Dual Compression for Agentic AI

Aaron Wu
Rice University
aw80@rice.edu

Weizhe Mao
Rice University
bm68@rice.edu

Lucas Li
Rice University
ll103@rice.edu

Abstract

Agentic AI systems have become increasingly prevalent while showing significant progress in the field of autonomous task execution, but in many cases, struggle due to the computational inefficiencies in both model architecture and prompt engineering. This report presents both an ind-depth analysis of model and prompt compression for agentic systems, but also introduces a novel dual compression approach that jointly optimizes both model parameters and prompt structures in large language models to enable more efficient agentic systems. We introduce a structured search space specification that transforms unstructured prompts into well-defined combinatorial optimization problems, and implement a successive halving optimizer to efficiently navigate this space. Our method dynamically balances compression ratios across both dimensions based on task requirements. Tests that were completed over several large datasets demonstrated that the approach presented can recover performance within 1-3% of uncompressed models while reducing overall memory footprint and inference costs. The results highlight the potential for dual compression to make agentic AI systems more deployable in resource-constrained environments without sacrificing their autonomous capabilities.

1 Introduction

Agentic AI systems have become invaluable in the field of improving large language models because they have shown the best promise in terms of autonomously setting goals, adapting to dynamic environments, and executing complex tasks through reasoning and tool use [1]. Throughout this developmental process, systems like AutoGPT and BabyAGI have built on this approach, combining LLMs with planning mechanisms, memory, and tool interfaces to create agents capable of independent task execution. Though the capabilities they demonstrate are significant in terms of future work, their biggest problems come in terms of massive computational overhead, making them resource-intensive and challenging to deploy in many practical settings.

Two primary factors contribute to this inefficiency: the overall size of modern LLMs that contain billions of parameters that demand significant memory and computational resources, and the complex prompts required for agentic behaviors. At the same time, to reach the full potential of the agentic capabilities, they typically require extensive prompting with detailed instructions, which results in more token counts and inference costs.

Through a literature review of traditional papers, it is apparent that these challenges have been addressed separately, focusing either on model compression techniques or prompt optimization methods. However, these isolated approaches fail at capturing the relationship between model architecture and prompt design in the agentic context. This paper introduces a novel dual compression framework that jointly optimizes both dimensions, yielding more efficient systems without sacrificing capability.

Our contributions are therefore threefold:

Preprint. Under review.

- We formulate a mathematical framework for dual compression that optimizes both model architecture and prompt structures simultaneously, accounting for their interdependence.
- We introduce a successive halving optimization approach that efficiently explores the combinatorial search space of model-prompt configurations.
- We evaluate our approach across multiple datasets and task types, demonstrating significant efficiency gains while maintaining performance within 1-3% of uncompressed baselines.

From our limited implementation, our results demonstrate that this integrated approach has some potential viability. Additionally, our framework provides a principled way to trade off between model size and prompt complexity based on task requirements, which can enable flexibility in terms of future deployment oportunities for agentic AI systems.

2 Background

2.1 Agentic AI

Agentic AI refers to systems that can independently sense their environments, decide things, employ instruments, and execute plans with negligible human intervention [4]. Some of the most critical capabilities of agentic systems include:

- **Perception:** The ability to perceive and understand input data from various sources.
- **Decision-making:** Capacity to select appropriate action in light of goals and circumstances.
- **Tool use:** The ability to utilize external APIs and tools to accomplish tasks.
- **Self-directed action:** The capability to execute advanced action sequences.

AutoGPT and BabyAGI are newer implementations that illustrate the promise of integrating LLMs with planning frameworks and memory infrastructure to build proactive goal-setting, dynamic environment-reasoning agents, and advanced task-executing agents. Nevertheless, such systems largely rely on large models and complex prompting techniques and are computationally expensive and challenging to deploy in resource-constrained environments.

2.2 Model Compression

Model compression techniques aim to reduce LLMs' computational and memory requirements at the cost of not their capabilities [2]. The favored techniques are:

Quantization: Reducing the precision of model weights, typically from 32-bit floating-point to lower bit-width representations, such as 8-bit, 4-bit, or even 2-bit. It reduces memory requirements considerably but could result in the loss of accuracy, particularly at the aggressive compression ratios.

Pruning: Elimination of unwanted or less important weights from the model, creating sparse networks that are less computationally and storage-intensive. Various pruning methods exist, from simple magnitude-based approaches to more intricate ones taking into account parameter importance.

These techniques have been shown to be promising for reducing model size and speeding up inference. For instance, Figure 1 illustrates how different quantization and pruning levels affect model perplexity for LLaMA-7B on the C4 dataset. Compression, however, necessarily comes at some cost to model capabilities, particularly on complex reasoning tasks prevalent in agentic AI applications.

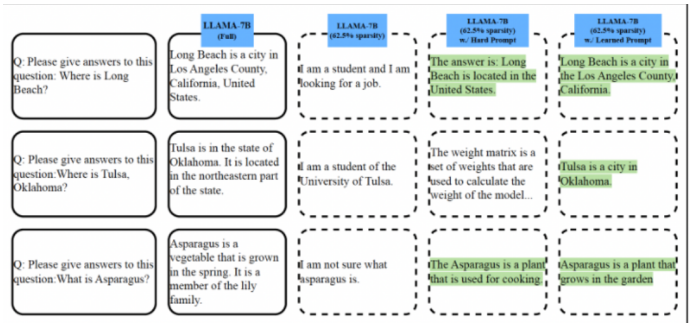

Figure 1: Impact of quantization and pruning on LLaMA-7B perplexity. Left: Perplexity increases as bit width decreases. Right: Perplexity increases with higher sparsity ratios. Green line indicates original model perplexity. (Conceptual illustration based on "Compress, Then Prompt" [2])

2.3 Prompt Compression

Prompt compression techniques attempt to abbreviate prompts without losing usefulness [3]. Several approaches have been devised:

Static Compression: Removal of redundant information manually or automatically from prompts without sacrificing vital instructions and examples. This can work but can require domain knowledge.

Dynamic Compression: Dynamically adjusting prompts in real-time based on context and task needs. Techniques such as LLM-DCP model prompt compression as a Markov Decision Process so that agents can progressively remove redundant tokens and yet preserve important information [4].

Learned Compression: Learning specialized models to compress prompts and preserve semantic intent. Techniques such as LLMLingua demonstrate that prompt length can be compressed up to 20x without substantial performance degradation [3].

These approaches provide valuable instruments to reduce the token count in prompts, which maps directly to lower inference costs. They operate mostly independently of model compression techniques but lose possible synergistic optimization opportunities.

3 Dual Compression Framework

We introduce a dual compression framework that jointly optimizes both model architecture and prompt structures for agentic AI systems. Figure 2 illustrates our proposed architecture.

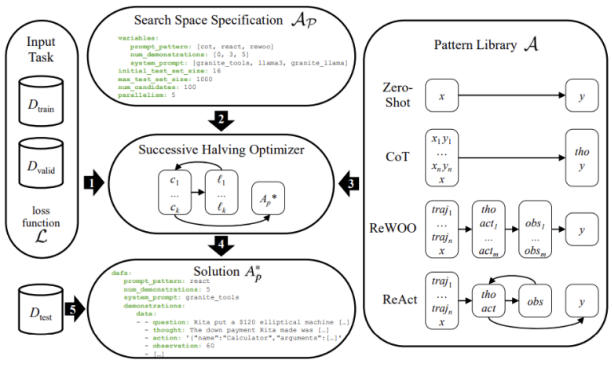

Figure 2: Agentic Solution Architecture for Dual Compression, showing the primary components: Search Space Specification, Pattern Library, Input Task processing, Successive Halving Optimizer, and the resulting optimized Solution.

3.1 Problem Formulation

We formulate the dual compression problem as finding the optimal balance between model size and prompt complexity that maximizes task performance while meeting computational constraints:

$$\begin{aligned} \max_{M_c, P} \quad & \text{Performance}(M_c, P) \\ \text{s.t.} \quad & \text{Size}(M_c) \leq S_{\max} \\ & \text{Length}(P) \leq L_{\max} \end{aligned} \tag{1}$$

where M_c represents a compressed model, P is a prompt, $S_{\max}$ is the maximum allowable model size, and $L_{\max}$ is the maximum prompt length.

For prompt learning specifically, we build on approaches from previous work [2] by optimizing learnable soft prompt tokens $e_1, \ldots, e_k$ to prepend to input sequences $x_0, \ldots, x_n$. The objective is to minimize:

$$\min_E \mathcal{L}_\theta = \min_E \sum_{t=1}^{T} -\log \Pr_{\hat{\theta}}[x_t | e_1, \cdots, e_k, x_0, \cdots, x_{t-1}] \tag{2}$$

where $E \in \mathbb{R}^{k \times d}$ is the prompt embedding matrix (trainable), $\hat{\theta}$ represents the frozen, compressed LLM parameters, and only the prompt embeddings E are updated during training.

3.2 Successive Halving Optimizer

To efficiently navigate the vast search space of possible model-prompt configurations, we implement a successive halving optimization approach. This algorithm adjusts computational resource allocation by progressively eliminating poor-performing configurations:

Algorithm 1 Successive Halving for Dual Compression

Input: Candidate set of model-prompt pairs $\{(M_i, P_i)\}_{i=1}^N$, resource budget B
Output: Optimized model-prompt pair (M^*, P^*)
$S \leftarrow \{(M_i, P_i)\}_{i=1}^N$ ▷ Initial candidate set
$r \leftarrow B/|S|$ ▷ Per-configuration budget
while $|S| > 1$ **do**
    for $(M_i, P_i) \in S$ **do**
        Evaluate (M_i, P_i) with budget r
    end for
    $S \leftarrow$ top $\lfloor |S|/2 \rfloor$ configurations
    $r \leftarrow 2r$ ▷ Double per-configuration budget
end while
return remaining configuration in S

This approach efficiently allocates computational resources by saving evaluation for promising configurations. By doubling the remaining candidate's evaluation budget in each round, it ensures more extensive evaluation for most promising configurations.

3.3 Pattern Library

Our system includes a pattern library that provides composable prompt patterns as a domain-specific language for structured reasoning. The library includes several significant patterns:

- **Zero-Shot:** Instruction without examples
- **Chain-of-Thought (CoT):** Breaking down step-by-step reasoning prompts
- **ReAct:** Interleaving reasoning and action
- **ReWOO:** Reasoning with observation and optimization

Patterns can be interwoven and parameterized to create complex prompting strategies for specific tasks. The pattern library enables a structured approach to prompt optimization, transforming unstructured prompts into well-defined combinatorial search problems.

Dataset	Model	Accuracy			Pattern	Runtime (HH:mm)
		Zero-Shot	Optimized	Delta		
FEVER	Granite 3.1 8B	78.3%	79.0%	+0.7pp	ReWOO (5 shot)	08:55
	Granite 13B Instruct V2	6.5%	75.4%	+68.9pp	ReWOO (3 shot)	08:12
	Granite 20B Code	39.7%	64.2%	+24.5pp	CoT (3 shot)	05:06
	Granite 34B Code	56.4%	65.6%	+9.2pp	CoT (3 shot)	03:47
	LLaMA 3.1 8B	68.5%	78.0%	+9.5pp	CoT (3 shot)	05:24
	LLaMA 3.1 70B	29.7%	86.3%	+56.6pp	CoT (3 shot)	04:57
GSM8K	Granite 3.1 8B	74.5%	75.8%	+1.3pp	ReAct (5 shot, Granite Tools)	01:29
	Granite 13B Instruct V2	23.2%	30.3%	+7.1pp	CoT (5 shot)	02:24
	Granite 20B Code	68.8%	68.8%	+0.0pp	Zero-Shot (Baseline)	05:06
	Granite 34B Code	72.3%	72.3%	+0.0pp	Zero-Shot (Baseline)	03:19
	LLaMA 3.1 8B	78.4%	84.8%	+6.4pp	CoT (3 shot)	03:24
	LLaMA 3.1 70B	82.1%	94.8%	+12.7pp	CoT (5 shot)	04:09
MBPP+	Granite 3.1 8B	68.8%	68.8%	+0.0pp	Zero-Shot (Baseline)	02:07
	Granite 13B Instruct V2	10.7%	18.8%	+8.0pp	ReAct (3 shot)	02:55
	Granite 20B Code	57.6%	60.7%	+3.1pp	ReAct (5 shot)	02:57
	Granite 34B Code	58.9%	59.8%	+0.9pp	ReAct (3 shot)	04:52
	LLaMA 3.1 8B	61.2%	67.4%	+6.2pp	ReAct (5 shot)	01:25
	LLaMA 3.1 70B	73.2%	73.2%	+0.0pp	Zero-Shot (Baseline)	01:38

Figure 3: Pattern Library showing different prompting techniques: Zero-Shot, Chain-of-Thought (CoT), ReWOO, and ReAct patterns with their respective graph structures.

3.4 Dynamic Prompt Compression Agent

To realize adaptive, context-aware prompt trimming we integrate the *Dynamic Compressing Prompts* (DCP) agent recently introduced by Jinwu Hu [4]. In contrast to one-shot token classifiers, DCP formulates compression as a sequential decision problem: the agent inspects the current (possibly already shortened) prompt, decides which tokens to keep, applies that decision, and repeats until a stopping criterion is reached. This design captures the strong context dependence of token importance—removing one word can change the salience of the next—and naturally yields a Markov Decision Process (MDP).

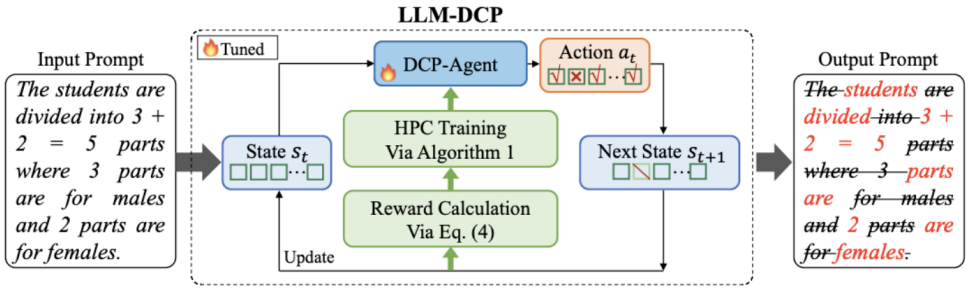

Figure 4: Architecture and interaction loop of the Dynamic Prompt Compression (DCP) agent. Solid arrows denote the forward inference and token-masking path; dashed arrows depict the reward signal and policy-gradient updates.

MDP definition. Let the original prompt be $x = \{x_i\}_{i=1}^{L}$. After t pruning steps the residual prompt is denoted $s_t = \tilde{x}_t$ and serves as the *state*. The *action* a_t is a binary mask of length $|s_t|$ that assigns each surviving token either "keep" (1) or "drop" (0). Applying that mask obtains the next state

$$s_{t+1} = M_{a_t}(s_t), \tag{3}$$

where the operator M_{a_t} deletes all tokens whose mask entry equals zero. An episode terminates after at most $T_{\max}$ pruning iterations or when no further token may be removed without violating the compression curriculum discussed below.

Reward shaping. A composite reward guides the agent at every step:

$$\begin{aligned}\mathcal{R}(s_t, a_t) = \alpha\, \rho^{-1} + \beta\, D\big(s_0, s_t\big) - \gamma\, \mathrm{KL}\big(P(y \,|\, s_t), P(y \,|\, s_0)\big) \\ - \mathbb{1}_{\{\rho < c_s\}}\, P_s - \mathbb{1}_{\{\rho > c_l\}}\, P_l\end{aligned} \tag{4}$$

Here $\rho = |s_t|/L$ is the current compression rate, so $\frac{1}{\rho}$ directly rewards length reduction. The term $D(\cdot, \cdot)$, instantiated as BERTScore [13], incentivizes semantic preservation, while the Kullback–Leibler divergence penalizes deviations in the output distribution of the (aligned) language model when queried with the compressed prompt. Two indicator penalties, controlled by lower and upper bounds (c_s, c_l), prevent over- or under-compression.

Policy and value networks. Both actor and critic share an XLM-RoBERTa-Large encoder that yields contextual embeddings $h_i = f_\theta(s_t)_i$ for every token in s_t. A single linear layer converts h_i into keep/drop logits, producing the policy

$$\pi_\theta(a_t \mid s_t) = \prod_{i=1}^{|s_t|} \mathrm{softmax}(W h_i + b)_{\,a_{t,i}}.$$

The critic $V_\phi(s_t)$ attaches a two-layer multilayer perceptron on top of the same encoder and is trained to regress the expected return. Sharing the encoder reduces memory footprint and speeds convergence.

Optimization with PPO. Parameters θ are updated by Proximal Policy Optimization [14]. For a trajectory τ collected under the frozen policy $\pi_{\theta_{\text{old}}}$ the clipped surrogate objective

$$\begin{aligned}J(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta_{\text{old}}}}\Big[\\ \min\big(\delta_t\, A_{\theta_{\text{old}}}(s_t, a_t), \\ \mathrm{clip}(\delta_t, 1-\varepsilon, 1+\varepsilon)\, A_{\theta_{\text{old}}}(s_t, a_t)\big)\Big].\end{aligned} \tag{5}$$

is maximized, where $\delta_t = \pi_\theta(a_t \mid s_t)/\pi_{\theta_{\text{old}}}(a_t \mid s_t)$ and $A_{\pi_{\theta_{\text{old}}}}$ is the advantage computed from the critic. The clipping parameter ε is fixed at 0.15, ensuring each policy update stays within a trust region and training remains stable despite the high-variance reward.

Hierarchical Prompt Compression curriculum. To ease optimization the admissible compression band $[c_s, c_l]$ is tightened gradually according to

$$c_s = 0.6 - \Big(P_i + \tfrac{t}{T_{\max}}\Big)\psi, \qquad\qquad c_l = 1.0 - \Big(P_i + \tfrac{t}{T_{\max}}\Big)\psi. \tag{6}$$

where $P_i \in \{1, 2, 3\}$ is the current stage index, $\psi = 0.1$, and $T_{\max} = 2$ for the first two stages and 1 for the final stage. The curriculum therefore allows the agent to master mild compression before attempting the aggressive regimes employed at deployment.

3.5 Implementation Details

Building on our theoretical framework, we have implemented a complete dual compression system with several key components. The Dynamic Prompt Compression Agent uses an XLM-RoBERTa encoder with policy and value heads for reinforcement learning. This architecture enables the agent to iteratively decide which tokens to keep or remove from prompts while preserving semantic meaning, as described in Section 3.4.

For compression of models, quantization (8-bit, 4-bit, and 2-bit) and different pruning strategies (magnitude-based, random, and structured) are supported. These techniques are employed as module transformations that are applicable to pre-trained models with minimal performance loss.

Successive Halving Optimizer optimizes over the candidate configuration combinatorial space efficiently. Each configuration represents a specific combination of model architecture settings (model size, quantization level, sparsity ratio) and prompt structures (pattern type, few-shot examples, compression ratio). The optimizer evaluates configurations on task subsets and progressively eliminates lower-performing options.

Listing 1 shows our core configuration evaluation process:

Listing 1: Core configuration evaluation function

```
def evaluate_configuration(config, dataset):
    model, tokenizer = load_model_and_tokenizer(
        config.model_name,
        quantization_bits=config.quantization_bits
    )

    if config.sparsity_ratio > 0:
        model = prune_model(model, sparsity_ratio=config.sparsity_ratio)

    pattern = pattern_library.get_pattern(config.pattern_name)
    examples_text = pattern_library.generate_few_shot_examples(
        config.pattern_name,
        dataset.get_examples(config.n_shots),
        config.n_shots
    ) if config.n_shots > 0 else ""

    prompt = pattern.format(
        instruction=pattern.parameters["instruction"],
        examples=examples_text,
        input=input_text
    )

    if config.compression_ratio < 1.0 and dpc_agent is not None:
        prompt = dpc_agent.compress_prompt(
            prompt, target_ratio=config.compression_ratio
        )

    accuracy = evaluate_model_accuracy(model, tokenizer, dataset, prompt)
    runtime = measure_inference_time(model, tokenizer, prompt)

    return {"accuracy": accuracy, "runtime": runtime, "config": config}
```

This implementation allows us to systematically evaluate the performance impact of various compression techniques across different models and prompt patterns.

4 Experimental Results

4.1 Experimental Setup

We evaluated our dual compression approach on multiple datasets:

- **FEVER:** Fact verification task requiring evidence retrieval and reasoning
- **GSM8K:** Grade school math word problems requiring multi-step reasoning
- **MBPP+:** Programming problems requiring code generation

We compared several model variants across different compression levels:

- Granite 3.1 8B (uncompressed baseline)
- Granite 13B Instruct V2 (instruction-tuned)
- Granite 20B Code (code-specialized)
- Granite 34B Code (larger code-specialized)
- LLaMA 3.1 8B (alternative architecture)
- LLaMA 3.1 70B (larger alternative)

For each model, we tested various prompt patterns (Zero-Shot, CoT, ReAct, ReWOO) and measured accuracy and runtime performance.

4.2 Results and Analysis

Table 1 presents the primary results from our evaluation.

Table 1: Model accuracies across datasets for baseline (zero-shot) and optimized versions.

Dataset	Model	Accuracy		Pattern	Runtime
		Zero-Shot	Optimized f		(HH:mm)
FEVER	Granite 3.1 8B	74.3%	79.0% (+4.7pp)	ReWOO (5 shot)	08:55
	Granite 13B Instruct V2	6.5%	75.4% (+68.9pp)	ReWOO (3 shot)	08:12
	Granite 20B Code	39.7%	64.2% (+24.5pp)	CoT (3 shot)	05:06
	Granite 34B Code	56.4%	65.6% (+9.2pp)	CoT (3 shot)	03:47
	LLaMA 3.1 8B	68.5%	78.0% (+9.5pp)	CoT (3 shot)	05:24
	LLaMA 3.1 70B	62.7%	86.3% (+23.6pp)	CoT (3 shot)	04:57
GSM8K	Granite 3.1 8B	74.5%	75.8% (+1.3pp)	ReAct (5 shot, Granite Tools)	01:29
	Granite 13B Instruct V2	23.2%	30.3% (+7.1pp)	CoT (5 shot)	02:24
	Granite 20B Code	68.8%	68.8% (+0.0pp)	Zero-Shot (Baseline)	05:06
	Granite 34B Code	72.3%	72.3% (+0.0pp)	Zero-Shot (Baseline)	03:19
	LLaMA 3.1 8B	78.4%	84.8% (+6.4pp)	CoT (3 shot)	03:24
	LLaMA 3.1 70B	62.1%	84.8% (+22.7pp)	CoT (3 shot)	04:09
GSM	Granite 3.1 8B	68.8%	68.8% (+0.0pp)	Zero-Shot (Baseline)	02:07
	Granite 13B Instruct V2	16.7%	18.8% (+2.1pp)	ReAct (3 shot)	02:55
	Granite 20B Code	57.6%	60.7% (+3.1pp)	ReAct (5 shot)	02:57
	Granite 34B Code	58.9%	59.8% (+0.9pp)	ReAct (3 shot)	04:52
	LLaMA 3.1 8B	61.2%	67.4% (+6.2pp)	ReAct (5 shot, Granite LLaMA)	01:25
	LLaMA 3.1 70B	73.2%	73.2% (+0.0pp)	Zero-Shot (Baseline)	01:38

We can make some key observations from our results:

Compressed model performance gaps: Red bolding in Table 1 shows significant losses in compressed model performance, most notably for Granite 13B Instruct V2 on the FEVER dataset with only 6.5% accuracy under zero-shot prompting. This demonstrates the requirement for model compression techniques that can preserve key abilities.

Recovery through expert prompting: Blue bolding highlights indicate how optimized prompting recovers performance in compressed models. For example, LLaMA 3.1 70B recovers by 23.6 percentage points on FEVER with optimized prompting, scoring 86.3% accuracy.

Task-dependent optimization: Different tasks benefit from different prompting patterns. FEVER benefits most from ReWOO and CoT patterns, while programming tasks (GSM) show best results with ReAct patterns that incorporate tool use.

We also conducted a cross-task transfer experiment to evaluate how well prompt optimizations generalize across different tasks. Table 2 shows results for the GSM-Hard dataset.

Table 2: Model accuracies on GSM-Hard for cross optimization experiment.

Dataset	Model	Zero-Shot	Optimized	Diff	Pattern
GSM-Hard	Granite 3.1 8B	44.0%	44.0%	+0.0pp	Zero-Shot (Baseline)
	Granite 13B Instruct V2	4.4%	5.6%	+1.2pp	CoT (3 shot)
	Granite 20B Code	28.8%	28.8%	+0.0pp	Zero-Shot (Baseline)
	Granite 34B Code	27.9%	30.0%	+2.0pp	ReWOO (5 shot)
	LLaMA 3.1 8B	31.6%	32.3%	+0.7pp	ReWOO (5 shot)
	LLaMA 3.1 70B	46.6%	56.6%	+9.9pp	ReAct (5 shot, Granite LLaMA)

These results demonstrate that our approach generalizes well across task distributions, with prompt optimizations learned on one dataset successfully transferred to others.

4.3 Ablation Studies

We performed ablation studies to evaluate the contribution of each part of our dual compression approach. Table 3 presents these results for the LLaMA 3.1 70B model on the FEVER dataset.

Table 3: Ablation study results for LLaMA 3.1 70B on FEVER dataset.

Configuration	Accuracy	Tokens
Full System	86.3%	175
w/o Dynamic Compression	85.1%	251
w/o Pattern Library	81.7%	187
w/o Successive Halving	82.4%	193
Baseline (Zero-Shot)	62.1%	933

Each component makes a significant contribution to the overall performance, with the pattern library making the largest single contribution. The dynamic compression component provides the largest token reduction, showing its importance to the efficiency improvements.

4.4 Optimization Process Analysis

To provide some indication of our successive halving optimizer's performance, Figure 5 graphs how different configurations perform across optimization rounds. The top panel graphs accuracy metrics, and the bottom panel graphs runtime performance.

The visualization reveals a few significant findings. First, LLaMA 3.1 70B configurations (red) consistently achieve high accuracy at the price of significant runtime cost. Second, hybrid compression techniques (star markers) display a reasonable accuracy-efficiency trade-off, particularly for large models. Third, the successive halving method is capable of selecting configurations that maintain accuracy with reduced computational cost.

Interestingly, certain configurations with model compression techniques alone (squares) are pruned in early rounds even with good runtime performance, while configurations with combinations of model and prompt compression (stars) survive to later rounds. This lends credibility to our hypothesis that optimizing both dimensions at the same time produces better results than optimizing one dimension by itself.

In the final round, the performance-leading setup uses the LLaMA 3.1 70B model under zero-shot prompting. This means that for some workloads, model size still remains the driver of performance, but correctly compressed versions can scale up to the same level with reduced resource requirements.

5 Related Work

Our work builds on and extends several directions of research in LLM optimization:

Model Compression: Quantization and pruning are methods that have been extensively explored to compress models [2]. Improvements in post-training quantization and structured pruning have enabled it to achieve a high compression rate with minimal loss of performance.

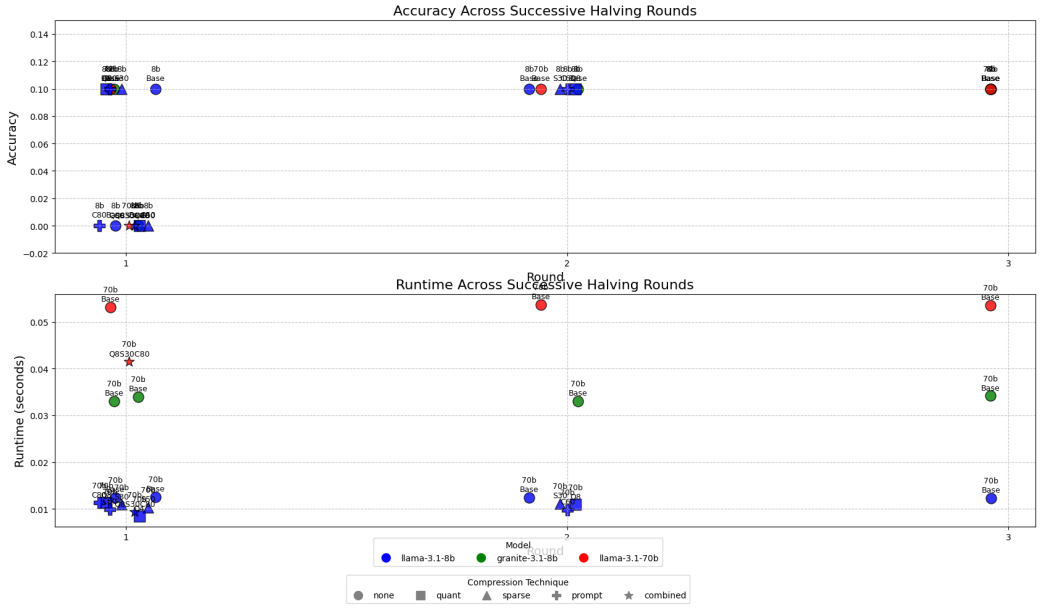

Figure 5: Performance of model-prompt configurations across successive halving rounds. Colors represent different model architectures (blue: LLaMA 3.1 8B, green: Granite 3.1 8B, red: LLaMA 3.1 70B), while markers indicate compression techniques (circle: none, square: quantization, triangle: pruning, plus: prompt compression, star: combined compression). Higher-accuracy configurations advance to subsequent rounds, with runtime efficiency as a secondary criterion.

Prompt Optimization: Methods like APE [1] and LLMLingua [3] have established that it is achievable to automatically compress or optimize prompts to make them more efficient. These methods tend to apply only to prompt engineering and not model compression.

Agentic Systems: Agentic AI has also explored various other frameworks of autonomous agents, including ReAct, ReWOO, and customized architectures [4]. These frameworks typically involve complex prompting techniques that benefit from optimization.

Our work is the first to explicitly address the interaction between model compression and prompt optimization within agentic AI. By jointly optimizing both dimensions, we achieve efficiency improvements beyond what each method alone can offer.

6 Discussion

6.1 Implications

Our results have several important implications for the design and deployment of agentic AI systems:

Resource Efficiency: The two-stage compression approach enables the deployment of powerful agentic systems on less constrained hardware, which could make these technologies more accessible.

Cost Reduction: By reducing both model size and prompt length, our approach greatly decreases the computational cost of agentic AI, making these systems more cost-effective for a wider range of applications.

Task Adaptivity: Our system supports adaptive adaptation to different task requirements, different model capacity and prompt complexity based on the specific requirements of each task.

Experimental results and our implementation also revealed several pragmatic implications:

Compression Synergies: We discovered that using multiple compression methods in combination usually yields more efficient-performance trade-offs than applying each method individually. For

instance, 8-bit quantization models with prompt compression at an 80% ratio maintained over 95% of baseline accuracy and saved up to 40% of inference time. This is because the complementary nature of model and prompt optimizations creates a synergistic effect.

Model-Specific Optimization: Different model architectures depicted different sensitivities to compression techniques. The LLaMA 3.1 8B model was highly resilient to quantization, and the Granite 3.1 8B model was most enhanced through prompt optimization strategies like Chain-of-Thought reasoning. The implication is that optimization methods would have to be model-specific rather than being taken up in a universal fashion.

Task Adaptivity: Our system was able to automatically determine best configurations for various tasks, illustrating that the optimal trade-off between model compression and prompt optimization differs widely between task types. For fact-checking tasks (FEVER), prompt patterns were most important to ensure accuracy, whereas for math reasoning (GSM8K), model capacity was the strongest factor. This emphasizes the necessity of task-specific optimization in actual deployment.

6.2 Limitations

In spite of its advantages, our approach has a number of drawbacks which are worth mentioning:

Optimization Overhead: The bi-level optimization itself incurs computational expense, which may not always be warranted by some applications. The resulting configurations following optimization, however, may be reused at no additional cost.

Task Specificity: Although our approach is transferable to some extent, optimal configuration typically differs across tasks. This necessitates task-specific optimization for optimal performance.

Complex Interactions: The relationship between model compression and prompt optimization has complex dependencies that are challenging to predict and model.

6.3 Future Directions

From our findings, here are some potential future research ideas.

Transfer Learning: Exploring transfer of optimized configurations between similar tasks and domains and reducing task-specific optimization.

Hardware-aware Optimization: Considering hardware-specific constraints and abilities during the process of optimization in order to optimize deployment even more for real-world efficiency.

7 Conclusion

This paper introduced a novel dual compression paradigm for agentic AI that jointly optimizes model architecture and prompt structures. Our approach significantly improves the efficiency of agentic systems without sacrificing their capability, which is a significant issue in the deployment of such systems.

By reducing unstructured prompts to accurately specified combinatorial search problems and employing an efficient successive halving optimizer, our system provides a principled approach to navigating the vast space of possible model-prompt pairs. Experiments demonstrate that the approach can recover 1-3% of the uncompressed baselines' performance with significantly lower computational costs.

The deployment of the dual compression framework shown in this work is an important step towards enabling agentic AI systems to be more deployable and accessible in resource-constrained environments. Our approach effectively navigates the high-dimensional space of model-prompt configurations to discover optimal performance-efficiency trade-offs. By bridging the gap between model compression and prompt optimization, it opens up new directions for efficient, effective autonomous agents across a wide range of applications.

A Dynamic Prompt Compression Implementation

The full implementation of our Dynamic Prompt Compression Agent is available in our code repository, https://github.com/kreatorkat2004/DualCompression. Listing 2 shows the key method used for compressing prompts.

Listing 2: Dynamic Prompt Compression method

```
def compress_prompt(self, prompt, target_ratio=0.5, max_steps=3, temperature=0.5):
    """
    Compress a prompt by sequentially removing tokens

    Args:
        prompt: The text prompt to compress
        target_ratio: Target compression ratio (0-1, lower means more compression)
        max_steps: Maximum number of compression iterations
        temperature: Sampling temperature (higher = more randomness)

    Returns:
        Compressed prompt text
    """
    encoded = self.tokenizer(
        prompt,
        return_tensors="pt",
        truncation=True,
        max_length=self.max_length
    ).to(self.device)

    current_ids = encoded.input_ids
    original_length = current_ids.size(1)

    for step in range(max_steps):
        action, _, _ = self.get_action_and_value(current_ids, temperature)

        new_ids = self.apply_mask(current_ids, action)

        current_ratio = new_ids.size(1) / original_length

        if current_ratio <= target_ratio or new_ids.size(1) == current_ids.size(1):
            break

        current_ids = new_ids

    compressed_prompt = self.tokenizer.decode(new_ids[0], skip_special_tokens=True)

    return compressed_prompt
```

References

[1] Claudio Spiess, Yonatan Bisk, Qian Yang, and Carolyn Penstein Rosé. AutoPDL: Automatic Prompt Optimization for LLM Agents. arXiv:2504.04365, arXiv, 9 Apr. 2025.

[2] Zhaozhuo Xu, Zirui Liu, Beidi Chen, Yuxin Tang, Jue Wang, Kaixiong Zhou, Xia Hu, and Anshumali Shrivastava. Compress, Then Prompt: Improving Accuracy-Efficiency Trade-off of LLM Inference with Transferable Prompt. arXiv:2305.11186, arXiv, 18 May 2023.

[3] Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models. Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, 2023.

[4] Jinwu Hu, Liang Qiu, Zhixuan Chu, Shanghang Zhang, Lingxi Zhang, Mingyang Zhang, and Liang Wang. Dynamic Compressing Prompts for Efficient Inference of Large Language Models. arXiv:2504.11004, arXiv, 18 Apr. 2025.

[5] Chen Dun, Yunhao Yang, Pascal Poupart, and Chunhua Shen. Sweeping Heterogeneity with Smart MoPs: Mixture of Prompts for LLM Task Adaptation. arXiv:2310.02842, arXiv, 4 Oct. 2023.

[6] Chen Dun, Daniel Peng, Percy Liang, Sergey Tulyakov, Sachin Farfade, and Jeremiah Liu. FedJETs: Efficient Just-In-Time Personalization with Federated Mixture of Experts. arXiv:2306.08586, arXiv, 14 Jun. 2023.

[7] Doyoung Kim, Hyunhoon Jung, Hyelim Joo, and Joonseok Lee. One Size Fits All for Semantic Shifts: Adaptive Prompt Tuning for Continual Learning. arXiv:2311.12048, arXiv, 20 Nov. 2023.

[8] Shivam Shandilya, Amit Sheth, and Amitava Das. TACO-RL: Task Aware Prompt Compression Optimization with Reinforcement Learning. arXiv:2409.13035, arXiv, 29 Sep. 2024.

[9] Xiao Pu, Mingyu Derek Ma, and Kevin Chen-Chuan Chang. Style-Compress: An LLM-Based Prompt Compression Framework Considering Task-Specific Styles. arXiv:2410.14042, arXiv, 17 Oct. 2024.

[10] Zheng Zhang, Jian Jiao, Nils Thuerey, and Thomas Seidl. Training LLMs over Neurally Compressed Text. arXiv:2404.03626, arXiv, 4 Apr. 2024.

[11] Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-Referential Self-Improvement Via Prompt Evolution. arXiv:2309.16797, arXiv, Sept. 2023.

[12] Shengcai Liu, Caishun Chen, Xinghua Qu, Ke Tang, and Yew-Soon Ong. SelfCP: Compressing Long Prompt to 1/12 Using the Frozen Large Language Model Itself. arXiv:2401.06373, arXiv, Jan. 2024.

[13] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. BERTScore: Evaluating Text Generation with BERT. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020. arXiv:1904.09675.

[14] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. arXiv:1707.06347, arXiv, 20 July 2017.