aggregation to scale inference, and ARMAP [7] learns a reward model directly from environment interactions to guide LLM-based agents in evaluating action trajectories and improving planning. [30] demonstrate that small LLMs can outperform much larger ones by optimizing the test-time scaling of policy models and reward-guided inference. [55] extend this idea through Monte Carlo Tree Diffusion, combining diffusion models with MCTS to support iterative, tree-structured planning. Similarly, [56] propose translating LLM outputs into symbolic PDDL representations to enable classical planning with $A^\star$, leveraging best-of-N sampling and verbalized refinement. [18] present a recurrent depth architecture that scales compute within hidden states to deepen reasoning dynamically. [48] introduce AStar, an MCTS-powered structured reasoning method for multimodal tasks, while [28] propose QLASS, a Q-value-guided stepwise inference framework that enhances reasoning by modeling intermediate decision quality via a reasoning tree. Together, these works highlight a shift toward leveraging structured search, symbolic abstraction, and latent computation for efficient and scalable reasoning.

A.11.3 ***KV Caching.*** : Recent advancements in KV cache management have significantly enhanced the efficiency of Large Language Model (LLM) inference. Efficient inference requires effective management of the Key-Value (KV) cache, which stores intermediate computations during generation. Adaptive and prompt-guided

Table 18: Latency and Throughput Improvements with Low-Latency Decoding Strategies

Method	Avg. Latency (Sec/query)	Throughput (tokens/Sec)
ATLAS+RAG (Baseline)	68.27	120
FlashAttention-2	29.55 (↓ **56.7%**)	208 (↑ **73.3%**)
Lookahead Decoding	23.15 (↓ **66.1%**)	255 (↑ **112.5%**)

strategies include Ada-KV [15], which dynamically distributes compression budgets across attention heads based on their attention patterns, improving memory usage while maintaining generation quality. FINCH [9] proposes a prompt-guided compression strategy that leverages pre-trained self-attention weights to iteratively select the most relevant KV pairs, enabling longer-context processing without requiring fine-tuning. For redundancy reduction, ThinK [51] introduces a query-dependent pruning strategy that identifies and removes less significant channels within the key cache, minimizing memory consumption without compromising model performance. SimLayerKV [61] focuses on inter-layer redundancies by detecting "lazy" layers—those contributing minimally to long-range dependencies—and selectively trimming their KV caches. This approach streamlines memory usage by eliminating unnecessary data storage. Novel mechanisms for long-context inference include DuoAttention [49], which separates attention heads into Retrieval Heads (accessing the full KV cache for global context) and Streaming Heads (operating with a constant-length cache focused on recent tokens). This selective caching reduces memory and latency while preserving the model's ability to handle long contexts. Similarly, SnapKV [27] exploits the observation that attention heads consistently focus on specific prompt features by clustering and retaining only the most relevant KV positions. This strategy improves efficiency while maintaining model performance.

Recent works have proposed efficient strategies for compressing KV caches to support long-context inference in large language models. One approach, L_2-Norm-Based Pruning [13], leverages the observed correlation between the L_2 norm of key embeddings and their attention scores, selectively retaining KV pairs with the lowest norms to reduce memory usage without sacrificing performance. Another line of work, KVQuant [20], applies advanced quantization techniques—including per-channel and pre-RoPE key quantization, non-uniform precision, and sparse-dense vector representations—to compress KV caches to ultra-low bitwidths. These methods enable scalable inference over extended context lengths while maintaining model fidelity. KVLink [53] enhances LLMs by precomputing key-value (KV) caches for individual documents, allowing for efficient reuse during inference and reducing redundant computations. To ensure coherence when combining these precomputed caches, KVLink adjusts positional embeddings to reflect their global positions, introduces trainable special tokens to restore self-attention mechanisms across documents, and employs mixed-data fine-tuning to maintain the model's original capabilities. Together, these advancements collectively optimize memory usage, processing speed, and inference efficiency in LLMs. They highlight a growing emphasis on adaptive, redundancy-aware, and context-sensitive strategies for KV cache management, paving the way for more efficient and scalable LLM inference.