# Extended Abstract

**Motivation** Retrieval-Augmented Generation (RAG) is a popular method to improve factuality and reasoning in language models by injecting external context. While it has been shown to be effective for large-scale models, small models, with their limited capacity and shorter attention spans, often perform worse when naively exposed to long retrieved documents. Motivated by this mismatch, we investigate whether RAG strategies, when paired with effective prompt compression, can help small instruction-tuned models perform better at test time.

Our key research question is whether test-time strategies, specifically retrieval and compression, can be used to boost performance of small language models like Qwen2.5-0.5B-Instruct, a 0.5B parameter open-source. We hypothesize that the compression of both prompts and retrieved content into shorter, more focused inputs may allow small models to benefit from additional information without being overwhelmed by verbosity.

**Method** We design and evaluate five different inference-time strategies:

1. **Prompt-only baseline:** The original user prompt is sent directly to the model.
2. **Prompt + raw RAG:** Retrieved passages from similar prompts are appended to the original prompt.
3. **Prompt + compressed RAG:** Retrieved passages are first summarized using a large language model (Gemma 12B), then appended to the prompt.
4. **Compressed prompt only:** The original prompt is rewritten by Gemma to make it shorter and more focused.
5. **Joint compression:** Both the prompt and the retrieved context are compressed before being passed to the small model.

To implement retrieval, we use the HuggingFaceH4 UltraFeedback dataset, containing 64K GPT-4 ranked prompts and completions. The retriever is a FAISS index built using MiniLM-L6-v2 sentence embeddings over the `train_prefs` split. For summarization, we use Gemma 3 12B to rewrite the prompt, context, or both using fixed prompt templates. Outputs are then scored using NVIDIA's LLaMA-3.1 Nemotron 70B Reward model, which estimates human preference with scalar scores.

**Implementation** The system includes a retriever (FAISS + MiniLM), a compressor (Gemma 12B), and a generator (Qwen2.5-0.5B-Instruct). We use fixed summarization templates and score each strategy on UltraFeedback's test set using the Nemotron reward model.

**Results** Raw RAG degrades performance (46% win rate), while prompt-only achieves 50%. Compressing the prompt improves performance to 50.7%, while compressed RAG and joint compression reach 47.6% and 48.4% respectively. Compression is key; retrieval helps only if condensed.

**Discussion** Our study highlights several insights. First, retrieval alone is insufficient for small models; in fact, it can hurt performance unless followed by compression. Second, prompt compression, even in isolation, appears to be a powerful and lightweight test-time intervention for small LLMs. Third, the combination of retrieval and summarization (joint compression) requires more careful balancing, as it can remove useful signal or preserve noise.

Inspired by hierarchical RL, our architecture decomposes the task of context planning (Gemma) from response generation (Qwen). This division enables us to leverage large models at inference time to improve the performance of smaller models, without training. Such designs are useful in deployment scenarios where large models are expensive to run end-to-end but can assist in preprocessing or optimization.

**Conclusion** Effective use of RAG in small LLMs requires more than retrieval in that it demands compression and planning. Our results show that inference-time prompt shaping via large model compression can improve the performance of RAG for small models without retraining, offering a practical pathway for improving test-time strategies under compute constraints.

# Improving Small Language Models via Test-Time Prompt Compression and Retrieval

**Neha Balamurugan**
Department of Computer Science
Stanford University
nabalmur@stanford.edu

**Keshav Patel Keval**
Department of Management Science and Engineering
Stanford University
keshav5@stanford.edu

**Pranava Singhal**
Department of Computer Science
Stanford University
psinghal@stanford.edu

## Abstract

This work investigates whether retrieval-augmented generation (RAG), when combined with test-time prompt compression, can improve the performance of small instruction-tuned language models. Using Qwen2.5-0.5B-Instruct as our target model, we evaluate multiple inference-time prompting strategies on the UltraFeedback dataset, including prompt-only, raw RAG, compressed RAG, compressed prompt, and joint compression. We use a FAISS-based retriever over MiniLM embeddings and larger models like Gemma 12B as a compression model. Outputs are scored using NVIDIA's Nemotron-70B reward model. Our findings show that naive RAG degrades performance, while prompt compression alone yields the best win rate (50.7%). Joint compression performs better than raw RAG but worse than compressing just the prompt. These results suggest that careful input compression, even without retrieval, is an effective strategy for small models, and that leveraging large models as inference-time planners may offer practical benefits in low-resource deployments.

## 1 Introduction

Large language models (LLMs) have demonstrated remarkable capabilities in a wide range of NLP tasks. However, their performance often comes at the cost of significant computational requirements. In contrast, small language models (SLMs), while more efficient and deployable in resource-constrained settings, lag behind in quality, especially in complex reasoning or fact-intensive tasks. A common approach to mitigate this gap is Retrieval-Augmented Generation (RAG), where retrieved external documents are appended to prompts to enrich the model's context. While RAG has shown clear benefits for large models such as GPT-4 and PaLM, its effectiveness for smaller models remains underexplored.

In this work, we investigate the potential of test-time RAG strategies for improving small instruction-tuned models, specifically Qwen2.5-0.5B-Instruct. Prior studies (e.g., Barnett et al. (2024)) caution that naively appending retrieved documents can overwhelm low-capacity models, leading to performance degradation. To address this, we explore whether compressing both the prompt and retrieved context can improve outcomes. Our hypothesis is that small models may benefit from additional information—but only if that information is distilled and focused.

To evaluate this idea, we design a modular inference pipeline involving (i) retrieval of similar prompt-response examples using FAISS over UltraFeedback, (ii) compression of raw retrieved text using a larger model (Gemma 12B), and (iii) execution by the small model (Qwen2.5-0.5B-Instruct). We compare several prompting strategies, from baseline prompt-only to joint compression of prompt and context. For evaluation, we use the Nemotron 70B reward model as a proxy for human preference.

Our results suggest that retrieval, when applied naively, harms performance in small models. However, simple compression techniques (particularly rewriting the prompt) can improve response quality. These findings have implications for real-world deployments of small models, where inference-time optimizations may serve as practical alternatives to expensive training or finetuning. Our architecture also draws inspiration from hierarchical planning in reinforcement learning: the large model (Gemma) plans the input structure while the small model (Qwen) executes, offering a lightweight and modular design that is both effective and scalable.

## 2 Related Work

Prior work has explored compressing input prompts to reduce inference costs or improve alignment. LLMlingua Jiang et al. (2023) proposes learning to compress user prompts using a small model before passing them into a larger one. Similarly, Style-Compress Pu et al. (2024) examines compressing text into a semantically rich but compact form to preserve style or instruction intent across generations. Unlike these approaches, which typically compress inputs to optimize large model inference, our work studies compression for the inverse setting: using large models to help small models perform better at inference time.

We also relate to recent research in retrieval-augmented generation, which has proven effective for large-scale models like GPT and PaLM. However, small models may suffer from exposure to irrelevant or excessive context. Recent work by Barnett et al. (2024) highlights that unfiltered retrieval often degrades performance in low-capacity settings. Our work contributes to this discussion by showing that summarizing retrieved context—especially in combination with prompt rewriting—offers a lightweight, inference-time alternative to training or finetuning.

Finally, hierarchical prompting frameworks, such as Toolformer Schick et al. (2023) or ReAct Yao et al. (2023), explore chaining model outputs across tasks. Our method is conceptually similar: we use a larger model (Gemma 12B) as a high-level planner that reshapes the task prompt to fit the small model's reasoning budget.

## 3 Method

We design five inference-time strategies that modify or enhance the input prompt before passing it to a small model. Each strategy manipulates the prompt via compression, retrieval, or both. Our goal is to assess whether these test-time augmentations can help a small model like Qwen2.5-0.5B-Instruct produce higher-quality completions, without requiring additional training.

### 3.1 Test-Time Strategies

We define and evaluate the following five strategies:

1. **Prompt-only baseline:** The original user prompt is sent unchanged to Qwen2.5-0.5B-Instruct.
2. **Prompt + raw RAG:** We retrieve similar prompt-response examples from UltraFeedback using a FAISS index built over MiniLM-L6-v2 sentence embeddings. Retrieved examples (prompt, chosen response, rejected response) are directly appended to the original prompt as in-context references.
3. **Prompt + compressed RAG:** The retrieved examples are summarized using a larger teacher model (Gemma 12B). We evaluate two types of compression: (i) behavioral hints, where Gemma identifies what makes the chosen responses helpful and appends guidance lines, and (ii) rewritten prompts, where Gemma modifies the user prompt based on retrieved examples.

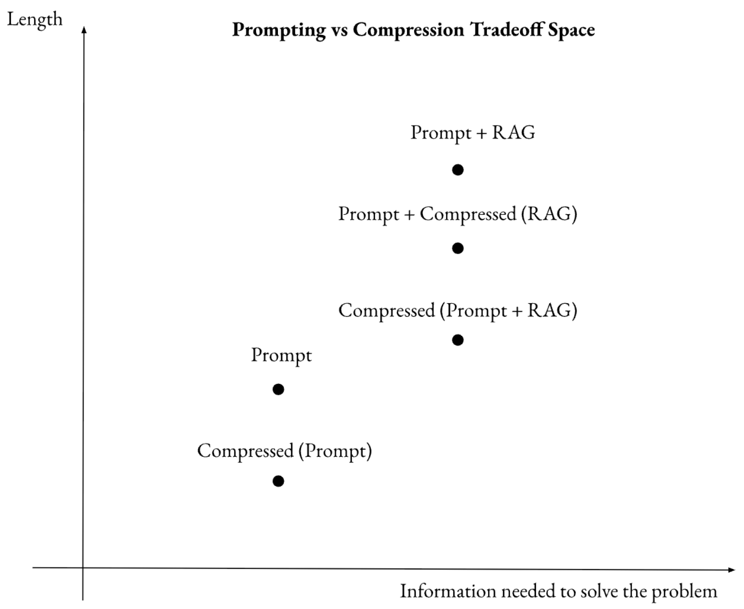
Length
Prompting vs Compression Tradeoff Space
Prompt + RAG
Prompt + Compressed (RAG)
Compressed (Prompt + RAG)
Prompt
Compressed (Prompt)
Information needed to solve the problem

Figure 1: Intuitive visualization of the tradeoff in instruction length and the amount of "useful information" in the prompt needed to give a good response. We define useful information as that which would be preserved under an ideal compressor such as a powerful language model which can remove any information in the prompt that does not help in writing a better response. The goal is to maximize information while reducing length so that smaller models are able to more effectively use their limited context.

4. **Compressed prompt only:** The original prompt is rewritten by Gemma 12B using a fixed template to improve clarity and brevity without removing task-critical information. No retrieval is performed.
5. **Joint compression:** Both the prompt and the retrieved examples are compressed. Retrieved examples are distilled into behavioral hints by Gemma and appended to a rewritten version of the prompt.

## 3.2 Retrieval-Augmented Generation (RAG)

For RAG-based strategies, we construct a vector index using the `train_prefs` split of HuggingFace's UltraFeedback dataset. Each entry in the index includes a prompt and its associated chosen and rejected completions. We use the `all-MiniLM-L6-v2` encoder to produce normalized embeddings and build a FAISS `IndexFlatIP` for fast top-$k$ retrieval.

At inference time, we retrieve $k = 3$ nearest neighbors based on the current prompt. Retrieved entries are formatted into structured references with the following template:

```
Example 1:
Retrieved prompt: ...
Chosen response (good): ...
Rejected response (bad): ...
```

These references are either appended directly (in raw RAG) or passed through the compression module before being combined with the prompt.

## 3.3 Prompt Compression

Compression is performed using Gemma 12B with nearly deterministic decoding (`temperature=0.01`). We employ two templates for different purposes:

**Prompt rewriting:** Gemma rewrites the prompt to be clearer and more concise while preserving all task-relevant details. The prompt is framed as an instruction to a prompt editor that emphasizes retention of meaning over brevity.

**Behavioral hint extraction:** Given retrieved examples, Gemma outputs 1–2 guiding sentences that encourage the target model to mimic the desirable qualities of chosen responses. This encourages alignment with high-quality behavior without copying content.

### 3.4 Execution and Scoring

Final prompts—after applying compression, retrieval, or both—are sent to Qwen2.5-0.5B-Instruct hosted via vLLM. For evaluation, we generate completions for each strategy and compare them pairwise against the baseline using NVIDIA's Nemotron 70B Reward Model. This reward model takes the original user prompt and candidate response as input and outputs a scalar preference score.

We compute win-rate by comparing the strategy response to the baseline for each prompt in the `test_prefs` split. A strategy receives a full point if its score is higher, half a point if equal, and zero if lower.

### 3.5 System Architecture

Our pipeline includes three components:

- **Retriever:** FAISS index over UltraFeedback prompts with MiniLM embeddings.
- **Compressor:** Gemma 12B used for rewriting prompts and summarizing retrieved examples.
- **Generator:** Qwen2.5-0.5B-Instruct model queried via vLLM.

Evaluation is parallelized across 32 threads to accelerate response generation and reward scoring.

## 4 Experimental Setup

We evaluate each test-time strategy on the `test_prefs` split of the HuggingFaceH4 UltraFeedback dataset. This dataset contains human-aligned preferences over model completions for diverse user prompts. We use 64K prompts from the `train_prefs` split to build the retrieval index and evaluate on the held-out `test_prefs` set.

### 4.1 Scoring with Nemotron

To assess output quality, we use NVIDIA's `llama-3.1-nemotron-70b-reward` model, which produces a scalar reward score reflecting human preference. For each prompt, we compare the baseline prompt-only response to the response from a candidate strategy. The two responses are independently scored using Nemotron, each in the context of the original user prompt.

Let $s_A$ and $s_B$ denote the Nemotron scores for the baseline and strategy outputs, respectively. We compute win-rate as follows:

- 1 point if $s_B > s_A$
- 0.5 points if $s_B = s_A$
- 0 points if $s_B < s_A$

The final win-rate is the average over all test prompts.

### 4.2 Retrieval Configuration

We encode prompts from the training set using the `all-MiniLM-L6-v2` model from SentenceTransformers. A FAISS `IndexFlatIP` is used to enable efficient top-$k$ similarity search. For each test prompt, we retrieve $k = 3$ neighbors and include both the chosen and rejected responses from each example.

### 4.3 Compression Setup

For strategies involving compression, we query Gemma 12B (hosted via vLLM) with fixed prompt templates. Prompt rewriting uses a template that instructs Gemma to produce a shorter but equally informative version of the original prompt. Behavioral hint extraction uses a different prompt that summarizes what makes the chosen completions better than the rejected ones.

All generations from Gemma use nearly deterministic decoding (`temperature=0.01`) to ensure consistency across reruns.

### 4.4 Generation and Infrastructure

Final prompts are passed to Qwen2.5-0.5B-Instruct, served via a local vLLM server. Generation uses `temperature=0.01`, `top_p=0.95`, and a maximum output length of 1024 tokens. We execute all generations using a multi-threaded pipeline with 32 concurrent threads to parallelize prompt construction, inference, and scoring.

### 4.5 Implementation Details

All experiments are run on eight A100 GPU machines. Code is modular, with each strategy implemented as a closure over a message formatting function. Strategies can be swapped without changing core evaluation logic.

## 5 Results

We evaluate each strategy by computing its win-rate against the prompt-only baseline using the Nemotron-70B Reward Model. A higher win-rate indicates that a given strategy produces outputs preferred more often than the baseline. We also conduct a quantitative and qualitative analysis to understand the factors contributing to performance gains or degradations.

### 5.1 Quantitative Evaluation

Table 1 shows the win-rates of all five strategies compared to the prompt-only baseline on the `test_prefs` split. Each strategy is evaluated across the same set of prompts, with reward scores computed independently using Nemotron.

Table 1: Win-rates against the prompt-only baseline (Qwen2.5-0.5B-Instruct). A win-rate above 50% indicates the strategy outperformed the baseline.

| Strategy | Win-Rate (%) | Description |
|---|---|---|
| Prompt-only baseline | 50.0 | Direct prompt, no augmentation |
| Prompt + raw RAG | 46.0 | Retrieved examples appended verbatim |
| Compressed prompt only | **50.7** | Rewritten prompt using Gemma |
| Prompt + compressed RAG | 47.6 | Retrieved examples summarized by Gemma |
| Joint compression | 48.4 | Prompt and retrieved context compressed |

Our results highlight several key takeaways:

- **Naive retrieval hurts:** The raw RAG strategy underperforms the baseline (46.0%), suggesting that appending full retrieved examples without filtering or summarization overwhelms the small model.
- **Compression helps:** Rewriting the prompt alone yields the best performance (50.7%), indicating that focused and concise prompts allow small models to better align with user intent.
- **Joint compression offers a moderate boost:** While not outperforming prompt-only compression, combining behavioral hints with a rewritten prompt (48.4%) performs better than raw RAG and shows that retrieved content is helpful if appropriately summarized.

### 5.2 Qualitative Analysis

To better understand these trends, we examine a random subset of test cases where strategy performance diverged. We observe the following patterns:

**Failure cases for raw RAG:** In many instances, appending long or tangential retrieved examples distracted the model, leading to generic or off-topic completions. The small model often copied irrelevant patterns from the retrieved completions.

**Success cases for compression:** Rewritten prompts tended to remove ambiguity, reduce verbosity, and surface the core task. This clarity helped the model provide more structured and relevant responses, even in the absence of additional context.

**Joint compression tradeoffs:** While combining compression and retrieval helped on some examples (especially where the retrieved context provided relevant patterns), overly aggressive summarization occasionally removed useful task-specific detail, leading to neutral or vague responses.

### 5.3 Error Analysis

We identify three main sources of degradation in performance:

- **Context overload:** Small models are sensitive to long prompts. RAG often exceeds the effective context window, leading to truncation of the user query.
- **Compression loss:** While compression reduces input length, it can also obscure task constraints or introduce ambiguity if the summarization prompt is too generic.
- **Prompt-template mismatch:** Certain prompt types (e.g., creative writing or multi-step reasoning) are more sensitive to prompt rewording and lose fidelity when rewritten.

### 5.4 Compression vs. Retrieval

Our results support the hypothesis that compression is the dominant contributor to performance gains. When retrieval is used without compression, performance drops. However, compressing just the prompt (without retrieval) already improves win-rate, suggesting that prompt clarity is more important than external context in many cases.

We conclude that retrieval is only beneficial when it provides examples that are relevant, concise, and easy to abstract into behavioral guidance for small models.

## 6 Discussion

Our study was motivated by a key question: *Can small language models be improved at test time through better prompting—without retraining—by leveraging the capabilities of larger models?* In large LLMs, retrieval has been instrumental in improving performance by supplementing context. However, for small models, naive retrieval often fails due to limited capacity and attention span. Our results confirm this: RAG degrades performance, while prompt rewriting alone leads to the largest gains.

### 6.1 Why Does Prompt Compression Work?

Prompt compression improves outcomes by reducing noise and focusing the model's attention on the core task. Small models, due to their restricted context windows and lower capacity, are especially sensitive to verbosity, ambiguity, and irrelevance. Rewriting the prompt helps in the following ways:

- It makes the task more explicit by removing unnecessary linguistic complexity.
- It reduces cognitive load on the model by shortening the prompt.
- It aligns the structure of the input with the style of instruction-following models.

This aligns with observations in prior work where minimal and well-structured prompts outperform verbose ones for small models.

### 6.2 Why Does Raw RAG Hurt?

Unlike large models that can effectively integrate and reason over long retrieved passages, small models lack the capacity to filter, rank, or ignore irrelevant context. The RA strategy introduces additional sequences—some loosely related—that may dilute or even conflict with the original user intent. Without any filtering or summarization, these examples often distract more than they help. This highlights a critical limitation in reusing large-model techniques in low-capacity settings without adaptation.

### 6.3 Rethinking the Role of Retrieval

Our findings suggest that retrieval remains useful for small models—but only when used judiciously. Compressing retrieved examples into high-level behavioral hints avoids overwhelming the model while still transmitting helpful patterns. Retrieval should be seen less as a way to inject content verbatim and more as a signal to shape how the model behaves.

An alternative framing is that retrieval serves as a form of weak supervision or regularization at test time. By conditioning on past examples—but only through abstracted or compressed forms—we can guide small models toward higher-quality completions without increasing the effective input length too much.

### 6.4 Hierarchical Planning as a Design Principle

Our approach implicitly follows a hierarchical planning paradigm. The large model (Gemma 12B) acts as a high-level planner: it chooses what parts of the input matter and how to phrase them. The small model (Qwen2.5-0.5B-Instruct) acts as the executor, generating a response based on the distilled input. This separation of roles reflects patterns in cognitive science and RL, where planners and actors operate at different levels of abstraction.

This design has practical implications:

- It enables deployment-time improvements for small models without retraining.
- It allows flexible division of compute: the large model can be used once or periodically, while the small model handles all real-time interaction.
- It sets the stage for test-time adaptation pipelines, where the compressor can be optimized (or learned) independently of the generator.

### 6.5 Challenges Faced

During implementation, we encountered several practical challenges. We observed that minor changes in summarization templates significantly influenced output quality, which made iterative debugging more involved. Moreover, it was difficult to choose the best template for RAG examples. An automated optimization pipeline using a framework such as DSPy could have improved performance further.

## 7 Conclusion

We discuss limitations and broader impact of our work as well as future directions of research.

### 7.1 Limitations and Future Work

While our results demonstrate the promise of compression, several limitations remain:

- **Compression quality depends on template design.** Poorly constructed summarization prompts can remove important task-specific details.
- **Retrieval relevance is not guaranteed.** FAISS-based retrieval depends on embedding similarity, which does not always correlate with functional similarity.
- **Evaluation via reward models has bias.** The Nemotron reward model is a strong proxy but may not reflect all dimensions of human preference (e.g., creativity, politeness).

Future work could address these by:

- Learning task-specific compression modules via distillation or imitation learning.
- Incorporating more sophisticated retrievers that optimize for downstream utility, not just similarity.
- Evaluating with human annotators or multi-dimensional feedback models.

### 7.2 Broader Impacts

As small models become increasingly attractive for edge deployment, our findings open a new direction: using large models as test-time optimizers rather than end-to-end generators. This hybrid approach balances quality and efficiency, and could benefit applications in mobile assistants, on-device summarizers, or chatbots running under resource constraints. It also encourages a shift in perspective—from training bigger models to using existing ones more intelligently.

## 8 Team Contributions

- **Keshav Patel Keval:** Implemented and tested RAG pipeline. Initial SFT implementation
- **Pranava Singhal:** Setup efficient inference and prompt compression. Initial DPO implementation
- **Neha Balamurugan:** Evaluation setup, poster write-up, integrating RAG with inference pipeline

**Changes from Proposal** We initially proposed a custom project focused on system-level benchmarking and optimization of reinforcement learning training. However, due to the complexities involved in setting up and conducting a systems analysis within Isaac Gym, we pivoted to the default project. Maintaining our interest in test-time methods, we proposed implementing Retrieval-Augmented Generation (RAG) and evaluating its performance with the Qwen model. Early experiments, however, highlighted the limitations of applying test-time techniques such as RAG and self-reflection to smaller models. This prompted us to explore a related direction: prompt and RAG compression, with the goal of enabling effective test-time improvements for lightweight models like Qwen.

## References

Scott Barnett, Stefanus Kurniawan, Srikanth Thudumu, Zach Brannelly, and Mohamed Abdelrazek. 2024. Seven Failure Points When Engineering a Retrieval Augmented Generation System. arXiv:2401.05856 [cs.SE] `https://arxiv.org/abs/2401.05856`

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models. arXiv:2310.05736 [cs.CL] `https://arxiv.org/abs/2310.05736`

Xiao Pu, Tianxing He, and Xiaojun Wan. 2024. Style-Compress: An LLM-Based Prompt Compression Framework Considering Task-Specific Styles. arXiv:2410.14042 [cs.CL] `https://arxiv.org/abs/2410.14042`

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. arXiv:2302.04761 [cs.CL] `https://arxiv.org/abs/2302.04761`

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629 [cs.CL] `https://arxiv.org/abs/2210.03629`