Distilling Sequential Computation in Transformer Language Models

Zixuan Lan[1] **Jessica Yang**[2] **Yanhong Li**[3] **Karen Livescu**[2] **Jiawei Zhou**[4]

[1]The University of Chicago [2]Toyota Technological Institute at Chicago
[3]Independent Researcher [4]Stony Brook University

zixuanlan@uchicago.edu jiaminy@ttic.edu klivescu@ttic.edu
yanhong.lbh@gmail.com jiawei.zhou.1@stonybrook.edu

Abstract

Transformer language models process sequences token by token in an autoregressive manner, making growing contexts increasingly expensive. Yet many adjacent token spans are highly predictable or frequently occur as stable units, suggesting that their representations may be compressible. We introduce a method for distilling sequential computation by replacing spans of input tokens with collapsed representations, computed on the fly by a lightweight merge module. This module generates a single surrogate embedding from a sequence of static token embeddings that captures the functional role of the multiple tokens, allowing pretrained models to operate on compressed inputs without architectural changes or re-training. We apply this approach during inference to compress both prompts and intermediate decoding steps, using a rollback mechanism to substitute stored multi-token KV cache entries with their single-step surrogates. Experiments across diverse models show that the merge module can be used to reduce effective sequence length by up to 40% with minimal accuracy degradation across language modeling evaluations and downstream tasks, including question answering, summarization, commonsense reasoning, and long-form mathematical reasoning. Additional lightweight adaptation of the merge module further improves the accuracy-compression trade-off in selected settings. These results demonstrate that sequential token computation in Transformers can be effectively approximated through condensed surrogate representations that approximate the original behavior without model updating. Code and project resources are available at `https://github.com/Zesearch/Umim-LLM`.

1 Introduction

Autoregressive large language models (LLMs), typically built on Transformer architectures [Vaswani et al., 2017], process sequences token by token, making long inputs and growing autoregressive contexts expensive to compute and cache. Yet many adjacent token spans are highly predictable or frequently recurring in a sequence but are still processed as separate tokens. For example, "machine learning" or "New York" often appear together and are easy to predict once the first token is observed.

Prior work has explored reducing sequential computation in Transformer language models [Lan et al., 2026] through alternative tokenization, multi-token decoding, and context reduction. Tokenization-based methods modify the input units on which models are trained, including byte- or patch-level representations [Pagnoni et al., 2024, Xue et al., 2022, Yu et al., 2023] and larger "superword" units [Liu et al., 2025], but typically require re-training because tokenization defines the model's input structure. Other methods target pre-trained models through extended vocabularies, drafting models, or external chunk representations [Lan et al., 2023, Leviathan et al., 2023, Li et al., 2025b], but introduce auxiliary components or inference overhead. Context-reduction methods prune or compress

inputs, hidden states, or KV caches [Zhang et al., 2023, Bolya et al., 2023, Mu et al., 2023, Kallini et al., 2024, Jiang et al., 2023, Xiao et al., 2024, Shao et al., 2024, Li et al., 2025a], but often rely on model-internal signals, custom compression objectives, or accuracy-efficiency trade-offs. It remains unclear whether multi-step Transformer computation can be replaced at inference time with single-step surrogates without accessing internal activations or updating the pretrained backbone.

In this work, we address the above question by distilling the sequential computation of multiple tokens into a single-step input representation for pre-trained language models. Given a pre-trained language model (LM) and a span of input tokens in context during generation, we aim to construct a *single surrogate embedding* that replaces the original token span *at the input level*, while preserving the model's next-token distribution.

To handle the large number of possible token spans during inference, we introduce **U**niversal **M**ulti-step **I**nput **M**erging (UMIM), a shared, lightweight merge module that computes surrogate embeddings on demand. UMIM takes the static embeddings of the tokens in a selected span and computes a single surrogate embedding. The resulting surrogate is span-specific but context-independent: For the same n-grams , the module produces the same surrogate wherever it appears, while different n-grams naturally produce different surrogates.

We implement the UMIM module as a lightweight, single-layer attention network that serves as a plug-in to any pre-trained Transformer language model, applied to input embeddings *prior to model computation* at inference. UMIM is trained using a predictive distillation loss, encouraging the model's output distribution with merged input embeddings to match that of the original uncompressed token sequences. Crucially, the pre-trained language model remains *frozen*, with only the external merge module updated during training. Once trained, the same UMIM module can be applied across different downstream tasks without re-training, offering a general and reusable mechanism for input-level sequential computation compression. Building on this general module, downstream performance can be further improved through task-specific merge-rule construction and lightweight adaptation of the merge module, while keeping the backbone language model frozen.

Using the shared UMIM module, we compress both static prompts and autoregressive decoding contexts, with rollback replacing multi-token KV-cache entries by single-step surrogate states. Experiments across diverse models and tasks show that UMIM directly reduces effective sequence length by up to 40% with minimal accuracy degradation, while preserving next-token predictive behavior.

2 Related Work

Tokenization Variations and Extension Prior work explores alternative tokenization strategies for Transformer language models. Byte-level and tokenizer-free models such as ByT5 [Xue et al., 2022], Megabyte [Yu et al., 2023], and ByteFusion [Pagnoni et al., 2024] reduce reliance on predefined vocabularies, while SuperBPE [Liu et al., 2025] introduces larger phrase-level units to reduce sequence length. Other methods extend token representations for span- or chunk-level decoding, including CoG [Lan et al., 2023], CD-LM [Li et al., 2025b], and adaptive hypertoken construction such as zip2zip [Geng et al., 2025]. However, these approaches generally require modifying the tokenizer, extending the token space, introducing auxiliary retrieval or encoding components, or retraining the model. In contrast, our method constructs surrogate embeddings for token spans post-training, without altering the tokenizer, vocabulary, or model architecture.

Latent Context Compression A closely related line of work compresses long contexts into learned continuous representations, including Gist Tokens [Mu et al., 2023], ICAE [Ge et al., 2024], Extensible Tokenization [Shao et al., 2024], 500xCompressor [Li et al., 2024], and multimodal processing of textual information via images [Li et al., 2025a]. These methods encode inputs into compact vectors or special-token representations that are later consumed by the language model. While effective, they typically require an additional encoding stage and treat compression as a separate interface. By contrast, our approach stays within the original token-based generation pipeline: It constructs surrogate embeddings for token spans directly at the input level and replaces multi-step computation with single-step surrogates during inference.

Prompt Compression and Context Reduction Prompt and context compression methods reduce inference-time computation through task-aware or task-agnostic content reduction. Task-aware

methods such as LongLLMLingua [Jiang et al., 2024], Recomp [Xu et al., 2024], and also CPC [Liskavets et al., 2024] retain content relevant to downstream questions or tasks. Task-agnostic methods, including ProCut [Xu et al., 2025], EHPC [Fei et al., 2025], LLMLingua [Jiang et al., 2023], LLMLingua2 [Pan et al., 2024], and self-information filtering [Li, 2023], estimate token or segment importance to remove less informative content. Other work studies abstractive prompt compression, where prompts are rewritten into shorter natural-language forms, as in Nano-Capsulator [Chuang et al., 2024] and Cmprsr [Zakazov et al., 2026]. In contrast, our approach is also task-agnostic but does not delete, filter, or rewrite input content.

KV Cache Management, Token Pruning and Merging Recent work reduces inference memory and computation through KV cache management, token pruning, or token merging. KV cache methods either statically filter tokens during prefill or dynamically retain, evict, or offload tokens during decoding, as in H_2O [Zhang et al., 2023], StreamingLLM [Xiao et al., 2024], KeyFormer [Adnan et al., 2024], OmniKV [Hao et al., 2025], PagedAttention [Kwon et al., 2023], and Blockwise Caching [Dao et al., 2022]. Other methods reduce context length through token pruning or merging, including MRT5 [Kallini et al., 2024] and hidden-state-based merging in language [Yuan et al., 2024], vision [Bolya et al., 2023, Shin et al., 2025], video [Ryoo et al., 2021]. In contrast, our method performs merging before Transformer computation rather than by pruning existing KV entries or combining hidden states during model execution. It replaces multi-token spans with single surrogate input embeddings and substitutes their corresponding multi-step KV cache entries with single-step surrogate states, preserving the functional role of the original tokens while reducing both effective sequence length and KV cache usage.

3 Methodology

We develop UMIM as an external plug-in for pre-trained Transformer language models, which maps selected contiguous token spans to single surrogate embeddings computed from their static input embeddings. Formally, let $\mathbf{x} = (x_1, \ldots, x_T)$ be a sequence of input tokens, and let $e(\cdot)$ denote the token embedding lookup, applied element-wise so that $e(x_i)$ is a single embedding and $e(x_{a:b})$ denotes the corresponding sequence of embeddings. A Transformer language model with parameters θ processes the embedded prefix $e(x_{<t})$ and outputs the next-token distribution $p_\theta^t(\cdot \mid e(x_{<t}))$. We learn a merge module $\mathcal{M}_\phi$, parameterized by ϕ, that maps a span $x_{t:t+n}$ to a surrogate embedding $\tilde{e} = \mathcal{M}_\phi(e(x_{t:t+n}))$, which replaces the original embeddings while preserving the subsequent predictive distribution. The following sections describe merge span selection, the merge module architecture, the distillation objective, and decoding algorithms for applying UMIM during prompt compression and generation.

3.1 Merge Rule

To determine which token spans to merge, we construct a frequency-based merging set from a general corpus $\mathcal{C}$. Specifically, we collect unique contiguous n-grams of length $2 \leq n \leq 4$ whose frequency exceeds a threshold τ, forming the merging set $\mathcal{R}$. High-frequency spans are likely to recur during inference, enabling more frequent compression, and also provide sufficient training instances for learning surrogate embeddings. The maximum of $n = 4$ mitigates data sparsity while still allowing meaningful multi-token groupings. When multiple candidates overlap, we apply a longest-match strategy, giving precedence to the longest valid span at each position. During training and inference, spans that match entries in $\mathcal{R}$ are treated as merge candidates. The merge rule is independent of the model architecture and depends only on the tokenizer and corpus statistics. The corpus $\mathcal{C}$ used to construct $\mathcal{R}$ is task-independent, though in-domain data can further improve compression coverage by capturing domain-specific frequent spans. See Appendix C for details.

3.2 Lightweight Merge Module Architecture

The UMIM is a lightweight single-layer multi-head attention pooling network that operates solely on static input embeddings, prior to any Transformer computation. It serves as a universal replacement for contiguous token spans, compressing variable-length inputs into a single surrogate embedding via a *learnable query* mechanism.

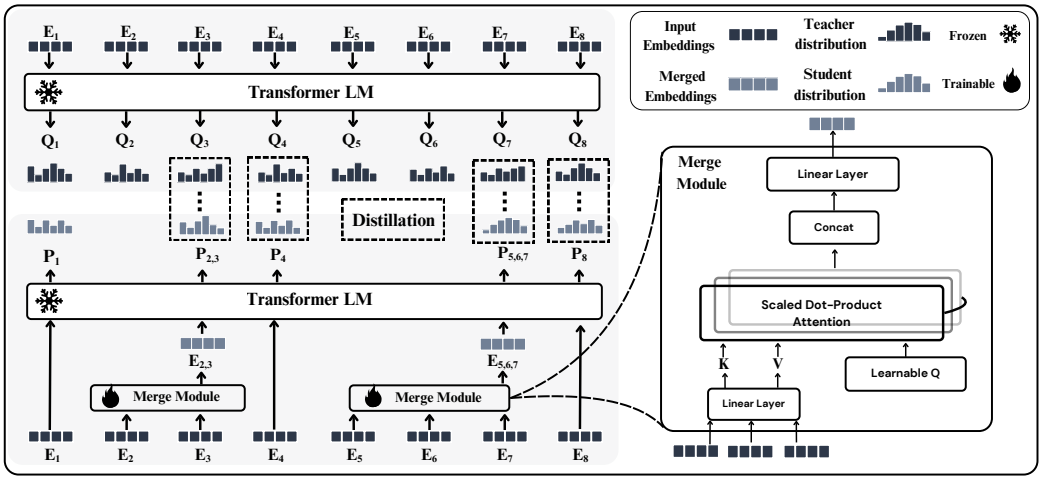


Figure 1: Training process of the UMIM module, with token-step aligned distillation loss and merge module architecture.

Formally, let d denote the embedding dimension of the target language model, and h the number of attention heads. Each head has dimension $d_h = d/h$. Given the token span embeddings $e(x_{t:t+n}) \in \mathbb{R}^{d\times n}$, we use a learnable query parameter $Q \in \mathbb{R}^{h\times d_h}$, where the ith row $q_i \in \mathbb{R}^{d_h}$ is the query vector for head i. We use a shared projection W^{kv} for span representations before attention pooling. After pooling, an output projection W^o maps the concatenated head outputs back to the model embedding space. The output surrogate embedding is computed as:

$$M_\phi(e(x_{t:t+n})) = W^o\left[\mathrm{o}_1;\cdots;\mathrm{o}_h\right]^\top, \quad \mathrm{o}_i = \mathrm{softmax}\left(\frac{q_i^\top W^{kv} e(x_{t:t+n})}{\sqrt{d_h}}\right)\left(W^{kv} e(x_{t:t+n})\right)^\top. \tag{1}$$

3.3 UMIM Learning by Distilling Sequential computation

We train the UMIM to produce single-step surrogate embeddings that can replace multi-tokens, while preserving the language model's next-token distributions. The language model parameters θ are frozen throughout training. To guide learning, we introduce a *sequential computation distillation loss*, aligning predictions made on merged inputs with those from the original token-by-token computation. Let $\mathcal{M}_\phi(e(x_{1:T})) = (e'_s)_{s\in\mathcal{S}}$ denote the compressed sequence of token embeddings after applying the merge module to all eligible token spans according to the merge rule $\mathcal{R}$. Here, $\mathcal{S}$ represents the set of *retained token positions* after merging. For instance, if tokens $x_{t:t+n}$ are merged into a single surrogate embedding $\mathcal{M}_\phi(e(x_{t:t+n}))$, then token steps t through $t+n-1$ are collapsed, and only step $t+n-1$ is retained in $\mathcal{S}$ (e.g., $\mathcal{S} = (\cdots, t-1, t+n-1, \cdots)$).

To align the behavior of the compressed input with the original, we minimize the KL divergence between the output distributions of the language model on unmerged vs. merged input prefixes:

$$\mathcal{L}_{\text{distill}} = D_{\mathrm{KL}}(p_\theta(\cdot \mid e(x_{\le s})) \parallel p_\theta(\cdot \mid M_\phi(e(x_{\le s})))) \tag{2}$$

This loss encourages the merged representation to produce predictive distributions that match those from the full token sequence. An illustration of the training procedure is shown in Figure 1.

3.4 Model Inference with Runtime Merging

Once trained, the UMIM module can be integrated into inference with the compatible pre-trained language model. During inference, UMIM reduces the effective sequence length by applying runtime merging in both prompt prefilling and autoregressive decoding as shown in Figure 2. Let $z = (z_1, \cdots, z_m)$ denote the prompt tokens, and $y = (y_1, y_2, \cdots)$ the generated tokens. During prefill, we scan the prompt for eligible n-grams in $\mathcal{R}$ and replace each matched span with its surrogate embedding. During decoding, after each new token is generated, we check whether the recent suffix forms an eligible n-gram; if so, we roll back the corresponding KV entries and replace them with

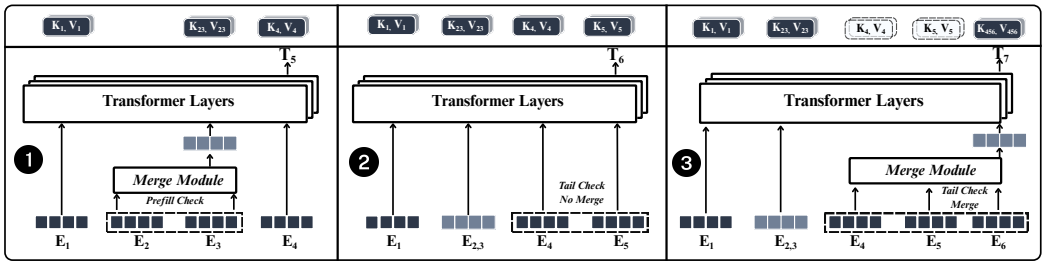


Figure 2: UMIM Decoding Process: **E** is embedding, **T** is token, **K,V** is KV cache (1) During prefill, mergeable spans are replaced by surrogate embeddings. (2) During autoregressive decoding, each new token is checked against the recent suffix; if no valid merge is found, decoding proceeds normally by appending a new KV cache. (3) If the suffix forms a mergeable span, UMIM rolls back the matched span and replaces its multiple KV entries with a single merged KV cache.

a single merged KV state. The checking and rollback overhead is amortized over long sequences, while the reduced sequence length directly lowers KV-cache size and attention computation.

By default, we apply the merge rule underlying the training data $\mathcal{R}$ for inference. This, however, can be relaxed by adapting the merge rule further to a specific downstream task based on in-domain data, facilitating a wider range of compression ratio and domain generalization. See Section 4.5 for details.

4 Experiments

4.1 Experimental Setup

We train UMIM modules for LMs including Llama 3.1 8B Grattafiori et al. [2024], Llama 3.2 1.5B, and GPT2-XL 1.5B Radford et al. [2019] using the WikiText-103 [Merity et al., 2016] corpus, and DeepScaleR-1.5B-Preview[1] using a long reasoning trajectory corpus we collected. Each merge module uses $h = 4$ attention heads and contains 7M–50M learnable parameters, depending on the backbone size. To build the merging rule $\mathcal{R}$, we extract $n \in \{2, 3, 4\}$-grams with threshold $\tau = 5$, yielding over 2M unique token spans per model. More details can be found in Appendix C

We first measure the distribution alignment quality after applying the embedding merges compared with the original token-based LLMs. The metrics include: Top-1 accuracy—percentage of token positions after merging where the top-1 tokens in the predictive distribution exactly match, Top-3/10 overlap—average number of overlapped tokens in the top-3/10 tokens from corresponding distributions at aligned token positions, Top-p overlap—the overlap between tokens whose cumulative probability mass reaches p in each distribution, and Mean Reciprocal Rank (MRR)—the average reciprocal rank of the teacher (original LLM)'s top-1 token within the distribution after span merging at corresponding token positions.

Further, we measure the language modeling performance using perplexity (PPL) with UMIM on different test datasets (Sec. 4.3), including WikiText-103 which the module is trained on, as well as BookCorpus [Zhu et al., 2015] and OpenWebText [Gokaslan and Cohen, 2019] with the same merge module without re-training. To quantify compression, we report token reduction (TR, %), which reflects the reduction in sequence length and KV cache usage.

Finally, we assess the **generalization** ability of UMIM on downstream tasks including question answering (QA), document summarization, and math reasoning (Sec. 4.4). In the **task-agnostic** setting, a merge module trained on WikiText-103 is directly applied to all downstream tasks without task-specific supervision or fine-tuning, evaluating its transfer across unseen domains. We further study a **downstream adaptation** setting, where the pretrained merge module is specialized to target tasks through task-specific merge-rule construction, fine-tuning, and preference optimization, while keeping the backbone language model frozen. We compare with representative context reduction and prompt compression baselines, including Select Context [Li et al., 2023], Llmlingua [Jiang et al., 2023], Llmlingua2 [Pan et al., 2024], H2O [Zhang et al., 2023], StreamingLLM [Xiao et al., 2024],

[1] `https://huggingface.co/agentica-org/DeepScaleR-1.5B-Preview`.

Table 1: Distribution alignment between the original LLMs and the merged-input models. We report all in percentage (%). See Sec. 4.1 for metric definitions.

Model	TR	Top-1	Top-3	Top-p	MRR
Llama-3.1-8B	37.8	79.2	77.6	92.9	87.1
Llama-3.2-1.5B	37.8	74.1	74.1	91.6	83.3
GPT2-XL	35.7	69.6	71.3	90.7	79.3
DeepScaleR-1.5B	54.0	77.0	69.3	91.0	86.0

Table 2: **Language modeling performance under token reduction.** We report perplexity (PPL; lower is better) and token reduction (TR, %; higher means more tokens are merged). Original denotes the base model. See reference plot in figure 5

Backbone	Method	WikiText-103		BookCorpus		OpenWebText		PPL Avg. ↓
		PPL ↓	TR (%) ↑	PPL ↓	TR (%) ↑	PPL ↓	TR (%) ↑	
Llama-3-8B	Original	13.4	–	15.3	–	9.2	–	
	SelectContext	65.0	36.0	46.4	14.0	18.5	13.0	43.3
	LLMLingua-2	76.5	36.0	75.5	14.0	22.9	13.0	58.3
	Merge Module	13.8	36.1	15.3	14.8	9.8	13.8	13.0
Llama-3.2-1B	Original	20.1	–	21.2	–	13.4	–	
	SelectContext	130.4	36.0	67.0	14.0	30.7	13.0	76.0
	LLMLingua-2	129.0	36.0	101.8	14.0	37.3	13.0	89.4
	Merge Module	23.4	36.1	22.0	14.8	14.7	13.8	20.7
GPT2-XL	Original	28.6	–	27.1	–	13.1	–	
	SelectContext	202.5	35.1	93.4	14.0	39.2	12.0	111.7
	LLMLingua-2	250.3	35.1	124.2	14.0	41.5	12.0	138.7
	Merge Module	37.9	35.1	30.0	14.1	14.4	12.5	27.4

CPC [Liskavets et al., 2024], and EHPC [Fei et al., 2025]. We also show more experimental results about the long context task in the Appendix I.

4.2 Predictive Distribution Alignment

Table 1 reports predictive distribution alignment results on the corresponding *test* split after training. Higher values indicate better agreement between the predictive distributions produced after token merging and those of the original token-based LLM at aligned positions. Across all backbones, UMIM achieves strong alignment scores while reducing the effective sequence length by 35.7%–54.0%. These results validate that the proposed merge module can preserve much of the original next-token behavior using only static input embeddings, providing a strong foundation for downstream compression and generation.

4.3 Language Modeling and Generation

Table 2 reports language modeling performance. All merge modules are trained on the WikiText-103 training split, and the merge rules are derived from the same corpus. Overall, UMIM preserves language modeling behavior across both in-domain and out-of-domain data. Llama-3.1-8B remains close to the original token-based model across all three datasets while achieving substantial token reduction, and Llama-3.2-1.5B shows the same trend with slightly larger degradation. GPT2-XL is the main exception, exhibiting a larger PPL increase after merging, likely due to its weaker backbone and smaller merge module; nevertheless, it remains stronger than external prompt-compression baselines at comparable compression rates. In terms of compression, the WikiText-103 test split, which matches the training domain of the merge module, naturally exhibits the highest token reduction, with nearly 40% of tokens removed. More importantly, UMIM still achieves non-trivial compression on unseen domains, yielding 12–15% token reduction on BookCorpus and OpenWebText without any retraining. This result suggests that the learned merge operator generalizes beyond the training corpus not only in predictive behavior, but also in its ability to induce effective sequence compression.

We also provide qualitative generation examples in Table 3, using greedy decoding with Llama 3.1 8B. In several cases, token merging leaves the generated output unchanged. In others, the output

Table 3: **Qualitative generations with dynamic token-span merging.** We compare continuations produced by the base model and by UMIM under the same prompt. Bold bracketed spans indicate token spans that are merged into a single surrogate embedding, either in the prompt or during autoregressive generation.

Prompt	Base Model	UMIM
Traditional Chinese literary criticism emphasized **[the life of the]** author when interpreting a work	The author's life was seen as the key to understanding the work.	The author's life **[was seen as]** the key **[to understanding the work]**.
In the spring, **[his youngest son]**, Ian, was born.	He was a beautiful baby boy, and he was the light of his father's life.	He **[was a beautiful baby]**, **[with a head of thick]**, dark **[hair and]** a round, chubby face.

differs slightly from the original model, but remains coherent and semantically plausible. These examples complement the quantitative PPL results by showing that UMIM can reduce the effective input length while largely preserving generation quality. More examples are provided in Appendix I.

4.4 Task-Agnostic Generalization to Downstream Tasks

QA and Summarization With the **same** merge modules trained on WikiText-103, we directly apply them to downstream QA and summarization tasks *without any task-specific tuning*. We evaluate zero-shot QA on PIQA [Bisk et al., 2020], Copa [Gordon et al., 2012], OpenBookQA [Mihaylov et al., 2018], ARC_Easy, and ARC_Challenge [Clark et al., 2018], and summarization on CNN/DailyMail [Nallapati et al., 2016] using Rouge [Ganesan, 2018] and BERTScore [Zhang et al., 2020]. For fair comparison, the token reduction (TR) of all baselines is matched to that of our method. Additional results, including code generation, are provided in Appendix I. Across QA benchmarks, UMIM consistently provides the strongest or most robust accuracy–compression trade-off among the compared methods. This holds across all three backbones and across tasks entirely unseen during training, showing that the same pretrained merge module can transfer directly to unseen downstream tasks without task-specific tuning. A similar pattern is observed on CNN/DailyMail summarization. Under the same controlled token reduction, UMIM remains substantially closer to the original model than all compared baselines, achieving the best Rouge and BERTScore among compressed variants. This further suggests that the learned merge operator preserves generation quality beyond classification-style QA tasks.

Table 4: Performance of UMIM against compression baselines. Results are reported as correct/total problems. TR (%) measures tokens removed.

Method	AIME	TR(%)	AMC	TR(%)
DeepScaleR	15/30	–	37/40	–
UMIM	**13/30**	40.0	**32/40**	42.4
SelectContext	8/30	40.0	21/40	42.4
LLMLingua2	11/30	40.0	28/40	42.4

Advanced Math Reasoning We further evaluate UMIM on the AIME 2024 [Mathematical Association of America, 2024] and, we allow up to 15K generation tokens and evaluate pass@k accuracy under extended chain-of-thought decoding. As shown in Table 4 AMC mathematics benchmarks using DeepScaleR, where the merge module is trained on collected long chain-of-thought reasoning trajectories. To specifically test UMIM in a **long-form reasoning setting**, UMIM remains clearly more robust than Select Context and LLMLingua2 under more aggressive compression, retaining strong reasoning performance at over 40% token reduction. Together with the QA and summarization results, this shows that the same pretrained merge operator transfers effectively beyond its training objective, including challenging long-form generation settings with substantial reasoning depth.

4.5 Task-Specific Adaptation of a Pretrained Merge Module

We have shown that a merge module trained on WikiText-103 transfers well to unseen downstream tasks. We denote this module `WT`. The module itself is general, but the WikiText-derived merge rules are not equally well matched to every target task. We therefore study whether UMIM can obtain a better accuracy–compression trade-off by adapting its merge rules and lightweight merge module to the target domain, while keeping the backbone LLM frozen. We call this **Task Adaptation**. Task adaptation consists of two conceptually separate operations. First, we re-mine high-frequency token

Table 5: QA task performance of our UMIM against compression baselines. Acc (%) measures accuracy on each benchmark; TR (%) measures the percentage of tokens removed from the input. Bold denotes the best result among compression methods. See reference plot in figure 6

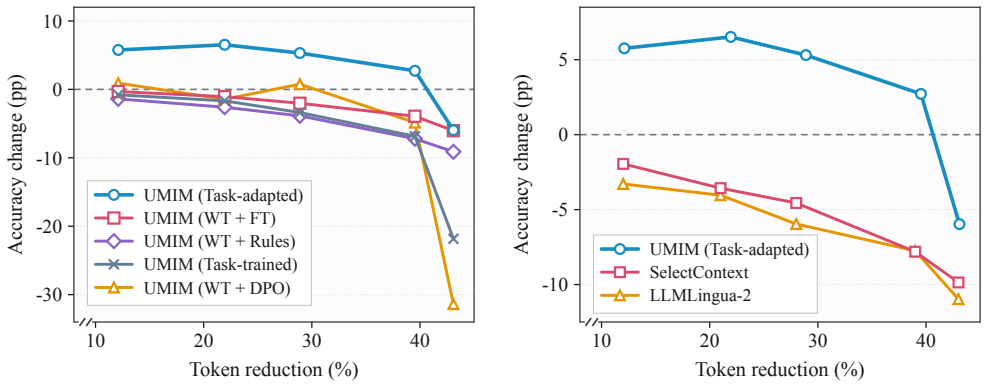
Methods	PIQA		Copa		OpenBookQA		ARC-Easy		ARC-Challenge		Avg (%)
	Acc (%)	TR (%)	Acc (%)	TR (%)	Acc (%)	TR (%)	Acc (%)	TR (%)	Acc (%)	TR (%)	
Llama-3.1-8B	79.6	–	76.8	–	43.6	–	76.5	–	49.5	–	65.2
UMIM	**78.4**	10	**76.8**	6	**42.4**	10	**73.7**	14	**44.8**	14	**63.2**
Select Context	73.3	10	72.6	6	40.8	10	57.2	14	43.5	14	53.5
LLMLingua-2	75.1	10	72.2	6	32.8	10	44.0	14	32.8	14	51.4
CPC	68.7	10	73.4	6	32.0	10	42.5	14	33.8	14	50.1
EHPC	72.7	10	64.6	6	28.2	10	38.3	14	28.8	14	46.5
Llama-3.2-1B	75.5	–	70.6	–	36.0	–	58.8	–	35.5	–	56.3
UMIM	**72.2**	10	68.8	6	31.2	10	**55.3**	14	**33.4**	14	**52.3**
Select Context	68.8	10	65.8	6	**35.0**	10	46.5	14	32.8	14	49.7
LLMLingua-2	69.1	10	**69.2**	6	31.2	10	37.9	14	26.4	14	46.8
CPC	52.7	10	68.6	6	29.4	10	53.2	14	31.4	14	47.1
EHPC	69.4	10	60.4	6	27.0	10	35.8	14	25.1	14	43.5
GPT2-XL	70.3	–	65.2	–	28.0	–	47.7	–	29.8	–	48.2
UMIM	**68.8**	9	62.0	9	**29.0**	10	**43.3**	14	**26.8**	14	**46.0**
Select Context	64.8	9	61.0	9	**29.0**	10	40.9	14	25.4	14	44.2
LLMLingua-2	66.6	9	**63.8**	9	26.4	10	30.7	14	25.8	14	42.6
CPC	63.2	9	61.8	9	28.0	10	34.6	14	27.8	14	43.1
EHPC	65.7	9	57.0	9	25.8	10	32.8	14	20.1	14	40.3

Table 6: CNN/DailyMail summarization performance of our Merge Module against compression baselines on Llama-3.1-8B. TR (%) measures the percentage of tokens removed from the input. R-1, R-2, R-L, and R-Lsum denote ROUGE-1, ROUGE-2, ROUGE-L, and ROUGE-Lsum scores respectively; BERTScore measures semantic similarity. All metrics are higher-is-better. Bold denotes the best result among compression methods. See reference plot in figure 7

Method	TR (%) ↑	R-1 ↑	R-2 ↑	R-L ↑	R-Lsum ↑	Avg ↑	BERTScore ↑
Llama-3.1-8B	–	37.4	15.6	24.3	31.3	27.2	86.8
UMIM	16	**35.5**	**13.9**	**23.1**	**29.7**	**25.5**	**86.4**
SelectContext	16	33.5	11.2	21.4	27.6	23.4	85.1
LLMLingua	16	33.6	10.3	21.3	27.7	23.2	85.3
LLMLingua2	16	33.0	11.3	23.0	28.8	24.0	85.0
StreamingLLM	16	28.8	10.1	18.9	24.3	20.5	84.5
H2O	16	24.4	9.3	16.3	21.2	17.8	82.7

spans from the training split of the target task and use them to replace the original WikiText-derived merge rules. Second, we update only the parameters of the merge module M_ϕ, first through supervised fine-tuning and then through RL. For the experiments in this section, we re-mine task-specific bigram rules only. Consequently, the maximum possible token reduction would be 50%. We vary the frequency threshold used to retain bigrams, producing several operating points with different merge rates. We compare five settings on HellaSwag: `Task-trained:` we discard `WT` and train a new merge module from scratch using task-specific merge rules.; `WT + Rules:` we replace the WikiText rules with task-specific rules but keep the pretrained `WT` merge module fixed; `WT + FT:` Starting from `WT`, we apply supervised fine-tuning to M_ϕ using the task-specific rules; `WT + DPO:` we apply DPO directly to M_ϕ, without the preceding supervised fine-tuning stage; and `Task-Adapted`, our full route defined as `WT + Rules + FT + DPO`. We first introduce task-specific rules, then fine-tune M_ϕ, and finally continue its adaptation with DPO. Across all five settings, the tokenizer and backbone language model remain unchanged and frozen. Thus, the additional task-specific cost is limited to collecting bigram statistics and, when applicable, updating the small merge module.

HellaSwag. Figure 3 separates the contributions of task-specific rules, WikiText initialization, supervised fine-tuning, and DPO. When the original WikiText rules are used directly on HellaSwag, their coverage is limited because the span distribution differs from that of WikiText-103. Re-mining bigrams from the HellaSwag training split produces operating points with merge rates from 12.10%



(a) UMIM task-adaptation variants (b) UMIM vs. compression baselines

Figure 3: **Task-specific adaptation on HellaSwag.** The y-axis reports accuracy change relative to the uncompressed model (0 = baseline). (a) We compare five adaptation settings, four of which build on the WikiText-103 pretrained module (`WT`); `Task-Adapted` (`WT + Rules + FT + DPO`) achieves the best accuracy–compression trade-off. (b) The strongest adapted variant also consistently outperforms external baselines at comparable merge rates.

to 43.09%. The `WT + Rules` setting shows that the WikiText-trained merge module can immediately operate with a newly constructed rule set, without any parameter updates. Its accuracy decreases as increasingly aggressive rules are introduced, but it provides substantially greater task-specific coverage. This result indicates that the learned merge operation is not tied to the specific spans seen during WikiText training. Supervised fine-tuning consistently improves upon `WT + Rules` across merge rates. It allows the merge module to accommodate the new span distribution while retaining the strong initialization learned from WikiText-103. In contrast, training the merge module from scratch (`Task-trained`) is less effective, showing that the transferable representation learned by `WT` remains useful even after the rules have been replaced. Applying DPO directly to `WT` is less stable, particularly at high merge rates. The complete `Task-Adapted` route instead performs supervised fine-tuning before DPO and achieves the strongest accuracy–compression trade-off. It reaches 84.86% accuracy at a 21.94% merge rate, compared with 78.34% for the uncompressed backbone. Even at a 39.53% merge rate, it obtains 81.06% accuracy, remaining 2.72 percentage points above the uncompressed baseline. Figure 3(b) also shows that `Task-Adapted` consistently outperforms other baselines at comparable merge rates.

ARC-Easy and ARC-Challenge. We further evaluate the same adaptation strategy on ARC-Easy and ARC-Challenge, as shown in Figure 4. Before task-specific training, `WT` already provides non-trivial compression on both tasks: it achieves 73.68% accuracy with 13.76% token reduction on ARC-Easy and 44.82% accuracy with 14.18% token reduction on ARC-Challenge. Adapting only the merge module substantially improves both operating points. On ARC-Easy, `Task-Adapted` reaches 77.88% accuracy with 22.00% token reduction, exceeding the uncompressed baseline of 76.49%. On the more difficult ARC-Challenge benchmark, it improves from the 49.50% uncompressed baseline to 54.85% accuracy while reducing the number of tokens by 20.00%. These results show that the benefit of task-specific adaptation is not limited to HellaSwag: the WikiText-trained module provides a reusable initialization, and lightweight downstream adaptation can simultaneously increase compression and recover—or even improve—task accuracy.

4.6 Ablation Study and Efficiency Analysis

We conduct ablations along two dimensions: merge-rule construction and merge-module design. Overall, the results show that UMIM is not highly sensitive to a single rule-construction strategy, but it is sensitive to overly aggressive merging. They also show that the current merge module achieves a strong accuracy–compression trade-off under a lightweight parameter budget, with limited gains from further increasing capacity. For merge rules (in Appendix K.3), we study different frequency thresholds, span construction strategies, and alternative rule-construction methods. The results show that setting the frequency threshold τ too low introduces many low-quality merges and leads to

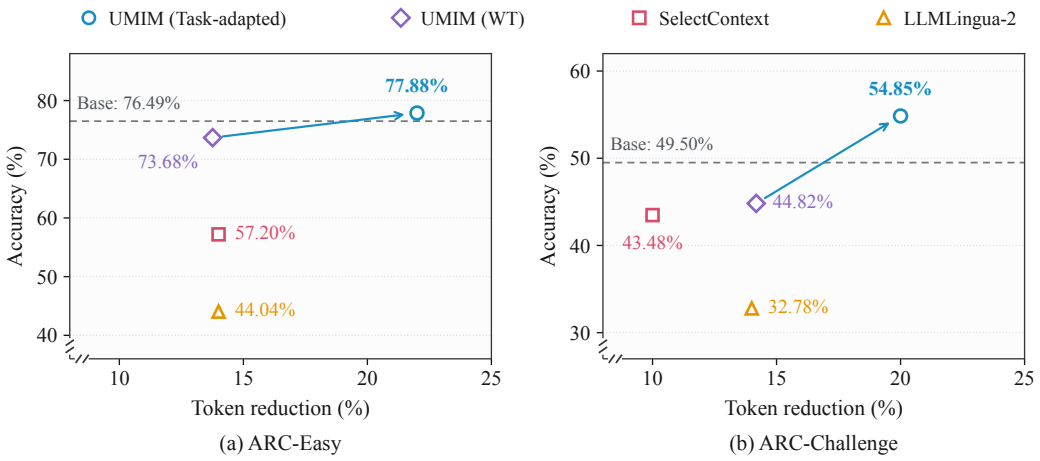


Figure 4: **Accuracy–compression trade-off on ARC-Easy and ARC-Challenge.** Each point shows a method's accuracy (%) vs. token reduction (%). The dashed horizontal line indicates the uncompressed baseline accuracy. The arrow marks the effect of downstream adaptation from `WT` to `Task-Adapted`.

clear performance degradation. This suggests that avoiding over-merging is more important than relying on a particular rule heuristic. Across the evaluated rule-construction strategies, we do not observe large performance differences, indicating that UMIM is reasonably robust to the exact rule design as long as the resulting merges provide sufficient training signal and do not over-compress the input. For the merge module (in Appendix K), we compare different parameter sizes and structural configurations. The current attention-based design already performs well with a small parameter budget, while increasing capacity yields only marginal improvements. We also find that simpler linear variants or deeper but less efficient structures do not provide better trade-offs. These results suggest that the current module design is effective for learning surrogate span representations, and that UMIM does not primarily rely on brute-force scaling to achieve strong compression performance.

Following the evaluation protocol of Zip2Zip [Geng et al., 2025], we conduct an end-to-end throughput study on Llama-3.1-8B with a fixed 50% merge ratio. We report tokens/sec for both prefill and decoding, with each setting in Table 19 denoted as `prefill length + decode length`. As shown in Table 19, UMIM provides substantial prefill speedups, with larger gains at longer input contexts because merging directly reduces the effective prompt length. Decoding throughput also improves consistently across settings. These results show that UMIM improves end-to-end inference efficiency in both prompt encoding and autoregressive generation.

5 Conclusion

We introduced UMIM, an token merging framework. Rather than deleting tokens or modifying the Transformer architecture, UMIM maps a matched multi-token span to a single surrogate embedding through a lightweight merge module. In our base setting, high-frequency n-gram spans are mined from base corpus to construct the merge rules, and the merge module is trained by distilling the behavior of the original uncompressed sequence. The resulting module operates only on static input embeddings, while the backbone language model remain unchanged. At inference, UMIM first merges eligible spans in the prompt before prefill. During decoding, it generates every token as usual and then checks whether the new suffix matches a merge rule. If a match is found, UMIM replaces the corresponding KV entries with one merged entry, shortening the sequence and KV cache used in later steps without skipping any token generation. Our experiments show that the WikiText-trained merge module transfers directly to unseen corpora and downstream tasks without retraining. For a new task, simply re-mining the merge rules increases coverage, while fine-tuning followed by DPO further improves the accuracy–compression trade-off. Only the merge module is adapted; the backbone language model remains frozen throughout.

Limitations

Our current study focuses on establishing UMIM as a practical input-level compression framework for frozen language models across representative models and tasks. Future work may explore more adaptive merging strategies, broader model scales and domains, and the interaction between pretrained merge modules and downstream adaptation.

Acknowledgment

We thank the Google Gemma Academic Program for their partial support of Jiawei Zhou and for providing computational resources. Jiawei Zhou is also supported by an Amazon Research Award and a Stony Brook OVPR Seed Grant.

References

Muhammad Adnan, Akhil Arunkumar, Gaurav Jain, Prashant J Nair, Ilya Soloveychik, and Purushotham Kamath. Keyformer: Kv cache reduction through key tokens selection for efficient generative inference. *Proceedings of Machine Learning and Systems*, 6:114–127, 2024.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439, 2020.

Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your vit but faster. In *ICLR*, 2023.

Yu-Neng Chuang, Tianwei Xing, Chia-Yuan Chang, Zirui Liu, Xun Chen, and Xia Hu. Learning to compress prompt in natural language formats, 2024. URL `https://arxiv.org/abs/2402.18700`.

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

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

Weizhi Fei, Xueyan Niu, Guoqing Xie, Yingqing Liu, Bo Bai, and Wei Han. Efficient prompt compression with evaluator heads for long-context transformer inference, 2025. URL `https://arxiv.org/abs/2501.12959`.

Kavita Ganesan. Rouge 2.0: Updated and improved measures for evaluation of summarization tasks, 2018. URL `https://arxiv.org/abs/1803.01937`.

Tao Ge, Hu Jing, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model. In *The Twelfth International Conference on Learning Representations*, 2024.

Saibo Geng, Nathan Ranchin, Yunzhen yao, Maxime Peyrard, Chris Wendler, Michael Gastpar, and Robert West. zip2zip: Inference-time adaptive tokenization via online compression, 2025. URL `https://arxiv.org/abs/2506.01084`.

Aaron Gokaslan and Vanya Cohen. Openwebtext corpus. `http://Skylion007.github.io/OpenWebTextCorpus`, 2019.

Andrew Gordon, Zornitsa Kozareva, and Melissa Roemmele. SemEval-2012 task 7: Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In Eneko Agirre, Johan Bos, Mona Diab, Suresh Manandhar, Yuval Marton, and Deniz Yuret, editors, **SEM 2012: The First Joint Conference on Lexical and Computational Semantics – Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*, pages 394–398, Montréal, Canada, 7-8 June 2012. Association for Computational Linguistics. URL `https://aclanthology.org/S12-1052/`.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James

Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models, 2024. URL `https://arxiv.org/abs/2407.21783`.

Jitai Hao, Yuke Zhu, Tian Wang, Jun Yu, Xin Xin, Bo Zheng, Zhaochun Ren, and Sheng Guo. Omnikv: Dynamic context selection for efficient long-context llms. In *The Thirteenth International Conference on Learning Representations*, 2025.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Llmlingua: Compressing prompts for accelerated inference of large language models. *arXiv preprint arXiv:2310.05736*, 2023.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1658–1677, 2024.

Julie Kallini, Shikhar Murty, Christopher D Manning, Christopher Potts, and Róbert Csordás. Mrt5: Dynamic token merging for efficient byte-level language models. *arXiv preprint arXiv:2410.20771*, 2024.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023.

Tian Lan, Deng Cai, Yan Wang, Heyan Huang, and Xian-Ling Mao. Copy is all you need. *arXiv preprint arXiv:2307.06962*, 2023.

Zixuan Lan, Yanhong Li, and Jiawei Zhou. Reduced matrix multiplication: Input-adaptive matrix-product reduction for llm inference, 2026. URL `https://arxiv.org/abs/2608.13426`.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR, 2023.

Yanhong Li, Zixuan Lan, and Jiawei Zhou. Text or pixels? evaluating efficiency and understanding of llms with visual text inputs. In *EMNLP (Findings)*, pages 10564–10578, 2025a.

Yanhong Li, Karen Livescu, and Jiawei Zhou. Chunk-distilled language modeling. In *The Thirteenth International Conference on Learning Representations*, 2025b.

Yucheng Li. Unlocking context constraints of llms: Enhancing context efficiency of llms with self-information-based content filtering. *arXiv preprint arXiv:2304.12102*, 2023.

Yucheng Li, Bo Dong, Chenghua Lin, and Frank Guerin. Compressing context to enhance inference efficiency of large language models. *arXiv preprint arXiv:2310.06201*, 2023.

Zongqian Li, Yixuan Su, and Nigel Collier. 500xcompressor: Generalized prompt compression for large language models, 2024. URL https://arxiv.org/abs/2408.03094.

Barys Liskavets, Maxim Ushakov, Shuvendu Roy, Mark Klibanov, Ali Etemad, and Shane Luke. Prompt compression with context-aware sentence encoding for fast and improved llm inference, 2024. URL https://arxiv.org/abs/2409.01227.

Alisa Liu, Jonathan Hayase, Valentin Hofmann, Sewoong Oh, Noah A Smith, and Yejin Choi. Superbpe: Space travel for language models. *arXiv preprint arXiv:2503.13423*, 2025.

Mathematical Association of America. American Invitational Mathematics Examination (AIME) 2024. https://maa.org/math-competitions/american-invitational-mathematics-examination-aime, February 2024. Accessed: 2025-05-16.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016. URL https://arxiv.org/abs/1609.07843.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering, 2018. URL https://arxiv.org/abs/1809.02789.

Jesse Mu, Xiang Li, and Noah Goodman. Learning to compress prompts with gist tokens. *Advances in Neural Information Processing Systems*, 36:19327–19352, 2023.

Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Çağlar Guılçehre, and Bing Xiang. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In Stefan Riezler and Yoav Goldberg, editors, *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, pages 280–290, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/K16-1028. URL https://aclanthology.org/K16-1028/.

Artidoro Pagnoni, Ram Pasunuru, Pedro Rodriguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason Weston, Luke Zettlemoyer, et al. Byte latent transformer: Patches scale better than tokens. *arXiv preprint arXiv:2412.09871*, 2024.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression, 2024. URL https://arxiv.org/abs/2403.12968.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Michael Ryoo, AJ Piergiovanni, Anurag Arnab, Mostafa Dehghani, and Anelia Angelova. Tokenlearner: Adaptive space-time tokenization for videos. *Advances in neural information processing systems*, 34:12786–12797, 2021.

Ninglu Shao, Shitao Xiao, Zheng Liu, and Peitian Zhang. Flexibly scaling large language models contexts through extensible tokenization. *arXiv preprint arXiv:2401.07793*, 2024.

Jaekang Shin, Myeonggu Kang, Yunki Han, Junyoung Park, and Lee-Sup Kim. Atom: Adaptive token merging for efficient acceleration of vision transformer. *IEEE Transactions on Computers*, 2025.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*, 2024.

Fangyuan Xu, Weijia Shi, and Eunsol Choi. Recomp: Improving retrieval-augmented lms with context compression and selective augmentation. In *The Twelfth International Conference on Learning Representations*, 2024.

Zhentao Xu, Fengyi Li, Albert Chen, and Xiaofeng Wang. Procut: Llm prompt compression via attribution estimation, 2025. URL `https://arxiv.org/abs/2508.02053`.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. Byt5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306, 2022.

Lili Yu, Dániel Simig, Colin Flaherty, Armen Aghajanyan, Luke Zettlemoyer, and Mike Lewis. Megabyte: Predicting million-byte sequences with multiscale transformers. *Advances in Neural Information Processing Systems*, 36:78808–78823, 2023.

Xin Yuan, Hongliang Fei, and Jinoo Baek. Efficient transformer adaptation with soft token merging. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3658–3668, 2024.

Ivan Zakazov, Berke Argin, Oussama Gabouj, Kamel Charaf, Alexander Sharipov, Alexi Semiz, Lorenzo Drudi, Nicolas Baldwin, and Robert West. Cmprsr: Abstractive token-level question-agnostic prompt compressor, 2026. URL `https://arxiv.org/abs/2511.12281`.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert, 2020. URL `https://arxiv.org/abs/1904.09675`.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27, 2015.

Technical Appendices

A Reference Plots **17**

B Empirical Study **17**

B.1 Empirical Summary 17

B.2 LLM Background 19

B.3 Early Attempt One: Embedding Similarity-Based Token Merging 20

B.4 Early Attempt 2: Training-Based Pairwise Token Merging 21

B.5 Foundation to our current Work 21

B.6 Information Preservation and Approximation Scope 23

B.7 Generalization of Context-Shared Surrogate Embeddings 24

B.8 UMIM as Parameter-Efficient Task Adaptation 25

B.9 Choice of n-gram Lengths 26

B.10 Conclusion 26

C Merge-Rule Mining Details **27**

D Module Architecture and Integration **28**

D.1 Attention-based Pooling Architecture 29

D.2 Difference from Standard Self-Attention 29

D.3 Parameter Count 30

D.4 Training-time Integration 30

D.5 Prompt-time Integration 30

D.6 Decoding-time Integration 30

E Predictive Distillation Loss and Alignment Metrics **31**

F Inference with UMIM: Prompt and Decoding-Time Merging **34**

G Task Adaptation with SFT and RL **38**

G.1 Task-Specific Merge-Rule Construction 38

G.2 Distillation-Based SFT 39

G.3 RL Adaptation 39

G.4 Adaptation and Ablation Settings 40

G.5 Task-Adaptation Experimental Configuration 40

G.6 Implementation Procedures 41

H Base Merge-Module Training Setup **44**

H.1 Training Corpora 44

H.2 Optimization and Distributed Training 44

H.3 Model-Specific Configurations 45

I Additional Experiments and Evaluation **45**
I.1 Perplexity Evaluation 45
I.2 Text Generation 47
I.2.1 Generation Setup 47
I.2.2 Qualitative Analysis 47
I.2.3 Quantitative Results 47
I.3 Downstream Evaluation Protocols 48
I.3.1 Multiple-Choice QA 48
I.3.2 CNN/DailyMail Summarization 49
I.3.3 Long-Form Mathematical Reasoning 49
I.3.4 Long-Context Evaluation 50
I.4 Specialized-Domain Evaluation: Code-Adapted UMIM 51

J Efficiency Analysis **52**
J.1 Empirical Efficiency Evaluation and Throughput Analysis 52

K Ablation Study **54**
K.1 Frequency Threshold and Statistical Support 54
K.2 Merge-Module Capacity 54
K.3 Alternative Merge-Rule Construction 55
K.4 Positional Encoding under Token-Span Merging 55

A Reference Plots

B Empirical Study

B.1 Empirical Summary

We will subsequently provide additional content to supplement the main text in the following sections for UMIM. The central question of this work is whether the sequential computation of multiple adjacent tokens in a frozen autoregressive language model can be approximated by a single learned input-level representation. This is different from token deletion, prompt rewriting, hidden-state pruning, or introducing a new tokenizer. UMIM keeps the pretrained language model unchanged: the tokenizer, vocabulary, model architecture, and backbone parameters are all fixed. Only a lightweight merge module is trained to map selected token spans into single surrogate embeddings.

The main empirical message of the paper is that input-level token-span merging can substantially reduce the effective sequence length while preserving the behavior of the original model to a large extent. The merge module operates before Transformer computation, using only static token embeddings as input. After training, the resulting surrogate embedding can replace a short multi-token span as a single computational unit. This allows the model to process compressed prompts and compressed decoding contexts without changing the underlying language model. In this sense, UMIM is not merely a heuristic for shortening text; it is a learned input-level approximation of multi-step token computation in frozen Transformer language models.

A key strength of UMIM is that the compression is achieved with strong performance preservation. Across language modeling and downstream tasks, the compressed model remains close to the uncompressed base model while reducing the effective sequence length. This is important because many efficient inference methods obtain speed or memory savings by deleting tokens, dropping

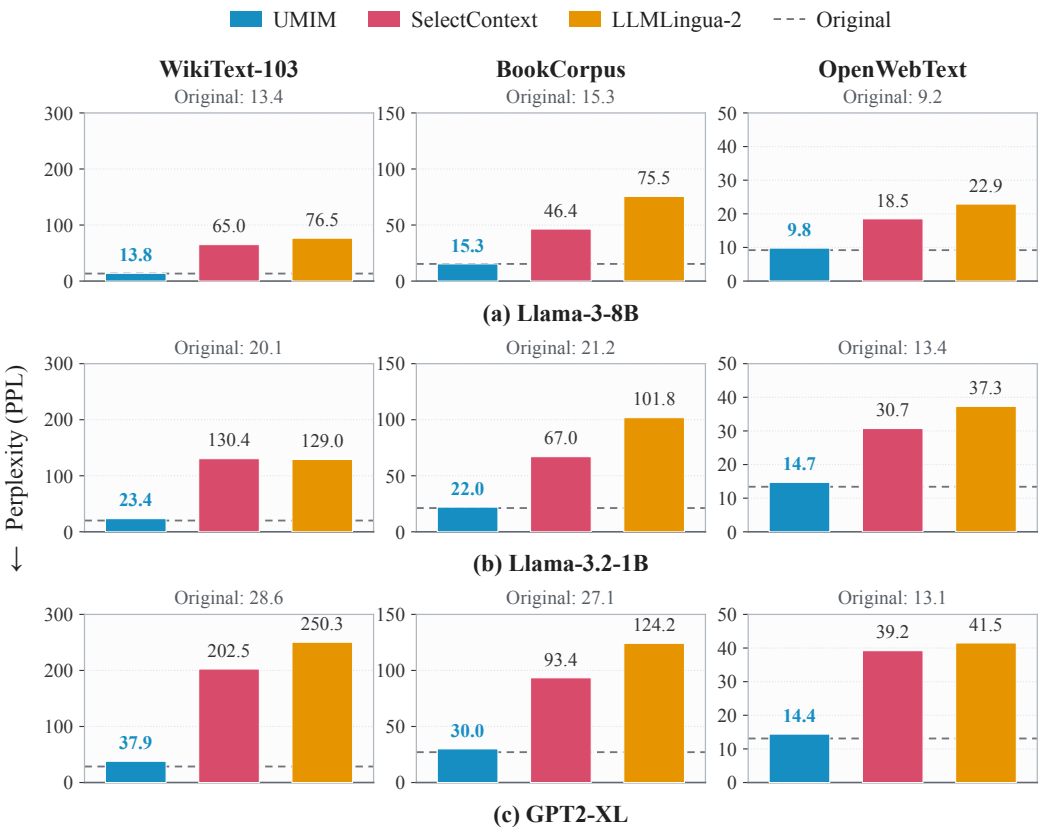


Figure 5: **Language modeling performance under token reduction.** Visualization of the results in Table 2. Rows correspond to backbones and columns to datasets. Bars show perplexity (PPL; lower is better), with values annotated above. Dashed horizontal lines indicate the corresponding original-model PPL, also reported above each panel. All bars start at zero. Y-axis scales are shared within each dataset column but differ across columns.

context, or compressing information in a task-dependent way, often causing a visible accuracy degradation. UMIM instead attempts to preserve the functional role of the merged span: the span is still represented, but its multi-step computation is replaced by a single learned surrogate embedding. This design explains why UMIM can reduce sequence length and KV-cache usage while maintaining strong predictive and downstream performance.

Another important finding is that the merge module trained on a general corpus can transfer directly to downstream tasks. In the task-agnostic setting, the UMIM module trained on WikiText-103 is applied to unseen tasks without updating the backbone language model and without task-specific supervision. Even under this direct-transfer setting, UMIM achieves a strong accuracy–compression trade-off and consistently outperforms representative prompt-compression and context-reduction baselines at comparable token reduction rates. This result suggests that the learned module is not simply memorizing a fixed training corpus; rather, it learns a reusable span-to-surrogate mapping that can be consumed by frozen language models across different tasks.

The distinction between merge rules and the merge module is central to the method. Frequency-based rules decide where merging is attempted, but they do not define the compressed representation itself. The compressed representation is produced by the learned merge module. Therefore, UMIM should not be understood as only an n-gram compression heuristic. The rules provide stable and recurring candidate spans, while predictive distillation teaches the module how to replace the original multi-token computation with a single embedding that preserves the language model's output behavior. This separation makes the framework both reusable and adaptable: the pretrained module can be

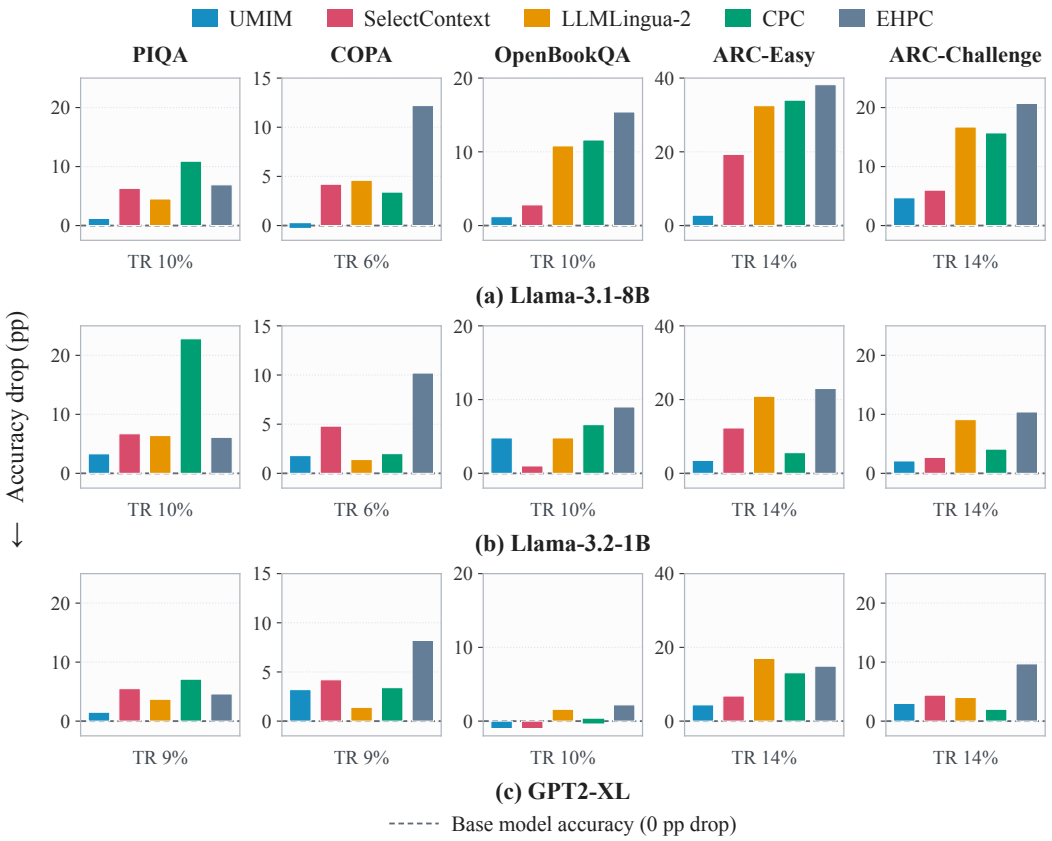


Figure 6: **QA accuracy drops under token reduction.** Visualization of the per-task results in Table 5. Rows correspond to backbones and columns to benchmarks. Each bar shows the original-model accuracy minus the compressed-model accuracy, measured in percentage points; lower is better. Zero denotes unchanged accuracy, and negative values indicate accuracy gains. Short colored marks represent exactly zero drops. All compression methods within each panel use the indicated TR. Vertical-axis ranges vary across benchmarks but are shared across backbones within each benchmark.

reused across domains, while the rule set can be reconstructed when a target task requires higher merge coverage.

The task-specific adaptation experiments further demonstrate this point. Starting from the WikiText-trained merge module, we can rebuild task-specific merge rules and then adapt only the merge module through lightweight fine-tuning and preference optimization. The backbone language model remains frozen throughout this process. This setting is substantially more practical than retraining or modifying the full language model. More importantly, the adapted module can achieve a stronger accuracy–compression trade-off than both the directly transferred module and external compression baselines. In some settings, the adapted UMIM variant can preserve or even exceed the accuracy of the original uncompressed base model while still reducing the effective sequence length by a large margin. This indicates that UMIM is not only a compression method, but also a practical framework for learning reusable and adaptable input-level approximations of sequential computation.

B.2 LLM Background

Modern language models generate text in an autoregressive manner, where tokens are processed and generated sequentially. This sequential process leads to high inference cost, especially when the input or generated sequence becomes long. Motivated by this practical limitation, we explore whether multiple tokens can be represented by a single learned embedding during inference. The goal is to

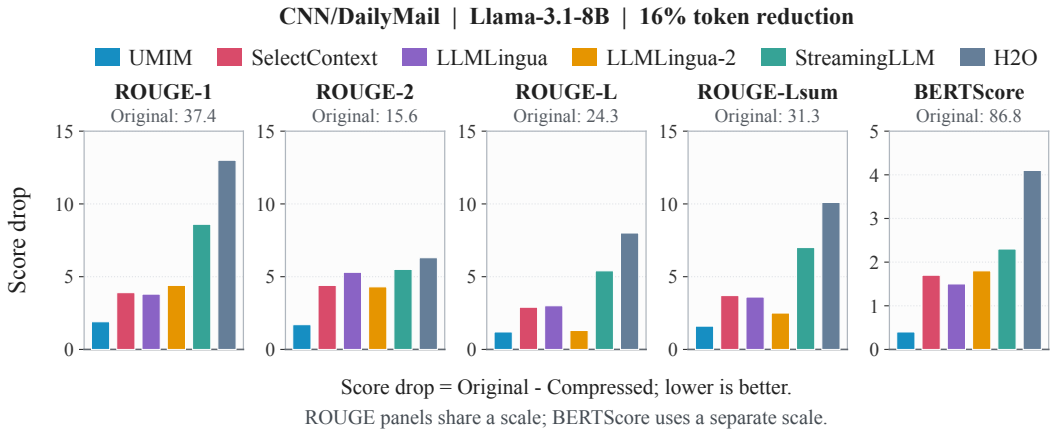


Figure 7: **Summarization score drops under token reduction.** Visualization of the results in Table 6 on CNN/DailyMail with Llama-3.1-8B. All compression methods use 16% token reduction. Each bar shows the original-model score minus the compressed-model score; lower is better. Original-model scores are listed above the panels. The four ROUGE panels share the same vertical scale, while BERTScore uses a separate scale.

reduce token-level computation while preserving the model's original predictive behavior as much as possible.

In the following subsections, we describe our early attempts and empirical observations, which may provide valuable experience.

B.3 Early Attempt One: Embedding Similarity-Based Token Merging

In the early stages of this work, we explored token merging from a purely representation-based and linguistically motivated perspective. Since our goal was to reduce redundant token-level computation without introducing additional training or modifying the language model, a natural first attempt was to identify semantically similar tokens and merge them directly at the embedding level.

Concretely, given a token sequence (x_1, x_2, x_3, x_4), we considered computing the cosine similarity between adjacent token embeddings, such as $e(x_2)$ and $e(x_3)$. If the similarity exceeded a predefined threshold, the two tokens were regarded as semantically redundant and merged into a single representation, typically by simple averaging of their embeddings. This approach was motivated by the intuition that semantically similar tokens may play interchangeable roles in language understanding, and that their embeddings—learned by large language models—should reflect this similarity.

Importantly, this design deliberately avoided any form of training or supervision. The merging decision relied solely on static token embeddings and geometric similarity measures, making it lightweight and model-agnostic. From a linguistic and representation-learning standpoint, this approach appears reasonable and aligns with common practices in embedding-based clustering and semantic similarity analysis.

However, empirical results quickly revealed that this strategy failed to preserve model behavior in practice. Even when merging token pairs with relatively high cosine similarity, the resulting generations often deviated noticeably from the original outputs. In some cases, merging led to abrupt changes in the predicted next-token distribution, indicating that semantic proximity in embedding space was insufficient to guarantee computational substitutability within the language model.

Upon further analysis, we identified several fundamental limitations of this approach. First, cosine similarity in high-dimensional embedding spaces lacks a stable and transferable threshold. The similarity distribution is highly concentrated, making it difficult to distinguish genuinely mergeable token pairs from unrelated ones in a robust manner. Small variations in threshold choice often resulted in either excessive merging or negligible compression.

More fundamentally, semantic similarity between token embeddings does not imply equivalence in their computational roles within a Transformer. Token embeddings serve only as inputs to a highly nonlinear and context-dependent computation pipeline involving positional encodings, multi-head attention, and deep feed-forward transformations. Two tokens that appear similar in embedding space may interact very differently with surrounding context, attend to different positions, or trigger distinct internal computation patterns. As a result, replacing multiple tokens with an averaged embedding introduces a distribution shift that the model was never trained to handle.

B.4 Early Attempt 2: Training-Based Pairwise Token Merging

After observing that purely heuristic or geometry-based token merging strategies fail to reliably preserve model behavior, we naturally shifted to a training-based perspective. From this viewpoint, it is reasonable to hypothesize that there exists a learnable *super embedding* in the embedding space that can replace multiple token embeddings and approximate their joint computational effect. The theoretical motivation behind this assumption will be discussed in later sections.

In this stage, we began with the simplest and most controlled setting: *pairwise token merging*. Given an input sequence (x_1, x_2, x_3, x_4), we deterministically merged (x_1, x_2) into a single token and (x_3, x_4) into another, thereby reducing the sequence length by half while keeping the merging rule fixed. This design avoids the complexity of dynamic span selection and allows us to isolate the learnability of the merging operation itself.

As an initial sanity check, we froze a GPT-2 model and trained only a lightweight merging module on a small number of sampled sentences. Even under this minimal setup, we observed that the merged representations could approximate the original next-token predictive distributions after a few training steps. This result indicates that, in principle, such super embeddings do exist and can be learned under constrained conditions.

We then extended this experiment to a more realistic setting using the IMDB dataset. In this configuration, input sequences of length 512 were uniformly compressed via pairwise merging into 256 tokens. At this scale, the problem shifted from feasibility to stability and generalization. We found that the expressive capacity of the merging module became a critical factor. To achieve satisfactory distribution alignment with the frozen GPT-2 model, the merging module required substantially increased capacity. Consequently, we employed a deep, over-parameterized linear network with nonlinear activations, LayerNorm, and residual connections to transform the merged embeddings back to the original embedding dimension. While this design lacks theoretical elegance, it represented the most effective empirical solution under the constraints of this experimental setup.

Regarding the training objective, we used distribution-level alignment metrics consistent with those in the main experiments (e.g., Top-1 and Top-p overlap). A notable empirical observation was that using KL divergence alone for distillation often resulted in unstable training or poor convergence. In this setting, the merging module was required to directly approximate the predictive distribution induced by two consecutive decoding steps, leading to noisy and weak optimization signals. To mitigate this issue, we introduced an additional cosine similarity constraint at the embedding or sentence level, providing a geometric regularization that significantly improved training stability.

It is important to emphasize that the design choices in this stage—including fixed pairwise merging, a highly over-parameterized merging network, and the combination of KL-based distillation with cosine regularization—were driven primarily by empirical considerations rather than theoretical optimality. Although this approach can yield reasonable results in controlled and moderate-scale settings, its reliance on fixed segmentation and excessive parameterization exposes clear limitations in scalability and generalization. These observations motivated the search for a more general and structurally grounded merging formulation, which we present in the subsequent sections.

B.5 Foundation to our current Work

After extensive exploration and empirical refinement, we arrive at the current merging formulation. The concrete architectural and implementation details have already been described thoroughly in the main text and appendices. In this section, we focus instead on a more fundamental question: *why this approach is expected to work in principle*. Our goal is to clarify that the method is not a heuristic construction, but rather the result of parallel theoretical reasoning and engineering validation.

We begin with the construction of the merge set. Any token merging strategy must first address a basic question: *which token pairs or spans should be merged*. This question is closely tied to the tokenization mechanisms employed by modern language models.

Most contemporary language models rely on Byte Pair Encoding (BPE) or its variants for tokenization. BPE is not designed to recover linguistically complete units; instead, it is a statistically driven compression algorithm. Starting from atomic units (characters or bytes), BPE repeatedly merges the most frequent adjacent token pairs in the training corpus, aiming to reduce sequence length while maintaining a compact vocabulary. As a result, the segmentation produced by BPE is governed by frequency statistics rather than semantic or linguistic completeness.

While highly effective in practice, this process often leads to the fragmentation of expressions that humans naturally perceive as single semantic units. For example, adverbs such as *excitedly* may be decomposed into *excited* and *ly*, and common multi-word expressions such as *machine learning* or *love to do something* are represented as multiple tokens. From a linguistic or cognitive perspective, these expressions typically function as indivisible units, yet they are split due to the statistical nature of the tokenizer.

Our approach does not seek to revise or replace BPE, but rather to complement it at inference time. Specifically, we ask whether frequently occurring composite expressions that are split by the tokenizer can be re-aggregated into larger units during model execution. This motivates the use of high-frequency n-grams as candidates for merging. Such a choice can be interpreted both as compensating for the side effects of BPE segmentation and as explicitly modeling structures that the language model has already encountered repeatedly during training.

A functional approximation view. The existence of a useful surrogate embedding can be understood through the same approximation principle that underlies ordinary model training. A language model does not have direct access to the true data-generating distribution. Instead, its parameters are learned from empirical observations:

$$\theta^* = \arg\min_{\theta} \mathbb{E}_{(x,y)\sim p_{\text{data}}} \left[-\log p_\theta(y \mid x) \right]. \tag{3}$$

Once pretrained, the frozen model p_θ defines a stable and observable mapping from an input sequence to a sequence of predictive distributions. UMIM introduces a second approximation problem. Rather than learning p_θ from data, it keeps p_θ fixed and learns a merge module M_ϕ whose compressed inputs reproduce the behavior of p_θ on the original sequence.

Let $\widetilde{x} = C_\phi(x; \mathcal{R})$ denote the sequence obtained by replacing spans selected by the merge rules $\mathcal{R}$ with surrogate embeddings

$$z_s = M_\phi(e(s)). \tag{4}$$

If $\mathcal{A}(x)$ denotes the alignment between retained positions in the original and compressed sequences, the merge module is trained according to

$$\phi^* = \arg\min_{\phi} \mathbb{E}_{x\sim\mathcal{D}} \left[\sum_{(t,\widetilde{t})\in\mathcal{A}(x)} D_{\text{KL}} \left(p_\theta(\cdot \mid x_{\le t}) \,\middle\|\, p_\theta(\cdot \mid \widetilde{x}_{\le \widetilde{t}}) \right) \right]. \tag{5}$$

The underlying hypothesis is therefore not that a single embedding exactly reconstructs every intermediate state produced by a multi-token span. Rather, we hypothesize that, over the contexts in which a recurring span appears, there exists a compact input representation that approximately preserves the span's functional effect on the frozen model's subsequent predictive distributions. UMIM learns this representation directly from the behavior of the uncompressed model.

High-frequency spans are particularly suitable for this approximation because they provide repeated observations across many contexts. For a span s, its surrogate representation is effectively optimized over the empirical context distribution in which s occurs. Frequent spans provide more samples from this distribution, making the expected behavioral effect of replacing s better specified and more reliably estimable. The shared merge module further amortizes this approximation across different spans, rather than learning an independent embedding for every entry in the rule set.

From this perspective, UMIM performs functional distillation at the input level. It does not require the surrogate embedding to reproduce the original token embeddings or all of their intermediate hidden states. It only requires the compressed sequence to preserve the predictive behavior that matters to the frozen language model.

B.6 Information Preservation and Approximation Scope

UMIM does not claim that a surrogate embedding can losslessly replace the original token span in every context. Nor does it require every hidden state, attention score, or KV state to remain identical before and after compression. We instead adopt an operational definition of information preservation: with the backbone language model frozen, the compressed sequence should preserve the predictive behavior of the original sequence at subsequent positions as closely as possible. Thus, UMIM targets end-to-end predictive behavior rather than exact layer-wise equivalence.

UMIM also differs from direct token deletion. For each merged span, the embeddings of all constituent tokens are provided to the merge module and jointly used to produce the surrogate embedding. Their information is therefore not discarded without processing, but compressed through a trainable mapping. During training, we compare the predictive distributions of the original and compressed sequences across many naturally occurring contexts. Distillation then teaches the surrogate to approximate the aggregate effect of the complete span on subsequent predictions. High-frequency n-grams are selected primarily because they provide more training observations across contexts, not because frequent spans are assumed to be unimportant or semantically empty.

Although the same n-gram produces the same static surrogate embedding at the input level, its role inside the model is not fully context-independent. Once the surrogate is passed into the frozen Transformer, it interacts with surrounding tokens through positional encoding, self-attention, MLP layers, and subsequent Transformer computation. Different contexts can therefore produce different hidden states, attention interactions, and downstream effects from the same initial surrogate. Context independence applies to the initial output of the merge module, not to the complete contextual computation performed by the Transformer.

Our experiments examine this preservation at several levels. The distribution-alignment metrics directly measure agreement between the original and compressed predictive distributions at aligned positions. Table 3 provides qualitative examples in which generation behavior is preserved after merging, while Figure 4 shows that surrogate states remain involved in subsequent attention rather than being ignored. The PPL, QA, and summarization experiments further evaluate preservation through downstream behavior. The task-adaptation results also show that when a distribution shift exists between the general-purpose merge module and a target task, applying SFT and RL only to the small merge module can further reduce the performance loss caused by merging while keeping the backbone LLM frozen.

At the same time, we acknowledge a structural limit of this approximation. After multiple token-level KV states are compressed into one surrogate state, a future query can no longer access every constituent token independently in exactly the same way as in the original model. We therefore do not claim that a context-independent surrogate preserves every internal computation or task-critical detail in every context. Highly ambiguous, strongly context-dependent, or task-critical spans may introduce greater approximation error. This helps explain the remaining performance gap in some task-agnostic evaluations and why the approximation can become more difficult under extreme length and domain shifts.

Whether a token is critical is itself dependent on the context, query, and downstream task, and frequency-based rules are not semantic-importance detectors. Our claim is therefore not lossless semantic compression, but an effective functional approximation over the observed data distribution. For settings requiring more conservative treatment of critical content, the framework can use stricter merge rules, a higher frequency threshold, task-defined protected spans, or task-specific rules. The merge module can also be further adapted through SFT and RL. These choices provide different ways to control the accuracy–compression trade-off.

In summary, UMIM does not guarantee lossless semantic replacement for every span and context. It processes all constituent token embeddings and distills their aggregate predictive effect under the frozen LLM, thereby providing practical information compression with substantial behavioral preservation. Our results show that this approximation is effective across the evaluated models and

tasks, while the remaining errors define the current scope of the method and motivate conservative rule construction and task adaptation.

B.7 Generalization of Context-Shared Surrogate Embeddings

The preceding discussion defines what UMIM aims to preserve and clarifies the scope of this approximation, but leaves open an important question: if a surrogate is optimized on a finite set of observed contexts, why should it preserve the frozen model's predictive behavior in previously unseen contexts? We provide a conditional generalization analysis that relates this discrepancy to the frequency support used during rule construction.

Let g be a candidate n-gram and let c denote a context in which g occurs. The frozen teacher model produces the predictive distribution $p_\theta(\cdot \mid c, g)$ from the original sequence. Replacing g with $M_\phi(e(g))$ produces the corresponding compressed-model distribution $p_{\theta,\phi}(\cdot \mid c, g)$. We define the behavioral approximation loss as

$$\ell_\phi(c, g) = D_{\mathrm{KL}}\left(p_\theta(\cdot \mid c, g) \parallel p_{\theta,\phi}(\cdot \mid c, g)\right). \tag{6}$$

When distillation is applied at multiple aligned positions, $\ell_\phi(c, g)$ denotes the average KL divergence over those positions. The soft-target cross-entropy used during training satisfies

$$\mathrm{CE}\left(p_\theta, p_{\theta,\phi}\right) = H(p_\theta) + D_{\mathrm{KL}}\left(p_\theta \| p_{\theta,\phi}\right). \tag{7}$$

Because the teacher and its entropy are fixed with respect to ϕ, minimizing the soft-target cross-entropy is equivalent to minimizing the forward KL divergence.

For a fixed g, suppose that its N_g observed training contexts are $c_1, \ldots, c_{N_g}$. Its empirical and population risks are

$$\widehat{R}_g(\phi) = \frac{1}{N_g} \sum_{i=1}^{N_g} \ell_\phi(c_i, g), \qquad R_g(\phi) = \mathbb{E}_{c \sim P(c|g)}\left[\ell_\phi(c, g)\right]. \tag{8}$$

We make the following standard assumptions for the analysis: for each fixed g, the observed contexts are modeled as independent draws from $P(c \mid g)$; the loss satisfies $0 \le \ell_\phi(c, g) \le B$; and the loss class induced by the capacity-controlled merge-module family Φ satisfies

$$\widehat{\mathfrak{R}}_{N_g}\left(\mathcal{L}_{\Phi,g}\right) \le \frac{C_\Phi}{\sqrt{N_g}}. \tag{9}$$

The bounded-loss condition can be implemented through probability clipping, or viewed as an explicit bounded-logit assumption over the distributions included in the analysis. Let U be a fixed finite universe of candidate n-grams defined before observing the training corpus, and let

$$G_\tau = \{g \in U : N_g \ge \tau\} \tag{10}$$

be the set retained by a frequency threshold τ.

A standard Rademacher-complexity argument, followed by a union bound over U, implies that, with probability at least $1 - \delta$, simultaneously for every $g \in G_\tau$ and every $\phi \in \Phi$,

$$R_g(\phi) \le \widehat{R}_g(\phi) + 2\widehat{\mathfrak{R}}_{N_g}\left(\mathcal{L}_{\Phi,g}\right) + 3B\sqrt{\frac{\log(2|U|/\delta)}{2N_g}}. \tag{11}$$

Using the assumed capacity bound and $N_g \ge \tau$ gives

$$R_g(\phi) \le \widehat{R}_g(\phi) + \frac{2C_\Phi + 3B\sqrt{\log(2|U|/\delta)/2}}{\sqrt{\tau}}. \tag{12}$$

Equation 12 separates two sources of error. The empirical term $\widehat{R}_g(\phi)$ contains the approximation error incurred when a multi-token computation is represented by a single surrogate. This term may remain nonzero even with unlimited observations. The second term captures statistical uncertainty from learning the surrogate using finitely many contexts and decreases as the minimum frequency support τ increases.

Pinsker's inequality additionally gives

$$\mathrm{TV}\left(p_\theta, p_{\theta,\phi}\right) \leq \sqrt{\frac{1}{2} D_{\mathrm{KL}}\left(p_\theta \| p_{\theta,\phi}\right)}. \tag{13}$$

Together with Jensen's inequality, this implies

$$\mathbb{E}_c\left[\mathrm{TV}\left(p_\theta, p_{\theta,\phi}\right)\right] \leq \sqrt{\frac{R_g(\phi)}{2}}. \tag{14}$$

The result is deliberately conditional. It does not establish that one context-independent surrogate exactly reproduces every state associated with a multi-token span, nor does it guarantee small approximation error for an arbitrary span. Instead, it shows that when the merge module fits the observed teacher behavior well, stronger frequency support provides tighter control of its expected discrepancy on unseen contexts drawn from the same distribution. The frequency-threshold ablation in Section K empirically evaluates the qualitative prediction of this analysis.

B.8 UMIM as Parameter-Efficient Task Adaptation

The preceding analysis considers generalization to unseen contexts drawn from a similar distribution. In downstream tasks, however, the context distribution, frequent token spans, and task objective may differ from those observed in WikiText-103. UMIM addresses this distribution shift through lightweight task adaptation. The WikiText-trained base merge module serves as a reusable initialization that has already learned a general span-to-surrogate mapping. Given a target task, we first reconstruct task-specific merge rules from its training split to increase merge coverage. If no additional training is desired, the new rules can be directly used with the fixed base merge module, corresponding to the `WT + Rules` setting.

For further adaptation, `WT + FT` initializes from the WikiText-trained base merge module and performs distillation-based SFT on the target-task training corpus. This is not conventional label-only fine-tuning. The frozen backbone processes the original uncompressed sequence to provide the teacher distribution and processes the compressed sequence to produce the student distribution. Their predictions are aligned at valid positions, and only the merge module is updated to minimize the distribution-matching loss. The `Task-Adapted` setting then starts from this SFT checkpoint and further applies RL to the merge module using the downstream task signal. The backbone LLM remains frozen throughout all stages.

This formulation makes UMIM a parameter-efficient task-adaptation method in addition to a token-compression method. Similar to parameter-efficient fine-tuning methods such as LoRA, UMIM updates only a small set of additional parameters while leaving the pretrained backbone unchanged. However, rather than inserting trainable parameters into the internal Transformer layers, UMIM adapts an external input-level merge module. It therefore provides two forms of efficiency simultaneously: only a small module is trained, and the adapted model can process a shorter effective sequence with reduced KV-cache usage. Task adaptation can consequently improve the accuracy–compression trade-off in both directions: task-specific rules increase the achievable merge ratio, while SFT and RL help preserve or improve task performance under this more aggressive compression.

The base merge module is important to this process. The `Task-trained` ablation in Figure 3 trains a new merge module directly from the downstream corpus, whereas `WT + FT` and `Task-Adapted` reuse the WikiText-trained initialization. Their stronger performance indicates that the base module provides reusable compression knowledge that cannot always be learned reliably from a smaller task-specific corpus alone. We do not claim that adaptation must outperform the uncompressed model on every task. Rather, Figure 3 and Appendix Figure 5 show that, across the three evaluated tasks, adapting only the merge module can support substantially higher merge ratios while preserving and, in some settings, improving downstream accuracy.

B.9 Choice of n-gram Lengths

This subsection clarifies why the current implementation of UMIM restricts candidate spans to n-grams of length 2–4. This range reflects a practical choice based on statistical support, training feasibility, and the timing of compression during autoregressive decoding. It is not a fundamental limitation of the merge-module architecture.

Our base merge module is trained on WikiText-103, from which we extract high-frequency contiguous token spans. For a fixed corpus, the number of possible n-grams grows rapidly with n, while the number of repeated occurrences of any particular span generally decreases. Longer spans therefore tend to provide fewer contextual observations for learning a reliable surrogate. Restricting the rule set to 2–4 grams allows each retained span to satisfy a meaningful minimum-frequency threshold while keeping the number of candidate rules manageable.

The current implementation also uses a single-stage, non-overlapping merge assignment. Overlap is resolved during rule construction, so the merge rules applied during training and inference do not compete for the same token positions. For example, given a sequence (x_1, x_2, x_3, x_4), if (x_2, x_3) is selected and replaced by a surrogate x_{23}, the resulting surrogate is not recursively merged with x_4. Representing (x_2, x_3, x_4) with one surrogate instead requires the complete 3-gram to be selected and merged in a single operation.

Recursive or hierarchical merging is conceptually possible, but it would define a different learning problem. The representation of a span would depend on the order in which its sub-spans were merged, producing dynamic and path-dependent merge structures. Supporting these structures would complicate rule assignment, teacher–student alignment, and efficient batch construction. We therefore use a fixed non-overlapping assignment so that every compressed sequence and every distillation target can be determined before the forward pass.

Shorter spans also provide favorable statistical coverage. A longer recurring expression may contain shorter sub-spans that already satisfy the merge threshold. Consequently, parts of the same region may already be compressed through shorter, non-overlapping matches without explicitly retaining the complete longer expression as an additional rule. Extending the rule set to higher-order n-grams may therefore provide diminishing additional coverage, while increasing the number of candidates that must be counted, stored, and observed sufficiently often during training. We treat this as a statistical and computational prior rather than as a claim that all longer expressions can be represented equivalently by shorter merges.

The timing of a match is also important during autoregressive decoding. A span can be merged only after the token IDs of the complete span have been generated. A bigram can therefore be detected immediately after its second token becomes available. Once merged, it reduces the effective sequence length and KV-cache length used by subsequent decoding steps. A 5-gram, by contrast, cannot be matched until all five token IDs have been generated. Before that point, some parts of the same region may already have become eligible for shorter, non-overlapping matches. Longer rules can thus delay the point at which compression begins, without necessarily providing a proportional increase in token reduction.

The merge module itself can process variable-length spans and does not conceptually require $n \leq 4$. The framework could therefore be extended to longer n-grams, or to recursive merging with an appropriately redesigned training procedure. In the present work, however, 2–4 grams provide a balanced trade-off among contextual support, rule-set size, batch-level training, non-overlapping assignment, and early compression during decoding. We therefore adopt this range as the default operating regime of UMIM, rather than as a theoretical upper bound on mergeable span length.

B.10 Conclusion

Having presented the methodology, empirical development, and theoretical analysis of UMIM, we conclude by clarifying the broader perspective of this work. Our goal is not to claim a single definitive strategy for token merging, but to introduce and validate a different way of thinking about token-level computation in frozen language models. Our experiments show that a learned input-level surrogate can replace selected multi-token spans while approximately preserving the predictive behavior of the original model. This provides empirical evidence that part of the sequential computation associated with recurring token spans can be compressed into a single learned representation.

The merge module and frequency-based rule construction used in this paper represent one practical and empirically effective implementation of this idea, rather than its only possible realization. The rules identify spans with sufficient statistical support, while predictive distillation trains the merge module to approximate their functional effect on the frozen language model. Our theoretical discussion further clarifies why stronger frequency support can improve generalization across contexts, while also recognizing that a context-independent surrogate cannot guarantee exact equivalence for every span and context.

Beyond task-agnostic compression, UMIM also provides a parameter-efficient approach to task adaptation. A base merge module trained on a general corpus can be reused directly or adapted to a downstream task by reconstructing task-specific merge rules, continuing distillation-based SFT, and optionally applying RL. Throughout this process, the backbone LLM remains frozen and only the lightweight merge module is updated. The resulting adaptation can support higher merge ratios while preserving or, in some settings, improving downstream accuracy. Thus, UMIM provides efficiency in both training and inference: it requires updating only a small additional module while reducing the effective sequence length and KV-cache usage during inference.

Our current exploration remains limited by available training data and computational resources. Future work may investigate stronger merge criteria, context-dependent or hierarchical surrogate representations, long-context training, and more optimized inference implementations. Another promising direction is to incorporate token merging directly into language-model pretraining or to jointly optimize the merge mechanism and backbone, rather than treating merging only as a post-hoc component.

Overall, the contribution of this work is not limited to a particular merge module. It introduces a broader research direction in which token-level sequential computation is treated as a structure that can be learned, approximated, adapted, and compressed. We hope this perspective encourages further work on input-level computation reduction and parameter-efficient adaptation for more efficient and scalable language models.

C Merge-Rule Mining Details

Given a corpus $\mathcal{C}$, we tokenize its documents using the tokenizer of the corresponding backbone model. The tokenized documents are concatenated and partitioned into non-overlapping segments for batched data processing. We use a segment length of $L = 512$ for Llama-3.1-8B, Llama-3.2-1B, and GPT-2-XL, and $L = 1024$ for DeepScaleR-1.5B-Preview. The following procedure constructs the merge rules used for training the base merge module.

N-gram Extraction and Frequency Counting Let $S = \{s_1, \ldots, s_N\}$ denote the resulting tokenized segments, where $s_i = (t_{i,1}, \ldots, t_{i,L})$. For each segment, we enumerate all contiguous n-grams with $n \in \{2, 3, 4\}$. For an n-gram

$$g = (t_{i,j}, t_{i,j+1}, \ldots, t_{i,j+n-1}),$$

we compute its corpus-wide occurrence count

$$f(g) = \sum_{i=1}^{N} \sum_{j=1}^{L-n+1} \mathbb{1}[(t_{i,j}, \ldots, t_{i,j+n-1}) = g] \,.$$

The frequency is aggregated across all segments in the training corpus.

Frequency-based Candidate Selection We retain an n-gram only if it occurs at least τ times:

$$\mathcal{R}_{\tau}^{(n)} = \{g : |g| = n,\ f(g) \geq \tau\} \,, \qquad n \in \{2, 3, 4\}.$$

Unless otherwise stated, the base merge rules use $\tau = 5$. We vary this threshold in the frequency-threshold ablation to study the trade-off between merge coverage and the statistical support available for learning surrogate embeddings.

Intra-order Overlap Filtering Frequency thresholding may retain same-order rules that compete for overlapping positions. For example, two bigrams (a, b) and (b, c) can occur at consecutive

Table 7: Number of retained merge rules used for base merge-module training. All rule sets use the frequency threshold $\tau = 5$.

Model	2-grams	3-grams	4-grams
Llama-3.1-8B	216,700	684,793	1,266,080
Llama-3.2-1B	216,700	684,793	1,266,080
GPT-2-XL	218,415	702,570	1,241,657
DeepScaleR-1.5B-Preview	35,691	212,367	1,665,423

positions and share token b. Analogously, two n-grams may overlap when the $(n-1)$-token suffix of one candidate equals the $(n-1)$-token prefix of another. For such competing candidates, we retain the rule with the larger corpus frequency. This filtering reduces systematic conflicts among rules of the same order while favoring candidates with stronger training support.

Cross-order Containment Filtering We next remove lower-order rules that are fully contained in retained higher-order rules. Specifically, a trigram is removed if it is a contiguous sub-span of a retained 4-gram, and a bigram is removed if it is a contiguous sub-span of a retained 3-gram or 4-gram. This gives priority to longer and more specific spans when the same local token pattern is represented at multiple orders. The resulting rule set is

$$\mathcal{R} = \widetilde{\mathcal{R}}_\tau^{(2)} \cup \widetilde{\mathcal{R}}_\tau^{(3)} \cup \widetilde{\mathcal{R}}_\tau^{(4)},$$

where $\widetilde{\mathcal{R}}_\tau^{(n)}$ denotes the filtered rules of order n.

Non-overlapping Assignment in a Sequence After constructing $\mathcal{R}$, we match its rules against each token sequence. All matched occurrences are sorted first by their starting position and then, for matches with the same starting position, by decreasing span length. We scan the sorted candidates from left to right and retain a match only if it does not overlap with a previously selected match. Therefore, every token belongs to at most one merged span, and all merge locations can be determined before the forward pass. The current implementation performs a single stage of merging and does not recursively merge previously constructed surrogate embeddings.

Task-specific Rule Adaptation The main task-agnostic experiments use the rules mined from WikiText-103 together with the WikiText-trained base merge module. No downstream-task data are used to reconstruct rules in this setting. In the task-adaptation experiments, we apply the same frequency-counting and non-overlapping assignment procedure to the training split of the target task. Reconstructing only the rule set corresponds to the `WT + Rules` setting, while `WT + FT` and `Task-Adapted` additionally adapt the WikiText-trained merge module through distillation-based SFT and RL, respectively.

Choice of 2- to 4-grams The restriction to 2- to 4-grams is a practical design choice rather than an architectural limitation. It reflects the statistical support of recurring spans, the additional coverage provided by longer rules, and the delay before a complete span becomes available during autoregressive decoding. A detailed discussion is provided in Section B.9.

Number of Retained Merge Rules Table 7 reports the final rule sets used to train the base merge modules. The rules for Llama-3.1-8B, Llama-3.2-1B, and GPT-2-XL are mined from WikiText-103. Llama-3.1-8B and Llama-3.2-1B use the same tokenizer and therefore share the same rule set. For DeepScaleR-1.5B-Preview, the rules are mined from our generated mathematical-reasoning corpus described in Section H.1.

D Module Architecture and Integration

The UMIM merge module is an attention-based pooling network that operates on static input embeddings before they enter the frozen Transformer. Given a selected span of n token embeddings, the module produces one surrogate embedding with the same dimension as the backbone's input embeddings. The current experiments use $n \in \{2, 3, 4\}$, although the pooling architecture itself can process variable-length spans.

The merge module is trained separately for each backbone in the current work. Its architecture can be applied to different Transformer-based language models, but its parameters depend on the backbone's embedding dimension, embedding geometry, tokenizer, and predictive behavior. We therefore do not assume that one merge-module checkpoint transfers directly across different backbone models.

D.1 Attention-based Pooling Architecture

Let

$$E_s = [e(x_t), e(x_{t+1}), \ldots, e(x_{t+n-1})]^\intercal \in \mathbb{R}^{n \times d}$$

denote the static input embeddings of a selected token span $s = (x_t, \ldots, x_{t+n-1})$, where d is the embedding dimension of the frozen language model. Let h be the number of pooling heads and $d_h = d/h$.

The span embeddings are first processed by a shared token projection:

$$P = \mathrm{LN}_{kv}\left(\mathrm{GELU}\left(\mathrm{Linear}_{kv}(E_s)\right)\right) \in \mathbb{R}^{n \times d}, \tag{15}$$

where $\mathrm{Linear}_{kv} : \mathbb{R}^d \to \mathbb{R}^d$ has weight matrix

$$W^{kv} \in \mathbb{R}^{d \times d}.$$

The last dimension of P is then divided across h heads:

$$P = [P_1, \ldots, P_h], \qquad P_i \in \mathbb{R}^{n \times d_h}.$$

Each head has one input-independent learnable query

$$q_i \in \mathbb{R}^{d_h}.$$

Collectively, these queries form

$$Q = [q_1; \ldots; q_h] \in \mathbb{R}^{h \times d_h}.$$

For head i, the attention weights and pooled representation are

$$\alpha_i = \mathrm{softmax}\left(\frac{P_i q_i}{\sqrt{d_h}}\right) \in \mathbb{R}^n, \qquad o_i = \alpha_i^\intercal P_i \in \mathbb{R}^{d_h}. \tag{16}$$

Although Q is independent of the input embeddings, the attention weights remain input-dependent through P_i. Consequently, different token spans produce different attention distributions and different surrogate embeddings.

The outputs of all heads are concatenated and passed through an output processor:

$$z_s = \mathrm{LN}_o\left(\mathrm{GELU}\left(\mathrm{Linear}_o\left([o_1; \ldots; o_h]\right)\right)\right) \in \mathbb{R}^d, \tag{17}$$

where the output projection has

$$W^o \in \mathbb{R}^{d \times d}.$$

Because z_s has the same dimension as an ordinary token embedding, it can be directly passed to the frozen backbone as one input-level computational unit.

D.2 Difference from Standard Self-Attention

The merge module is better understood as multi-head attention pooling rather than a standard Transformer self-attention layer. Standard self-attention produces input-dependent queries, keys, and values for every token and returns an output sequence with the same length as its input. In contrast, the merge module uses one learned query per head and reduces an entire span to a single vector.

The projected span representation P_i is shared as both the key and value for head i. This shared key-value design keeps the module compact and is sufficient for pooling short spans. In our empirical exploration, introducing separate key and value projections or additional linear layers did not provide a clear improvement, while increasing the number of trainable parameters. We therefore use a shared W^{kv} followed by the output projection W^o.

These architectural choices are empirical rather than theoretically optimal. Among the alternatives we explored, the current design provided a favorable balance among predictive alignment, training stability, and parameter efficiency.

D.3 Parameter Count

The learnable parameters of the merge module are

$$\phi = \left\{Q, W^{kv}, b^{kv}, W^{o}, b^{o}, \mathrm{LN}_{kv}, \mathrm{LN}_{o}\right\}.$$

The query matrix contains d parameters. Each of the two linear layers contains $d^2 + d$ parameters, and each LayerNorm contains $2d$ parameters. The total parameter count is therefore

$$|\phi| = 2d^2 + 7d \approx 2d^2. \tag{18}$$

For Llama-3.1-8B, where $d = 4096$, the merge module contains approximately 33.6 million parameters, corresponding to about 0.42% of an 8-billion-parameter backbone. During both base training and task adaptation, the backbone remains frozen and only these merge-module parameters are updated.

D.4 Training-time Integration

During training, the merge rules first identify a set of non-overlapping token spans. For every selected span s, the original embeddings E_s are passed through the merge module and replaced by the resulting surrogate z_s. The remaining token embeddings retain their original order, producing a shorter embedding sequence that is passed to the frozen language model.

The uncompressed sequence is processed by the same frozen backbone to provide teacher predictive distributions. The compressed and uncompressed predictions are then aligned at valid positions after merging. The merge module is optimized through soft-target cross-entropy, which is equivalent to minimizing the forward KL divergence up to the fixed entropy of the teacher distribution. No backbone parameters are updated.

Because merge locations are non-overlapping and are determined before the forward pass, spans with the same length can be pooled in parallel. Training therefore does not require recursive merging or dynamic modification of intermediate Transformer hidden states.

D.5 Prompt-time Integration

For prompt processing, the input is tokenized using the unchanged backbone tokenizer. The resulting token IDs are matched against the merge-rule set, and non-overlapping occurrences are selected from left to right. If several rules begin at the same position, the longer span is preferred.

All selected spans are replaced at the embedding level before the prompt enters the Transformer. For example,

$$[x_1, x_2, x_3, x_4] \longrightarrow [e(x_1), z_{23}, e(x_4)], \qquad z_{23} = \mathcal{M}_\phi(e(x_2), e(x_3)).$$

The compressed sequence uses contiguous position indices. In this example, the three resulting input units receive positions $[0, 1, 2]$. The backbone then performs an ordinary forward pass over this shorter embedding sequence, producing a correspondingly shorter prompt KV cache.

D.6 Decoding-time Integration

During autoregressive decoding, UMIM does not skip token-generation steps. After each new token ID is generated, we check whether the unmerged suffix of the generated token history matches a rule in $\mathcal{R}$. The matching procedure checks longer rules first. If no rule is matched, the new token embedding is processed normally and one KV state is appended to the cache.

If a newly completed suffix

$$s = (x_{T-n+1}, \ldots, x_T)$$

matches a merge rule, the merge module computes

$$z_s = \mathcal{M}_\phi\left(e(x_{T-n+1}), \ldots, e(x_T)\right).$$

At this point, the first $n-1$ tokens of the suffix have already been processed and stored in the KV cache, while the newest token has just been generated. We therefore roll back the $n-1$ cached suffix states and process z_s as one input embedding instead of processing the newest token separately. The frozen backbone then produces one surrogate KV state for the complete span.

This operation changes neither the generated token IDs nor the decoded text. It only shortens the effective sequence and KV-cache representation used by subsequent decoding steps. Previously created surrogate units are not recursively merged, ensuring that every merge corresponds directly to one rule-defined span of original token IDs.

Pseudocode 1: Multi-head pooling used by the merge module.

```
# tokens: (batch_size, span_length, d_model)

processed = token_processor(tokens)
processed = processed.view(
    batch_size,
    span_length,
    num_heads,
    head_dim,
)
processed = processed.permute(0, 2, 1, 3)

# query: (num_heads, head_dim)
query = query.unsqueeze(0).unsqueeze(2)
query = query.expand(
    batch_size,
    num_heads,
    1,
    head_dim,
)

scores = einsum(
    "bhqd,bhkd->bhk",
    query,
    processed,
) / sqrt(head_dim)

weights = softmax(scores, dim=2).unsqueeze(-1)
pooled = (processed * weights).sum(dim=2)
pooled = pooled.reshape(batch_size, d_model)

output = output_processor(pooled)
```

In summary, the merge module provides an input-level mechanism for approximating the functional effect of selected multi-token spans. It produces backbone-compatible surrogate embeddings, requires no modification to the internal Transformer layers, and supports both prompt compression and dynamic KV-cache compression during autoregressive decoding.

E Predictive Distillation Loss and Alignment Metrics

This section describes how we align the predictive distributions of the original and compressed sequences, optimize the merge module through distillation, and evaluate their predictive agreement.

Original and Compressed Prediction Paths. Let F_θ denote the frozen backbone language model and M_ϕ the trainable merge module. For an input sequence X, the teacher distribution is obtained from the original sequence:

$$P^T = \mathrm{softmax}\left(F_\theta(X)\right).$$

The compressed path replaces each selected token span with a surrogate embedding produced by M_ϕ:

$$P^S = \mathrm{softmax}\left(F_\theta(M_\phi(X))\right).$$

The two paths share the same frozen backbone. Only ϕ, the parameters of the merge module, is updated during training.

Mask Construction and Sequence Alignment. Token merging shortens the sequence and shifts subsequent positions. We therefore construct separate masks for the original teacher sequence and the compressed student sequence.

For a merged span of length n, the implementation first constructs an intermediate length-preserving layout containing $n-1$ inactive placeholders followed by one valid surrogate position. The surrogate is therefore aligned with the final position of the original span. The corresponding keep mask, denoted by M^{keep}, marks retained tokens and surrogate positions as 1, and inactive placeholders and padding positions as 0.

Let s_b be the start position of the first merged span in sample b. The teacher loss mask is

$$M^T_{b,j} = M^{\text{keep}}_{b,j} \cdot \mathbf{1}[j \geq s_b].$$

Thus, positions before the first merge are excluded because the original and compressed computations are identical before any surrogate embedding is introduced. Within each merged span, only its final position is used as the teacher-side alignment anchor.

The inactive placeholders are then removed to form the packed compressed sequence. Its attention mask, M^{attn}, marks valid compressed positions. Let

$$N_b = \sum_j M^T_{b,j}, \qquad \widetilde{L}_b = \sum_j M^{\text{attn}}_{b,j}.$$

The first $\widetilde{L}_b - N_b$ valid positions in the compressed sequence correspond to the unaffected prefix before the first merge. We exclude these positions to construct the student loss mask M^S. Consequently,

$$\sum_j M^T_{b,j} = \sum_j M^S_{b,j} = N_b.$$

Teacher and student distributions are selected by their respective masks and packed in causal order:

$$\overline{P}^T_b = \text{Pack}(P^T_b, M^T_b), \qquad \overline{P}^S_b = \text{Pack}(P^S_b, M^S_b).$$

Here, Pack selects all positions marked by the mask while preserving their order. Therefore, no explicit original-to-compressed index table is required. The two packed sequences contain the same number of predictive distributions, and their corresponding rows are used for training and evaluation.

Alignment Example. Consider the original sequence

$$X = [x_1, x_2, x_3, x_4, x_5, x_6],$$

where (x_2, x_3) and (x_5, x_6) are replaced by surrogate embeddings z_{23} and z_{56}. The masks and compressed sequence are

$$\begin{aligned} M^T &= [0, 0, 1, 1, 0, 1], \\ \widetilde{X} &= [x_1, z_{23}, x_4, z_{56}], \\ M^{\text{attn}} &= [1, 1, 1, 1], \\ M^S &= [0, 1, 1, 1]. \end{aligned}$$

The aligned prediction positions are therefore

$$(x_3, z_{23}), \qquad (x_4, x_4), \qquad (x_6, z_{56}).$$

The notation above refers to the states at which the corresponding next-token distributions are produced, rather than requiring their hidden representations to be identical.

Distillation Objective. Let

$$\mathcal{I} = \{(b, \ell) \mid 1 \leq \ell \leq N_b\}$$

denote all aligned positions in a batch, and let

$$N_{\text{valid}} = \sum_b N_b.$$

We optimize the soft-target cross-entropy between the teacher and student distributions:

$$\mathcal{L}_{\text{distill}} = -\frac{1}{N_{\text{valid}}} \sum_{(b,\ell)\in\mathcal{I}} \sum_{v=1}^{|\mathcal{V}|} \overline{P}^T_{b,\ell}(v) \log\left(\overline{P}^S_{b,\ell}(v) + \epsilon\right), \tag{19}$$

where $\mathcal{V}$ is the vocabulary and ϵ is a small constant for numerical stability. The loss is normalized by the total number of aligned positions rather than by the padded sequence length.

The forward KL divergence satisfies

$$D_{\mathrm{KL}}(P^T \| P^S) = H(P^T, P^S) - H(P^T).$$

Because the teacher and backbone are frozen, $H(P^T)$ is constant with respect to ϕ. Minimizing the soft-target cross-entropy therefore gives the same gradient with respect to the merge-module parameters as minimizing the forward KL divergence.

Practical Note on Full-Vocabulary Distillation. Our implementation uses the complete teacher distribution over the vocabulary rather than a truncated Top-k target. At each aligned position, the training objective is

$$\mathcal{L}_{\mathrm{full}} = -\sum_{v \in \mathcal{V}} P^T(v) \log P^S_\phi(v),$$

where P^T is the predictive distribution of the original frozen model and P^S_ϕ is the distribution produced by the compressed path. The only trainable component in the compressed path is the merge module.

This choice preserves the teacher distribution without introducing a truncation threshold or modifying its probability mass. A Top-k target retains only a subset $\mathcal{K}$ of the teacher vocabulary support and either removes or renormalizes the remaining probability mass. The resulting objective therefore matches the student to a modified target rather than to the original teacher distribution.

Importantly, our motivation is not that tokens outside the teacher's Top-k set necessarily receive no gradient. If a sparse teacher target is combined with a full student softmax, an outside token with target probability zero still receives a gradient proportional to its student probability. The relevant difference is that the relative probabilities assigned by the teacher outside $\mathcal{K}$ are no longer preserved. Truncation therefore changes the desired distribution and removes part of the information available for matching the original model's predictive behavior.

This distinction is particularly relevant for UMIM because the compressed path can initially differ substantially from the original path. The merge module must learn an input embedding whose effect propagates through the entire frozen Transformer. Unlike conventional LM-to-LM distillation, the trainable component does not have its own language-model layers or intermediate supervision. Full-vocabulary matching provides the merge module with the complete output-level behavior of the frozen backbone at every aligned position.

During early development, we tested teacher Top-k targets with $k \in \{128, 256, 512, 1024\}$. None of these settings produced reliable convergence or useful teacher–student alignment in our preliminary runs, whereas full-vocabulary distillation trained successfully. These exploratory runs were used to select the final training configuration, but they were not designed or retained as a controlled quantitative ablation. We therefore report this observation as a practical implementation finding rather than as evidence that Top-k distillation cannot work for UMIM.

Full-vocabulary distillation increases training-time memory and computation because teacher and student probabilities are retained over the complete vocabulary. This cost is limited to merge-module training and does not affect inference. More sophisticated sparse or adaptive distillation objectives may provide a better memory–quality trade-off, but we leave their systematic study to future work. In our experiments, full-vocabulary matching was the most reliable configuration among the variants we tested. Top-k and Top-p are used only as alignment metrics and are not training objectives.

Alignment Metrics. In addition to the distillation loss, we report four metrics over the same aligned positions. These metrics are used only to evaluate predictive agreement and are not additional training objectives.

- **Top-1 Agreement.** Let

$$y_i^T = \arg\max_v \overline{P}_i^T(v), \qquad y_i^S = \arg\max_v \overline{P}_i^S(v).$$

Top-1 agreement is

$$\mathrm{Top1} = \frac{1}{N_{\mathrm{valid}}} \sum_{i=1}^{N_{\mathrm{valid}}} \mathbf{1}[y_i^T = y_i^S].$$

- **Top-k Overlap.** Let $\mathcal{K}_i^T$ and $\mathcal{K}_i^S$ be the sets of the k most probable tokens under the teacher and student distributions. We use $k = 3$ and $k = 10$:

$$\mathrm{Overlap}_k = \frac{1}{N_{\mathrm{valid}}} \sum_{i=1}^{N_{\mathrm{valid}}} \frac{|\mathcal{K}_i^T \cap \mathcal{K}_i^S|}{k}.$$

- **Top-p Overlap.** Let $\mathcal{P}_i^T(p)$ and $\mathcal{P}_i^S(p)$ be the smallest token sets whose cumulative probability reaches the threshold p. Their overlap is

$$\mathrm{Overlap}_p = \frac{1}{N_{\mathrm{valid}}} \sum_{i=1}^{N_{\mathrm{valid}}} \frac{|\mathcal{P}_i^T(p) \cap \mathcal{P}_i^S(p)|}{\min\left(|\mathcal{P}_i^T(p)|, |\mathcal{P}_i^S(p)|\right)}.$$

  The value of p is specified by the corresponding experimental setting.

- **Mean Reciprocal Rank.** For each aligned position, we take the teacher's top-1 token y_i^T and measure its rank $\mathrm{rank}_S(y_i^T)$ under the student distribution:

$$\mathrm{MRR} = \frac{1}{N_{\mathrm{valid}}} \sum_{i=1}^{N_{\mathrm{valid}}} \frac{1}{\mathrm{rank}_S(y_i^T)}.$$

Implementation Summary. Algorithm 1 summarizes the alignment and loss computation. Padding is introduced only after teacher and student positions have been selected, and padded rows are excluded from both the loss and all alignment metrics.

Algorithm 1: Aligned Predictive Distillation

Require: Teacher probabilities P^T, compressed probabilities P^S, keep mask M^{keep}, compressed attention mask M^{attn}, and first merge position s_b for each sample
Ensure: Distillation loss $\mathcal{L}_{\mathrm{distill}}$
Initialize aligned teacher list $\mathcal{T}$ and student list $\mathcal{S}$
For each sample b:
  $M_b^T \leftarrow M_b^{\mathrm{keep}}$
  Set $M_{b,j}^T \leftarrow 0$ for all $j < s_b$
  $N_b \leftarrow \sum_j M_{b,j}^T$
  $\widetilde{L}_b \leftarrow \sum_j M_{b,j}^{\mathrm{attn}}$
  $M_b^S \leftarrow M_b^{\mathrm{attn}}$
  Set the first $\widetilde{L}_b - N_b$ valid entries of M_b^S to 0
  $T_b \leftarrow \mathrm{Pack}(P_b^T, M_b^T)$
  $S_b \leftarrow \mathrm{Pack}(P_b^S, M_b^S)$
  Verify that $|T_b| = |S_b| = N_b$
  Append T_b and S_b to $\mathcal{T}$ and $\mathcal{S}$
Compute $\mathcal{L}_{\mathrm{distill}}$ from all aligned rows using Equation 19
return $\mathcal{L}_{\mathrm{distill}}$

F Inference with UMIM: Prompt and Decoding-Time Merging

UMIM can be applied during prompt processing and autoregressive decoding. Prompt merging compresses the input before prefill, while decoding-time merging further compresses generated-token KV states. The two mechanisms can be used separately for ablation, while the full UMIM pipeline applies both.

The merge decision depends only on realized token IDs and is independent of the token-selection rule. It can therefore be combined with greedy decoding, top-k sampling, or top-p sampling by changing

the token-selection step. Our reference implementation uses greedy decoding. The implementation requires the backbone to accept input embeddings and expose a KV cache that can be cropped during generation.

Rule Matching. For prompt merging, we scan the tokenized input from left to right. At each position, candidate spans are checked in descending length order, i.e., 4-, 3-, and then 2-grams in our experiments. If a match is found, we select the longest matching span and advance to the first token after it. Otherwise, we advance by one token. This produces a set of non-overlapping merge spans.

The same longest-first policy is used during decoding, but only the unmerged raw-token suffix ending at the newly generated token needs to be checked. A surrogate created by an earlier merge is treated as one internal state and is not recursively included in another merge. This prevents overlapping or nested merges.

Prompt Merging. Given an input prompt with token IDs

$$X = [x_1, \ldots, x_{L_p}],$$

we first identify a set of non-overlapping spans $\mathcal{S}_p$. For every span $(i, j) \in \mathcal{S}_p$, the original embeddings

$$[e(x_i), \ldots, e(x_{j-1})]$$

are passed through the merge module to obtain one surrogate embedding. The original span is then replaced by this surrogate.

For efficiency, all selected spans of the same length can be processed together in one batched merge-module call. After replacing all selected spans, the compressed prompt length is

$$\widetilde{L}_p = L_p - \sum_{(i,j)\in\mathcal{S}_p} (j - i - 1).$$

The compressed embeddings use contiguous positions and are passed to the frozen backbone for prefill. Consequently, the initial KV cache contains $\widetilde{L}_p$ states rather than L_p states.

If decoding-time merging is disabled, generation then proceeds normally without further cache modification. Prompt merging therefore directly reduces prefill sequence length and initial KV-cache length, while leaving the subsequent autoregressive procedure unchanged.

Pseudocode 2: Prompt-Only Merging

```
# x: input text
# R[n]: merge rules of length n
# M: trained merge module
# LM: frozen language model

token_ids = Tokenizer(x)
spans = []
i = 0

# Left-to-right, longest-first matching
while i < len(token_ids):
    matched = False

    for n in [4, 3, 2]:
        span = tuple(token_ids[i:i+n])

        if len(span) == n and span in R[n]:
            spans.append((i, i+n))
            i = i + n
            matched = True
            break

    if not matched:
        i = i + 1

# Spans of the same length can be pooled in batch
surrogates = {}
```

```
for n in [2, 3, 4]:
    spans_n = [
        (start, end) for start, end in spans
        if end - start == n
    ]

    if len(spans_n) > 0:
        span_embeddings = stack([
            model_embedding(token_ids[start:end])
            for start, end in spans_n
        ])
        pooled = M(span_embeddings)

        for (start, end), embedding in zip(
            spans_n, pooled
        ):
            surrogates[start] = (end, embedding)

# Construct the packed compressed prompt
compressed_embeddings = []
i = 0

while i < len(token_ids):
    if i in surrogates:
        end, embedding = surrogates[i]
        compressed_embeddings.append(embedding)
        i = end
    else:
        compressed_embeddings.append(
            model_embedding(token_ids[i])
        )
        i = i + 1

E = stack(compressed_embeddings)

generated_ids = autoregressive_decode(
    LM,
    prompt_embeddings=E,
    merge_during_decoding=False
)

return Tokenizer.decode(generated_ids)
```

Decoding-Time Merging and KV-Cache Update. Decoding-time merging is performed after each new token ID is selected. Suppose the newly generated token completes an unmerged suffix

$$g = (x_{t-n+1}, \ldots, x_t)$$

that appears in the merge rules. At this point, the newest token x_t has been sampled from the current logits but has not yet been forwarded through the model. Therefore, only the preceding $n-1$ tokens in the matched suffix already have KV entries.

We crop these $n-1$ KV entries, compute the surrogate embedding

$$m_g = M_\phi\left(e(x_{t-n+1}), \ldots, e(x_t)\right),$$

and forward m_g through the frozen backbone using the cropped cache. This forward pass produces one new key and value state for the surrogate at every Transformer layer.

Thus, UMIM does not directly construct or average the original KV states. It removes the available raw-token KV entries and lets the frozen backbone compute a new KV state from the surrogate embedding. If the cache length before processing x_t is L_C, the cache length after merging an n-gram becomes

$$L'_C = L_C - (n-1) + 1.$$

By comparison, processing x_t without merging would produce a cache of length $L_C + 1$. The matched span therefore occupies one KV position instead of n positions.

If no rule matches the suffix, the newly generated token is forwarded normally and one KV entry is appended. In either case, one token is selected during every autoregressive step. UMIM does not skip token-generation steps; it reduces the effective sequence and cache lengths used by later steps.

Pseudocode 3: Prompt and Decoding-Time Merging

```
# x: input text
# R[n]: merge rules of length n
# M: trained merge module
# LM: frozen language model

token_ids = Tokenizer(x)

# Apply prompt merging and prefill the model
prompt_spans = find_nonoverlapping_spans(
    token_ids,
    R,
    longest_first=True
)
E = build_compressed_embeddings(
    token_ids,
    prompt_spans,
    M
)
logits, cache = LM.forward(
    inputs_embeds=E,
    use_cache=True
)

# generated_ids preserves the actual output tokens
generated_ids = []

# generated_state is used only for suffix matching.
# A merged marker prevents overlapping merges.
generated_state = []

while not finished:
    y = select_next_token(logits)
    generated_ids.append(y)

    if y is an end-of-sequence token:
        break

    generated_state.append(y)

    span = longest_matching_raw_suffix(
        generated_state,
        R,
        lengths=[4, 3, 2]
    )

    if span is None:
        step_embedding = model_embedding(y)

    else:
        n = len(span)

        # The newest token has not entered the cache.
        # Only the previous n-1 entries are removed.
        cache = crop_cache(
            cache,
            cache_length(cache) - (n - 1)
        )

        span_embeddings = model_embedding(span)
        step_embedding = M(span_embeddings)

        # Replace the raw suffix with an internal marker.
```

```
        generated_state = (
            generated_state[:-n]
            + [MERGED(span)]
        )

    logits, cache = LM.forward(
        inputs_embeds=step_embedding,
        past_key_values=cache,
        use_cache=True
    )

return Tokenizer.decode(generated_ids)
```

Output Preservation. Surrogate embeddings and merged markers are used only for the model's internal computation. We always preserve the complete sequence of generated token IDs separately. The final output is decoded from these original generated IDs, not from the compressed internal representation. Therefore, decoding-time merging reduces the internal sequence and KV-cache lengths without removing tokens from the returned text.

Computational Scope. Prompt merging provides an immediate reduction in prefill length, and this shorter prompt cache is retained throughout generation. Decoding-time merging further reduces generated-token KV-cache growth whenever a generated suffix matches the merge rules. However, it cannot skip autoregressive generation steps and introduces additional work for suffix lookup, merge-module evaluation, and cache cropping. This explains why the realized decoding speedup can be smaller than the prefill speedup even when the cache length is reduced.

The reported implementation uses standard Python and PyTorch operations without custom CUDA or Triton kernels. Since the surrogate embedding for a fixed token span is context-independent, frequently used surrogates could also be precomputed and retrieved through a lookup table. Suffix matching, surrogate lookup, and KV-cache updates could further be fused into optimized kernels. These engineering optimizations are not included in the reported implementation and are left for future work.

G Task Adaptation with SFT and RL

This section describes how UMIM is adapted to downstream tasks while keeping the backbone LLM fully frozen. Task adaptation is distinct from the task-agnostic evaluation in the main experiments. In the task-agnostic setting, we directly use the merge rules mined from WikiText-103 and the corresponding WikiText-trained base merge module, without using downstream training data. Task adaptation instead reconstructs merge rules from the target task's training split and optionally updates the merge module through SFT and RL.

The complete adaptation pipeline contains three stages:

1. Construct task-specific merge rules from the downstream training split.
2. Initialize from the WikiText-trained base merge module and perform distillation-based SFT.
3. Initialize from the SFT checkpoint and further optimize the merge module through RL.

The backbone LLM remains frozen throughout all stages.

G.1 Task-Specific Merge-Rule Construction

Given the target-task training set $\mathcal{D}_{\text{task}}$, we tokenize its examples using the backbone LLM's tokenizer and count adjacent token spans. In the reported task-adaptation experiments, we use task-specific bigrams and vary the frequency threshold to control the rule-set size and resulting merge ratio.

For a multiple-choice task, each training sequence is constructed by concatenating the task input with its correct answer or completion. We count the bigrams in these sequences and retain candidates whose frequency is no smaller than a threshold τ:

$$\mathcal{R}_{\text{task}}^{(\tau)} = \{g : \text{count}_{\mathcal{D}_{\text{task}}}(g) \geq \tau\}\,.$$

When applying the rules, we scan each token sequence from left to right using non-overlapping matching. If the bigram beginning at the current position belongs to $\mathcal{R}_{\text{task}}^{(\tau)}$, it is selected and the scan advances by two tokens. Otherwise, the scan advances by one token.

A lower threshold retains more task-specific bigrams and therefore increases the achievable merge ratio, but it may also introduce spans with less training support. A higher threshold gives a more conservative rule set. The different thresholds produce the accuracy–compression curves reported in Figure 3 and Appendix Figure 5.

If no additional training is desired, the task-specific rules can be used directly with the fixed WikiText-trained base merge module. This corresponds to the `WT + Rules` setting.

G.2 Distillation-Based SFT

To further adapt the merge module to the task-specific rules, we initialize it from the WikiText-trained base merge module M_{ϕ_0} and perform SFT only on the merge module. This SFT stage is not conventional label-only fine-tuning and does not update the backbone LLM. Instead, it continues to use predictive distillation.

For a task-training sequence X, the frozen backbone first processes the original uncompressed sequence and produces the teacher distribution:

$$P^T = \text{softmax}\left(F_\theta(X)\right).$$

The task-specific rules $\mathcal{R}_{\text{task}}$ determine the spans to be merged. Replacing these spans with surrogate embeddings produces the compressed sequence

$$\widetilde{X} = C_\phi\left(X; \mathcal{R}_{\text{task}}\right).$$

The same frozen backbone then processes the compressed sequence and produces the student distribution:

$$P_\phi^S = \text{softmax}\left(F_\theta(\widetilde{X})\right).$$

The original and compressed distributions are aligned using the mask-and-pack procedure described in Section E. The merge module is optimized using the aligned soft-target cross-entropy:

$$\phi_{\text{SFT}} = \arg\min_\phi \mathcal{L}_{\text{distill}}\left(P^T, P_\phi^S\right).$$

Only ϕ receives gradients, while the backbone parameters θ remain frozen. For example, in HellaSwag, we construct each SFT sequence by concatenating the context with the correct ending. The merge module therefore learns to preserve the original model's predictive behavior under the target-task distribution and task-specific rules.

This stage corresponds to `WT + FT`. It combines the general span-to-surrogate initialization learned from WikiText-103 with domain-specific distillation signals from the downstream task.

G.3 RL Adaptation

Distillation-based SFT aims to preserve the original model's overall predictive behavior, but it does not directly optimize the downstream decision objective. We therefore initialize a policy merge module from the SFT checkpoint $M_{\phi_{\text{SFT}}}$ and further optimize it through RL.

For a multiple-choice example, let x denote the task context, y^+ the correct ending, and y^- an incorrect ending. We construct

$$X^+ = [x; y^+], \qquad X^- = [x; y^-].$$

During this stage, the merge rules are applied to the task context shared by the correct and incorrect sequences. Both candidates therefore use the same compressed context, and the optimization signal is determined by their different endings.

Table 8: Task-adaptation and ablation settings. The backbone LLM remains frozen in every setting.

Setting	Initialization	Merge Rules	Distillation SFT	RL	Backbone
`WT + Rules`	WikiText base	Task-specific	No	No	Frozen
`WT + FT`	WikiText base	Task-specific	Yes	No	Frozen
`Task-Adapted`	WikiText base	Task-specific	Yes	Yes	Frozen
`Task-trained`	Random	Task-specific	From scratch	No	Frozen
`WT + DPO`	WikiText base	Task-specific	No	Yes	Frozen

We compute the mean token log-probability of each candidate ending:

$$s_\phi(x, y) = \frac{1}{|y|} \sum_{t=1}^{|y|} \log p_{\theta,\phi}\left(y_t \mid C_\phi(x; \mathcal{R}_{\text{task}}), y_{<t}\right).$$

The policy merge module and a frozen reference merge module are initialized from the same checkpoint. In the final `Task-Adapted` setting, both are initialized from $M_{\phi_{\text{SFT}}}$. The reference merge module remains fixed, while only the policy merge module receives gradients.

We use the following reference-regularized pairwise RL objective:

$$\begin{aligned}\mathcal{L}_{\text{RL}} = -\log \sigma\Big(\beta\big[\,&\big(s_\phi(x, y^+) - s_{\text{ref}}(x, y^+)\big) \\ &- \big(s_\phi(x, y^-) - s_{\text{ref}}(x, y^-)\big)\,\big]\Big),\end{aligned} \tag{20}$$

where β controls the strength of the update relative to the frozen reference. When a question contains multiple incorrect candidates, we first average the pairwise losses within that question and then average across questions in the batch.

This stage does not retrain the language model. It only changes how the merge module constructs surrogate embeddings, encouraging the compressed frozen LLM to assign higher scores to correct answers. Distillation-based SFT first preserves general predictive behavior under the target-task distribution, while RL subsequently optimizes the task-level decision objective. Their combination forms the final `Task-Adapted` setting.

G.4 Adaptation and Ablation Settings

Table 8 summarizes the task-adaptation and ablation settings used in Figure 3 and Figure 4.

In `Task-trained`, a new merge module is randomly initialized and trained directly on the target-task corpus using the same distillation objective. This setting measures the benefit of the WikiText-trained initialization. In `WT + DPO`, we skip distillation-based SFT and apply the RL objective directly to the WikiText-trained base merge module. These two settings are ablations rather than the final task-adaptation pipeline.

G.5 Task-Adaptation Experimental Configuration

This subsection provides the experimental configurations used for the task-adaptation results in Section 4.5. All experiments use Llama-3.1-8B. Except for the `Task-trained` ablation, the merge module is initialized from the base merge module trained on WikiText-103. Throughout adaptation, the tokenizer, input embedding matrix, and backbone LLM remain frozen; only the merge module parameters are updated.

Task-specific rule construction. For HellaSwag, task-specific merge rules are constructed using only the training split. Each training sequence is formed by concatenating the context with its correct ending and is truncated or padded to 128 tokens. We count adjacent token pairs and retain bigrams whose frequencies exceed thresholds in $\{2000, 200, 50, 5, 1\}$. These thresholds produce the operating points reported in Figure 3, with realized merge rates ranging from 12.10% to 43.09%. Candidate bigrams are applied from left to right using non-overlapping matching, so each token belongs to at most one merged span.

Table 9: Task-adaptation training configurations. Batch denotes the micro-batch size per GPU. The RL stage is implemented using DPO.

Task	Stage	Rule Threshold	Batch/GPU	Learning Rate	Epochs	Patience	β
HellaSwag	SFT	$\{2000, 200, 50, 5, 1\}$	32	1×10^{-4}	5	3	–
HellaSwag	RL	$\{2000, 200, 50, 5, 1\}$	32	5×10^{-5}	2	2	0.1
ARC-Easy	SFT	5	2	5×10^{-4}	12	3	–
ARC-Easy	RL	5	4	5×10^{-5}	10	3	0.1
ARC-Challenge	SFT	5	4	5×10^{-4}	10	5	–
ARC-Challenge	RL	5	2	5×10^{-4}	10	3	0.1

For ARC-Easy and ARC-Challenge, rules are constructed from the question tokens in each task's training split. We use bigram rules with frequency threshold 5 and a maximum sequence length of 128 tokens. The training sequence contains the question and its correct answer, while merging is applied to the question portion. Validation examples are used for model selection but not for rule construction or parameter updates.

SFT configuration. The SFT stage retains the predictive-distillation objective used to train the base merge module. For each training sequence, the frozen uncompressed LLM produces the teacher next-token distributions, while the compressed model produces the student distributions after applying the task-specific merge rules. After aligning the corresponding predictive positions, we minimize cross-entropy from the teacher distributions to the student distributions. Thus, although the adaptation data come from the downstream task, the backbone LLM is not trained with task labels; gradients are applied only to the merge module.

The `WT + FT` setting initializes the merge module from the WikiText-trained checkpoint. The `Task-trained` ablation uses the same task data and distillation objective but trains a newly initialized merge module instead of starting from the WikiText checkpoint. SFT checkpoints are selected according to validation distillation loss, with early stopping.

RL configuration. The RL stage is implemented using DPO. For each multiple-choice example, the correct option is used as the preferred continuation, and each incorrect option forms a rejected continuation. The preferred and rejected sequences share the same compressed context or question, ensuring that the optimization targets how the adapted merge module affects downstream predictions.

Both the policy merge module and the frozen reference merge module are initialized from the same SFT checkpoint. We compute the mean token log-probability of each answer continuation and optimize the DPO objective to increase the policy's relative preference for the correct continuation. The backbone LLM and reference merge module remain frozen, and only the policy merge module is updated. The `WT + DPO` ablation uses the same RL procedure but initializes directly from the WikiText-trained merge module, without the preceding SFT stage. RL checkpoints are selected according to validation accuracy.

Optimization settings. All adaptation runs use eight 48 GB GPUs, DeepSpeed ZeRO stage 2, and bf16 precision. We use AdamW with gradient accumulation set to 1, gradient clipping at 1.0, and a warmup-decay schedule with 10% of the optimization steps used for warmup. Weight decay is 10^{-3} for SFT and 10^{-4} for RL. The remaining task-specific configurations are summarized in Table 9.

G.6 Implementation Procedures

SFT and RL are implemented as two separate training stages rather than a joint optimization procedure. The SFT stage first produces a task-specific merge-module checkpoint. The RL stage then loads this checkpoint and uses it to initialize both the trainable policy merge module and the frozen reference merge module.

SFT implementation. Listing 4 summarizes the distillation-based SFT stage.

Pseudocode 4: Distillation-Based SFT for Task Adaptation

```
# D_task: downstream training split
# F_theta: frozen backbone LLM
# M_base: WikiText-trained base merge module

# Construct task-specific merge rules
R_task = mine_frequent_rules(D_task)

# Initialize from the WikiText checkpoint
M_sft = load_wikitext_merge_module()
freeze(F_theta)

for original_sequence in task_training_sequences:
    # Teacher path: original sequence
    with no_gradient():
        teacher_probs = F_theta(
            original_sequence
        )

    # Student path: compressed sequence
    compressed_sequence = merge(
        original_sequence,
        R_task,
        M_sft
    )
    student_probs = F_theta(
        compressed_sequence
    )

    # Align teacher and student positions
    teacher_aligned, student_aligned = align(
        teacher_probs,
        student_probs
    )

    loss = soft_target_cross_entropy(
        teacher_aligned,
        student_aligned
    )

    # The backbone remains frozen
    update_only(M_sft, loss)

save_merge_module(
    M_sft,
    "task_sft_checkpoint"
)
```

The SFT checkpoint is saved independently after distillation training. Only the merge-module parameters are stored and passed to the next stage; the frozen backbone is not modified.

RL implementation. Listing 5 summarizes the subsequent RL stage. It is launched as a separate training run after the SFT checkpoint has been produced.

Pseudocode 5: RL Adaptation from the SFT Checkpoint

```
# F_theta: frozen backbone LLM
# R_task: task-specific merge rules
# D_pairs: correct/incorrect task pairs

M_sft = load_merge_module(
    "task_sft_checkpoint"
)

# Both modules start from the same SFT weights
M_policy = copy(M_sft)
M_reference = frozen_copy(M_sft)
```

```
freeze(F_theta)
freeze(M_reference)

for question in D_pairs:
    context = question.context
    correct_answer = question.correct_answer
    wrong_answers = question.wrong_answers

    pair_losses = []

    for wrong_answer in wrong_answers:
        # The same compressed context is used for
        # the correct and incorrect candidates.
        policy_positive = ending_score(
            F_theta,
            M_policy,
            context,
            correct_answer,
            R_task
        )
        policy_negative = ending_score(
            F_theta,
            M_policy,
            context,
            wrong_answer,
            R_task
        )

        with no_gradient():
            reference_positive = ending_score(
                F_theta,
                M_reference,
                context,
                correct_answer,
                R_task
            )
            reference_negative = ending_score(
                F_theta,
                M_reference,
                context,
                wrong_answer,
                R_task
            )

        pair_loss = rl_objective(
            policy_positive,
            policy_negative,
            reference_positive,
            reference_negative
        )
        pair_losses.append(pair_loss)

    # Average over candidates from the same question
    loss = mean(pair_losses)

    # Only the policy merge module is updated
    update_only(M_policy, loss)

save_merge_module(
    M_policy,
    "task_adapted_checkpoint"
)
```

The two stages are therefore connected only through the merge-module checkpoint. Distillation-based SFT first adapts the WikiText-trained module to the target-task distribution and merge rules. RL then starts from this adapted checkpoint and further optimizes task-level decisions. At no point are the

SFT and RL objectives optimized jointly, and the backbone LLM remains frozen throughout both stages.

For the `WT + DPO` ablation, the RL procedure is unchanged, but both the policy and reference merge modules are initialized directly from the WikiText-trained base checkpoint rather than the SFT checkpoint. For `Task-trained`, the SFT procedure is initialized randomly instead of using the WikiText-trained merge module.

H Base Merge-Module Training Setup

This section describes the corpora, optimization settings, hardware, and model-specific configurations used to train the base merge modules. The task-adaptation stages are described separately in Section G.

H.1 Training Corpora

WikiText-based training. For Llama-3.1-8B, Llama-3.2-1B, and GPT-2-XL, we train the base merge modules using WikiText-103 (`wikitext-103-raw-v1`). Each corpus split is tokenized using the corresponding backbone tokenizer. We concatenate the tokenized documents within each split, discard the incomplete remainder, and partition the resulting token stream into non-overlapping sequences of 512 tokens.

We use the official WikiText-103 training, validation, and test splits. The merge rules are mined only from the training split and are then applied to all three splits. The resulting token IDs, attention masks, and merge positions are precomputed and stored for distributed training.

Although Llama-3.1-8B and Llama-3.2-1B use different backbone sizes, they have compatible tokenization in our experiments and therefore produce the same WikiText-derived rule counts. GPT-2-XL uses its own tokenizer and a separately constructed rule set.

Math-reasoning training corpus. For DeepScaleR-1.5B-Preview, we construct a separate math-reasoning corpus to demonstrate that a merge module can also be trained for long-form reasoning domains. The source questions are taken from `KbsdJames/Omni-MATH` and `RUC-AIBOX/STILL-3-Preview-RL-Data`. We retain questions of at most 200 tokens and use `agentica-org/DeepScaleR-1.5B-Preview` to sample three reasoning trajectories for each question. Generation uses nucleus sampling with $p = 0.7$, temperature 1.0, and a maximum total sequence length of 2048 tokens.

The generated texts, including the source questions and sampled reasoning, are tokenized and concatenated into a continuous token stream. This stream is then divided into non-overlapping sequences of 1024 tokens. Thus, 1024 is the fixed training-sequence length after corpus construction; individual generated trajectories are not required to contain exactly 1024 tokens.

We divide the resulting sequences into 90% training, 5% validation, and 5% test splits. The DeepScaleR merge rules are mined only from the training portion of this generated reasoning corpus. This setup is separate from the WikiText-based training used for the Llama and GPT-2 backbones.

H.2 Optimization and Distributed Training

All base merge modules are trained on eight NVIDIA GPUs. Depending on machine availability, the experiments use NVIDIA RTX A6000, RTX 6000 Ada, or L40S GPUs, each with 48 GB of memory. Llama-3.1-8B and DeepScaleR-1.5B-Preview use DeepSpeed ZeRO stage 2, while Llama-3.2-1B and GPT-2-XL use PyTorch DistributedDataParallel. All merge-module training is performed in `bfloat16` precision.

We use AdamW with weight decay 1×10^{-3}. The learning-rate schedule contains a linear warmup over the first 10% of training steps, followed by linear decay. Each run is configured for at most 15 epochs. We select the best checkpoint according to validation distillation loss and use early stopping with patience 3. Training can be resumed from a saved checkpoint when interrupted by the compute-time limit.

The original and compressed prediction paths use the same frozen backbone checkpoint. The original sequence produces the teacher distribution, while the compressed sequence produces the student

Table 10: Base merge-module training configurations. All runs use eight 48 GB GPUs with the backbone frozen.

Backbone	Corpus	Method	Length	Batch/GPU	LR	d
Llama-3.1-8B	WT-103	ZeRO-2	512	3	8×10^{-4}	4096
Llama-3.2-1B	WT-103	DDP	512	4	8×10^{-4}	2048
GPT-2-XL	WT-103	DDP	512	4	7×10^{-4}	1600
DeepScaleR-1.5B-Preview	Math	ZeRO-2	1024	3	8×10^{-4}	1536

distribution through the merge module and the frozen backbone. Gradients are propagated only to the merge-module parameters; all backbone parameters remain frozen.

The preprocessed token IDs and attention masks are stored as `.pt` files, while variable-length merge positions are stored as `.pkl` files. They are loaded through a custom dataset and distributed using `DistributedSampler`.

During training, we record the distillation cross-entropy, Top-1 agreement, Top-3 overlap, Top-10 overlap, Top-p overlap with $p = 0.9$, MRR, and the average number of merged spans. These quantities are evaluation and monitoring metrics; only the distillation cross-entropy is optimized. Training and validation statistics are logged with Weights & Biases.

H.3 Model-Specific Configurations

Table 10 reports the model-specific training configurations. Batch size is reported per GPU. All merge modules use four pooling heads.

Here, d denotes the backbone input-embedding dimension and therefore also the input and output dimension of the corresponding merge module. A separate base merge module is trained for each backbone because the module operates directly in that backbone's embedding space.

I Additional Experiments and Evaluation

This section provides additional evaluation protocols and results for perplexity, open-ended generation, downstream tasks, long-form mathematical reasoning, and specialized-domain adaptation.

I.1 Perplexity Evaluation

We evaluate language-modeling performance on WikiText-103, BookCorpus, and OpenWebText. The WikiText-derived merge rules and base merge module are applied directly to these datasets without task-specific training. Thus, BookCorpus and OpenWebText evaluate task-agnostic transfer beyond the merge module's WikiText training corpus.

A direct perplexity calculation on only the compressed sequence would omit the original tokens that are absorbed into surrogate embeddings. We therefore use a two-stage accounting procedure that assigns likelihood to all $N - 1$ next-token targets in an original sequence of length N.

In the first stage, the sequence is partitioned into unmerged tokens and merged spans. A single compressed forward pass evaluates transitions between consecutive compressed elements. When the next element is a merged span, its first constituent token is used as the next-token target. In the second stage, auxiliary prefix forwards evaluate the remaining constituent tokens inside every merged span. The two losses are summed and normalized by the original number of next-token targets. This procedure makes UMIM perplexity comparable to the original token-level perplexity while retaining the compressed sequence in the main forward pass.

Pseudocode 6: Two-Stage Token-Level PPL Accounting

```
# x: original token IDs of length N
# R: merge rules
# M: trained merge module
# LM: frozen language model
```

```
elements = merge_longest_first(x, R)

def embed(element):
    if isinstance(element, tuple):
        token_embs = stack([
            model_embedding(token_id)
            for token_id in element
        ])
        return M(token_embs)
    return model_embedding(element)

def first_token(element):
    if isinstance(element, tuple):
        return element[0]
    return element

# Stage 1: transitions between compressed elements
main_inputs = elements[:-1]
main_targets = [
    first_token(element)
    for element in elements[1:]
]

main_embeddings = stack([
    embed(element)
    for element in main_inputs
])
main_logits = LM.forward(main_embeddings)

total_nll = 0
target_count = 0

for position, target in enumerate(main_targets):
    total_nll += -log_prob(
        main_logits[position],
        target
    )
    target_count += 1

# Stage 2: tokens skipped inside merged spans
for index, element in enumerate(elements):
    if not isinstance(element, tuple):
        continue

    prefix = elements[:index]

    for offset in range(1, len(element)):
        # Use the compressed preceding context, followed
        # by the observed raw prefix of the current span.
        context = (
            prefix
            + list(element[:offset])
        )
        context_embeddings = stack([
            embed(item)
            for item in context
        ])
        logits = LM.forward(context_embeddings)

        target = element[offset]
        total_nll += -log_prob(
            logits[-1],
            target
        )
        target_count += 1

assert target_count == len(x) - 1
```

```
PPL = exp(total_nll / target_count)
```

This two-stage procedure is used only for token-level likelihood accounting. It is different from decoding-time KV-cache rollback, which is described in Section F.

I.2 Text Generation

I.2.1 Generation Setup

We conduct the open-ended generation experiments using Llama-3.1-8B. UMIM applies prompt merging before prefill and continues to check generated raw-token suffixes for decoding-time merging. Surrogate states are used only internally; the returned text always preserves the complete generated token-ID sequence.

For qualitative greedy-decoding results, we randomly select five prompts from the WikiText-103 test split. Depending on the length needed to present a complete example, we generate either 20 or 30 new tokens. We additionally use the prompt "Tell me something about Lakers." as an illustrative Top-p example. For this prompt, we draw three independent samples with $p = 0.9$ and temperature 0.8.

For quantitative evaluation, examples are constructed from the WikiText-103 test split. The first 100 tokens of each evaluated sequence are used as the prompt, and tokens 101–200 are used as the reference. Each model generates 100 new tokens conditioned on the prompt.

We evaluate three decoding strategies:

- greedy decoding;
- Top-p sampling with $p = 0.9$ and temperature 0.8;
- random sampling from the full next-token distribution without nucleus truncation.

For Top-p and random sampling, we perform five independent generation runs for each input and report the average results.

Generation quality is evaluated using repetition and diversity metrics (Rep-2/3/4 and Dist-2/3/4), as well as ROUGE-2, ROUGE-L, ROUGE-Lsum, BERTScore, chrF, and Mauve.

I.2.2 Qualitative Analysis

Tables 11 and 12 provide qualitative examples under greedy and Top-p decoding. The brackets indicate token spans merged internally by UMIM. These brackets are visual annotations only and are not part of the returned text.

The examples show that UMIM can produce fluent continuations while repeatedly replacing multi-token spans with surrogate states. They should be interpreted as qualitative illustrations rather than evidence of exact semantic equivalence in every context.

Figure 8 compares attention distributions before and after merging. The visualization shows that surrogate states continue to receive attention from later tokens rather than being ignored. However, attention weight alone does not prove complete semantic preservation or layer-wise equivalence.

We also observe that merged embeddings have near-zero cosine similarity with the static embeddings of their individual constituent tokens. This indicates that the merge module learns representations distinct from simply copying or averaging the constituent embeddings. It does not, by itself, establish that the surrogate is context-dependent or perfectly preserves the original span.

I.2.3 Quantitative Results

The results depend on the decoding strategy. Under Top-p sampling, UMIM reduces repetition and improves all reported diversity metrics, while the reference-based similarity metrics remain close to the base model. Under greedy decoding, repetition is slightly reduced and the remaining metrics are generally close, with UMIM obtaining a higher Mauve score.

Under unrestricted random sampling, the results are more mixed. UMIM improves several reference-based similarity metrics, including ROUGE and BERTScore, but produces lower Dist-n and Mauve

Table 11: Qualitative generation results with greedy decoding. Brackets mark spans merged internally by UMIM.

Prompt	Base Model	UMIM
Traditional Chinese literary criticism emphasized **[the life of the]** author when interpreting a work	The author's life was seen as the key to understanding the work.	The author's life **[was seen as]** the key **[to understanding the work]**.
In **[2025]**, **[he moved to the]** institute **[in an attempt]** to study.	He was a very good student, and he was very good at his job.	He **[was a student of]** **[the institute and]** **[was a member of]** the institute's football team.
In the spring, **[his youngest son]**, Ian, was born.	He was a beautiful baby boy, and he was the light of his father's life.	He **[was a beautiful baby]**, **[with a head of thick]**, dark **[hair and]** a round, chubby face.
The first public library in Bintulu **[was built in]** 1971.	It was located at the Bintulu District Office. The library was later moved to the Bintulu Town Council building in 1973.	It **[was located at]** the Bintulu District Office. The library was **[later moved to the]** Bintulu Town Council building **[in 1973]**.
Everyone considered the reviewers **[an improvement on the]** conference.	The reviewers were very helpful and the reviews were very thorough. The reviewers were very helpful and the reviews were very thorough.	The reviewers were very professional **[and the conference]** was very well organized. The conference **[was a great success]** and we are looking forward to next year's conference.

Table 12: Qualitative generation results with Top-p decoding. Brackets mark spans merged internally by UMIM.

Prompt: Tell me something about Lakers.

Method	Sample 1	Sample 2	Sample 3
Base	Why did they change their name from Minneapolis Lakers to Los Angeles Lakers? What are their nicknames? What are their logos? Who are their rivals? Who are their greatest players? What are their colors? What is their motto?	What do you think of Kobe? Lakers are the best team in the league, I believe. I think they will win the championship again. Kobe is the best player in the league, and I think he will win the MVP this year.	I've heard that they are the best team in the NBA, but I'm not sure about that. Who are the best players on the team? What are their chances to win the NBA championship this season? I want to see them in action.
UMIM	Tell me something about Lakers. I never **[been to a]** game. I want **[to know everything]** about **[the team and the]** players **[and the history of]** the team. I'm like **[a kid in]** **[a candy store]**. **[I just want to]** know everything.	What **[do you think of]** their chances **[to make the playoffs]** this year? What **[do you think of]** **[the Clippers]**? Do you think they'll be **[able to win the]** championship this year? What **[do you think of]** the Lakers' chances?	What is **[the most exciting thing]** about Lakers? Why do you love them? What makes **[them special]**? The Lakers **[are a very]** special team. They have had **[some of the greatest]** players **[to ever play the]** **[game of basketball]**.

scores and higher Rep-3/4. We therefore do not claim uniform preservation across every decoding distribution. Instead, these results show that the effect of compression interacts with the decoding strategy, with greedy and Top-p decoding providing the most stable generation behavior in our experiments.

I.3 Downstream Evaluation Protocols

I.3.1 Multiple-Choice QA

For PIQA, COPA, OpenBookQA, ARC-Easy, and ARC-Challenge, we follow a likelihood-based multiple-choice evaluation protocol. For each candidate answer, we concatenate the task prompt, question, and candidate into one sequence and compute its length-normalized negative log-likelihood. The candidate with the lowest loss is selected as the model prediction.

In the task-agnostic setting, every task uses the fixed merge rules mined from WikiText-103 and the WikiText-trained base merge module. No task-specific rules, SFT, or RL are used for these main QA results. The reported UMIM TR is the naturally realized reduction produced by the fixed WikiText-derived rules on each task.

Table 13: N-gram repetition and diversity results. Lower Rep-n and higher Dist-n are better. Bold indicates the better result within each decoding pair.

Method	Rep-2 ↓	Dist-2 ↑	Rep-3 ↓	Dist-3 ↑	Rep-4 ↓	Dist-4 ↑
Top-p (Base)	11.76	39.58	10.93	70.14	6.78	85.59
Top-p (UMIM)	**11.28**	**43.49**	**9.08**	**75.59**	**4.55**	**89.92**
Greedy (Base)	14.03	**40.99**	15.32	**60.38**	13.97	**69.75**
Greedy (UMIM)	**13.34**	39.04	**14.44**	58.96	**12.74**	69.30
Sampling (Base)	11.06	**50.72**	**7.13**	**83.10**	**2.84**	**94.68**
Sampling (UMIM)	**10.77**	33.96	12.02	59.34	9.76	74.25

Table 14: Text-similarity results against the held-out WikiText continuation. Higher is better. Bold indicates the better result within each decoding pair.

Method	ROUGE-2	ROUGE-L	ROUGE-Lsum	BERTScore	chrF	Mauve
Top-p (Base)	**5.49**	**17.85**	**19.66**	**83.75**	**28.53**	**34.87**
Top-p (UMIM)	4.87	17.17	19.14	83.57	28.39	33.90
Greedy (Base)	**5.82**	**18.89**	**20.03**	**83.35**	**26.99**	64.28
Greedy (UMIM)	5.38	18.44	19.47	82.98	26.08	**67.21**
Sampling (Base)	4.12	16.23	18.24	83.25	**28.28**	**36.81**
Sampling (UMIM)	**5.45**	**18.09**	**19.58**	**83.44**	27.67	27.41

I.3.2 CNN/DailyMail Summarization

We evaluate CNN/DailyMail using Llama-3.1-8B. The merge rules are mined from the WikiText-103 training split, and the WikiText-trained base merge module is applied directly without summarization-specific training. Generation uses greedy decoding with a fixed output length of 80 tokens.

In this experiment, token merging is applied only to the input prompt. It therefore evaluates task-agnostic prompt and prefill compression rather than decoding-time KV-cache compression.

For Selective Context, we adjust token-level prompt reduction to match UMIM's naturally realized TR. For LLMLingua-1, we use its default compression method, while LLMLingua-2 uses its small-model variant. For Sink Attention, we use four sink tokens and a sliding window equal to 16% of the total prompt-plus-generation length. Because LLMLingua and Selective Context do not always permit exact control of the retained token count, their realized reductions may differ slightly from the target budget.

The results show that the fixed WikiText-trained UMIM module remains competitive with, and in the reported metrics outperforms, the evaluated baselines at comparable token-reduction levels on an unseen summarization domain.

I.3.3 Long-Form Mathematical Reasoning

We evaluate AIME2024 and AMC using DeepScaleR-1.5B-Preview and the merge module trained on the math-reasoning corpus described in Section H.1. Because these tasks may generate very long reasoning traces, token merging is applied only to the input prompts. Generation is allowed to continue for up to 15,000 tokens, and performance is evaluated using pass@16.

This experiment evaluates prompt compression for long-form mathematical reasoning. It does not evaluate decoding-time merging or generated-token KV-cache reduction. Under this setting, UMIM achieves strong performance and outperforms Selective Context and LLMLingua in the reported results.

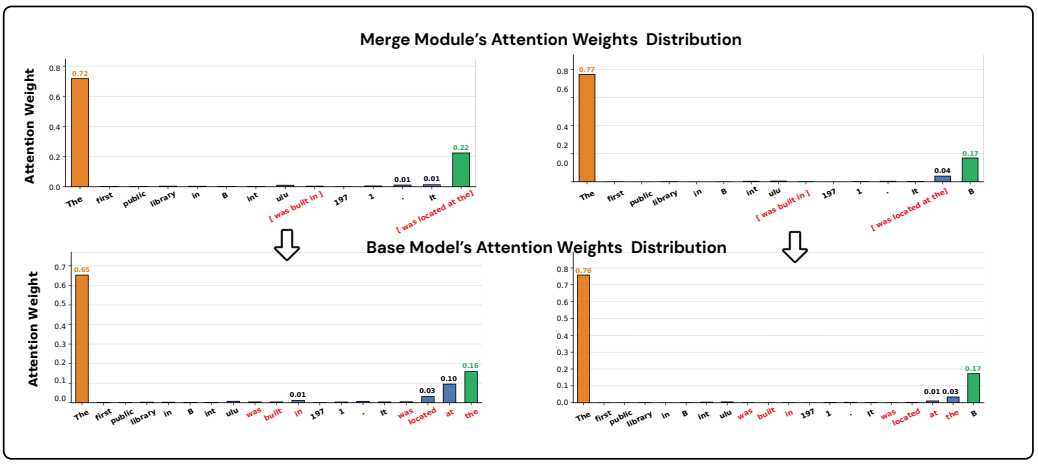


Figure 8: Attention distributions before and after applying UMIM. The visualization shows that surrogate states remain involved in subsequent attention, but does not imply exact attention or semantic equivalence.

Table 15: Results on RULER four-needle retrieval.

Context	Baseline Acc. (%)	UMIM Acc. (%)	Δ Acc. (pp)	TR (%)
1K	99.8	99.2	-0.6	33.3
2K	100.0	99.2	-0.8	35.5
4K	99.6	92.6	-7.0	34.9
8K	99.0	76.8	-22.2	34.6
16K	96.6	33.8	-62.8	33.7

I.3.4 Long-Context Evaluation

Beyond the long-form mathematical reasoning evaluation with DeepScaleR, we further evaluate UMIM on RULER four-needle retrieval, multi-document question answering, and long-document summarization. These experiments examine UMIM under long-range retrieval, cross-document information integration, and long-sequence generation settings.

All experiments use Llama-3.1-8B and its base merge module trained only on 512-token WikiText-103 sequences. The backbone LLM and merge module remain frozen, without long-context SFT, RL, or other parameter updates. Because the original WikiText-derived merge rules have limited coverage on these datasets, directly applying them produces too few merges for a meaningful compression evaluation. We therefore follow the `WT + Rules` setting: high-frequency n-gram rules are re-mined from each evaluation corpus and paired with the frozen WikiText-trained merge module. This changes only the set of eligible merge spans; neither the merge module nor the backbone LLM is retrained.

RULER Four-Needle Retrieval. We first evaluate UMIM on RULER four-needle retrieval. This task requires the model to retrieve multiple pieces of information located at different positions in a long context, directly testing whether token merging preserves information needed for long-range retrieval. TR denotes the percentage of input tokens removed.

Multi-Document Question Answering. We next evaluate whether UMIM can preserve and integrate evidence distributed across multiple documents. We use similar token-reduction ratios across context lengths to examine how performance changes as the input becomes longer.

Long-Context Summarization. Finally, we evaluate UMIM on GovReport and QMSum. In these experiments, merging is applied during both prompt processing and autoregressive decoding. Prompt TR denotes input-token reduction, Gen-KV TR denotes the reduction in generated-token

Table 16: Results on multi-document question answering.

Length	Baseline Acc. (%)	UMIM Acc. (%)	Δ Acc. (pp)	TR (%)
2K	74.8	70.4	-4.4	33.96
4K	74.2	59.8	-14.4	34.89
8K	60.6	50.2	-10.4	33.79
16K	58.0	19.4	-38.6	34.29

Table 17: Long-context summarization results. Prompt TR measures input-token reduction, while Gen-KV TR and Cumulative KV TR measure generated-token KV-cache reduction. All values are percentages.

Dataset	Method	Prompt TR	Gen-KV TR	Cumulative KV TR	R-1	R-2	R-L	R-Lsum	ROUGE Avg.	BERTScore
GovReport	Baseline	–	–	–	46.28	17.47	21.51	43.82	32.27	82.88
GovReport	UMIM	14.50	12.59	13.04	46.60	17.18	21.98	44.05	32.45	84.75
QMSum	Baseline	–	–	–	21.29	6.55	15.07	18.59	15.37	82.82
QMSum	UMIM	10.90	15.51	16.84	18.13	5.54	13.20	15.61	13.12	81.40

KV-cache length at the end of generation, and Cumulative KV TR denotes the reduction in cumulative generated-token KV-cache usage throughout autoregressive decoding. All lengths are measured using the Llama-3.1-8B tokenizer.

GovReport has a mean input length of 10,275 tokens, a median of 8,712, and a range of 2,054–51,427 tokens. QMSum has a mean input length of 13,917 tokens, a median of 13,002, and a range of 2,633–30,438 tokens. We evaluate 200 examples from each dataset.

Discussion. On RULER, UMIM remains close to the uncompressed baseline at context lengths of 1K and 2K while removing approximately one-third of the input tokens. As the context becomes longer and relevant information must be preserved over greater distances, the performance gap increases. Multi-document question answering shows a similar overall trend: UMIM retains performance relatively well at shorter lengths while achieving substantial compression, but preserving evidence becomes more difficult as context length increases.

For long-document summarization, UMIM maintains comparable overall quality on GovReport while reducing both prompt length and generated-token KV-cache usage, with slight improvements on several metrics. On QMSum, UMIM provides both prompt and decoding-time KV compression, although with some reduction in summarization quality. These results indicate that long-context tasks differ in their sensitivity to local token-span compression.

Importantly, these experiments directly transfer a base merge module trained only on 512-token WikiText-103 sequences to retrieval and question-answering contexts of up to 16K tokens and real documents containing up to 51K tokens. No long-context training, SFT, or RL is performed. The results therefore characterize how the base merge module generalizes well beyond its training length and domain, while also showing that training sequence length affects the preservation of long-range dependencies.

With sufficient training resources, long-context data should be incorporated from the beginning when training a base merge module for very long-context retrieval, multi-document question answering, long-form reasoning, or long-document summarization. Systematically training the merge module on long-context corpora with corresponding merge rules should further improve long-range performance while retaining UMIM's benefits in effective sequence length and KV-cache compression.

I.4 Specialized-Domain Evaluation: Code-Adapted UMIM

The main task-agnostic experiments use WikiText-derived merge rules and a WikiText-trained merge module. Code generation represents a more specialized domain with different syntax, symbolic structure, and token-frequency statistics. We therefore include a supplementary experiment to test whether the UMIM formulation remains effective when both the merge rules and merge module are constructed from in-domain code data.

Table 18: HumanEval results on 164 tasks using Llama-3.1-8B at a 25% token-reduction level.

Method	pass@1 (%)	Passed	Failed
Baseline	62.2	102	62
UMIM (code-trained)	50.6	83	81
LLMLingua-2	1.2	2	162
Selective Context	0.0	0	164

We mine a code-specific merge set from the DeepMind CodeContests corpus and train a code-specific UMIM module. This is a domain-specific training experiment rather than direct transfer from the WikiText base module. Because the amount of code training data and available compute are limited, we treat this result as a supplementary validation rather than a fully optimized code-compression system.

We evaluate the code-adapted merge module on HumanEval using Llama-3.1-8B at a 25% token-reduction level. The same compression level is used for LLMLingua-2 and Selective Context. HumanEval contains 164 tasks, and pass@1 is reported together with the number of passed and failed tasks.

Despite the limited in-domain training data, code-adapted UMIM retains approximately 81% of the original pass@1 performance, from 62.2% to 50.6%. Under the same token-reduction level, LLMLingua-2 and Selective Context retain substantially less performance.

This result does not imply that domain-specific training is required for every downstream task. Rather, it complements the task-agnostic experiments by showing that when a target domain has substantially different token statistics, UMIM can also be instantiated with domain-specific rules and training data. The remaining gap to the uncompressed baseline indicates that structurally sensitive code generation remains challenging and that larger in-domain training corpora or task adaptation may further improve performance.

J Efficiency Analysis

J.1 Empirical Efficiency Evaluation and Throughput Analysis

During autoregressive decoding, each step generates one new token while attending to the accumulated KV cache. Under common small-batch and long-context inference settings, this process is often strongly affected by memory bandwidth and repeated KV-cache access. Reducing the effective sequence and KV-cache lengths can therefore reduce both attention computation and memory traffic.

Evaluation protocol. To examine whether token reduction translates into actual inference speedup, we measure prefill and decoding throughput separately, following the evaluation protocol used by Zip2Zip. All experiments use Llama-3.1-8B on a single NVIDIA A100 GPU with batch size 1 and bf16 precision.

We evaluate a controlled 50% merge ratio while varying the prompt length. This controlled setting isolates the throughput effect of sequence-length reduction and is separate from the downstream evaluations, where TR is naturally determined by merge-rule coverage. A setting denoted by "$256 + 256$" contains a 256-token prompt followed by 256 generated tokens; the remaining settings follow the same notation. We keep the generation length fixed at 256 tokens and vary the prompt length from 256 to 2,048 tokens.

Each configuration is first executed for 10 warm-up runs, followed by 10 timed runs. GPU synchronization is performed around each timed region, and we report the mean throughput across the 10 measured runs.

Throughput is computed with respect to the original, uncompressed token count. For UMIM, prefill throughput therefore measures how many original prompt tokens are covered per second after compression, rather than the number of compressed embeddings passed to the backbone LLM. Decoding throughput measures the number of autoregressively generated tokens per second. Our current implementation is a non-fused Python-level prototype and does not use customized CUDA or Triton kernels for rule matching, merge-module execution, or KV-cache rollback.

Table 19: Throughput comparison for Llama-3.1-8B under a controlled 50% merge ratio. Throughput is measured with respect to the original token count. A setting denotes prompt length + generation length. Relative improvement is computed as $(\mathrm{UMIM}/\mathrm{Base} - 1) \times 100\%$.

Setting	Prefill Throughput (tokens/sec)			Decode Throughput (tokens/sec)		
	Base	UMIM	Relative	Base	UMIM	Relative
$256 + 256$	10,531.8	15,410.8	$+46.3\%$	64.7	65.6	$+1.3\%$
$512 + 256$	14,210.6	30,978.5	$+118.0\%$	65.3	65.9	$+0.9\%$
$1024 + 256$	16,138.3	50,516.0	$+213.0\%$	65.1	66.4	$+1.9\%$
$2048 + 256$	16,592.5	62,124.7	$+274.4\%$	64.8	66.4	$+2.5\%$

Results. UMIM improves prefill throughput across all evaluated context lengths. As the prompt length increases from 256 to 2,048 tokens, the prefill speedup grows from approximately $1.46\times$ to $3.74\times$. Prompt merging shortens the embedding sequence before it enters the backbone LLM, so every subsequent Transformer layer operates on the compressed sequence. The resulting computational benefit becomes larger as the original context length increases.

During autoregressive decoding, UMIM improves throughput by 0.9%–2.5% across the evaluated settings. These improvements are substantially smaller than the prefill gains but remain positive under the current non-fused implementation.

Interpretation. The difference between prefill and decoding follows from their different computational processes. During prefill, all matching prompt spans can be merged before the backbone LLM is executed. The reduced embedding sequence therefore lowers computation and memory use throughout the complete prefill pass.

During decoding, UMIM cannot skip autoregressive generation steps. After each new token is generated, UMIM checks whether the current sequence suffix matches a merge rule. If a match is found, KV-cache rollback replaces the corresponding KV entries with one merged KV state. The resulting benefit comes from reducing the effective sequence and KV-cache lengths used by subsequent decoding steps, rather than from reducing the number of generated tokens or autoregressive forward passes.

Our current decoding prototype performs suffix matching, merge-module execution, KV-cache rollback, and cache updates through Python-level control flow. These operations introduce scheduling overhead and additional small GPU kernel launches, offsetting part of the benefit from the shorter KV cache. The current results nevertheless show that the reduction in subsequent attention computation is sufficient to produce positive decoding throughput gains. They should be interpreted as evidence of initial practical feasibility rather than the performance of a fully optimized inference system.

Potential engineering optimizations. Under the current context-independent merge rules, a surrogate embedding depends only on the identities of the tokens in its span. Surrogate embeddings can therefore be precomputed and stored in a lookup table, avoiding repeated merge-module execution during inference. A more optimized implementation could further fuse suffix matching, embedding lookup, KV-cache rollback, and cache updates into CUDA or Triton kernels.

These optimizations would not change the underlying UMIM algorithm, but they require additional systems engineering and evaluation. The reported experiments are intended to establish that UMIM's reduction in effective sequence length can translate into actual inference speedup, rather than to present a fully optimized production inference system.

Summary. The throughput results confirm that UMIM reduces not only the nominal sequence length but also the realized inference cost. Prompt merging provides substantial prefill acceleration, particularly for longer contexts. Decoding-time merging cannot bypass autoregressive steps, but it continually shortens the KV cache used by subsequent attention operations and provides modest positive throughput improvements even in the current Python-level prototype.

Table 20: Validation distribution-preservation metrics under different training-corpus n-gram frequency thresholds. Higher values indicate closer agreement with the original uncompressed model.

Backbone	τ	**Top-1**	**Top-3**	**Top-10**	**Top-p**	**MRR**
Llama-3.2-1B	500	0.8437	0.8356	0.8433	0.9504	0.9078
Llama-3.2-1B	50	0.7883	0.7840	0.7943	0.9320	0.8691
Llama-3.2-1B	5	0.7410	0.7414	0.7528	0.9155	0.8297
GPT-2-XL	500	0.8414	0.8456	0.8579	0.9548	0.9048
GPT-2-XL	50	0.7588	0.7707	0.7893	0.9297	0.8447
GPT-2-XL	5	0.6955	0.7128	0.7359	0.9065	0.7928

Table 21: Merge-module capacity ablation on Llama-3.2-1B with $\tau = 5$.

Architecture	**Top-1**	**Top-3**	**Top-10**	**Top-p**	**MRR**
One attention layer	0.7410	0.7414	0.7528	0.9155	0.8330
Three attention layers	0.7427	0.7415	0.7534	0.9158	0.8341

K Ablation Study

We study three aspects of UMIM: the frequency threshold used to select merge candidates, the capacity of the merge module, and an alternative merge-rule construction strategy. Unless otherwise specified, all results are measured on held-out validation data using distribution-preservation metrics between the original and compressed models. Top-p uses $p = 0.9$.

K.1 Frequency Threshold and Statistical Support

The frequency threshold τ controls which n-grams are included in the merge-rule set. An n-gram is retained only if it appears at least τ times in the training corpus. A lower threshold admits a larger and less frequently observed candidate set, generally increasing potential merge coverage. A higher threshold retains spans with stronger statistical support, allowing the merge module to observe each candidate in more training contexts.

To evaluate this trade-off, we construct merge rules exclusively from the training split and evaluate the resulting merge modules on unseen validation sequences. Table 20 reports the results for Llama-3.2-1B and GPT-2-XL under thresholds $\tau \in \{500, 50, 5\}$.

Across both backbones, all alignment metrics improve consistently as τ increases. For Llama-3.2-1B, Top-1 increases from 0.7410 at $\tau = 5$ to 0.8437 at $\tau = 500$, while MRR increases from 0.8297 to 0.9078. GPT-2-XL shows the same pattern.

These results indicate that frequency serves not only as a rule-selection criterion but also as a minimum statistical-support condition. Frequently occurring spans provide more natural contexts from which the shared merge module can learn their surrogate representations. Lower thresholds expand the candidate set and potential compression coverage, but they also introduce spans supported by fewer training examples, reducing the reliability of the learned approximation. The threshold therefore provides a practical control over the compression–alignment trade-off.

K.2 Merge-Module Capacity

We next examine whether increasing the capacity of the merge module substantially improves predictive alignment. We train one-layer and three-layer attention-pooling variants on WikiText-103 using the same merge rules with $\tau = 5$. Both variants are evaluated on the same held-out validation data.

Increasing the module depth from one to three attention layers produces only marginal improvements. Top-1 increases from 0.7410 to 0.7427, while the remaining metrics change by at most approximately 0.0011. These results suggest that the shallow attention-pooling module already provides sufficient capacity for learning the surrogate embeddings in this setting. We therefore use the lighter one-layer design in the main experiments.

Table 22: Validation results using LM-confidence-based merge rules on Llama-3.2-1B. TR denotes the realized token reduction.

TR	Top-1	Top-3	Top-10	Top-p	MRR
10.3%	0.9107	0.9104	0.9152	0.9717	0.9496

K.3 Alternative Merge-Rule Construction

High-frequency n-grams provide a simple and statistically supported way to instantiate UMIM, but frequency is not the only possible merge-rule criterion. We therefore evaluate an alternative strategy based on the language model's token-level confidence.

For each input chunk, we compute the autoregressive probability assigned by Llama-3.2-1B to every observed token. We identify consecutive spans in which all token probabilities exceed 0.5 and retain spans containing at least two tokens. To remain compatible with the current online merging implementation, candidates longer than four tokens are discarded. The resulting 2- to 4-gram spans are used as merge rules, and a merge module is trained using the same predictive-distillation procedure as in the main method.

The LM-confidence-based rules achieve strong distribution-preservation metrics at a realized TR of 10.3%. Because this operating point has a relatively modest token reduction, the result should be interpreted as a feasibility study rather than a matched-TR comparison with frequency-based rules. It nevertheless demonstrates that UMIM is not tied to a single rule-construction heuristic. Other criteria can also provide effective merge candidates when the selected spans are sufficiently predictable and supported by the training data.

Overall, these ablations support three conclusions. First, the frequency threshold controls a practical trade-off between candidate coverage and statistical support. Second, increasing merge-module depth provides little benefit in the evaluated setting, supporting the lightweight architecture used in the main experiments. Third, alternative merge-rule criteria can also work, showing that high-frequency n-grams are an effective implementation of UMIM rather than a fixed limitation of the framework.

K.4 Positional Encoding under Token-Span Merging

Compressed-position convention. UMIM does not modify the positional-encoding formulation or parameters of the backbone LLM, nor does it introduce position interpolation or an additional position-reindexing mechanism. A merged span occupies one effective position in the compressed sequence, and subsequent positions are assigned contiguously according to the effective sequence length.

For example, consider the original sequence

$$[x_1, x_2, x_3, x_4]$$

with positions

$$[0, 1, 2, 3].$$

After merging x_2, x_3 into a surrogate token x_{23}, the input becomes

$$[x_1, x_{23}, x_4],$$

with compressed positions

$$[0, 1, 2].$$

Thus, UMIM does not preserve the original position indices of the merged tokens or manually compensate the positions of subsequent tokens. Training and inference consistently use this compressed-position convention.

Functional approximation rather than embedding equivalence. It is intuitive to view the surrogate token as using one token to represent the information originally carried by multiple tokens. More precisely, however, UMIM does not require the surrogate embedding to be semantically identical to any constituent embedding or to reproduce every intermediate state of the original span. The surrogate is a learned continuous soft token whose purpose is to induce similar predictive behavior after being processed by the frozen LLM.

Let F_θ denote the frozen language model, X and P the original token embeddings and positions, and $\widetilde{X}_\phi$ and $\widetilde{P}$ the compressed embeddings and positions produced using the merge module M_ϕ. The optimization target can be summarized as

$$F_\theta(X, P) \approx F_\theta(\widetilde{X}_\phi, \widetilde{P})$$

at aligned prediction positions. Importantly, $\widetilde{P}$ contains the contiguous compressed positions. The merge module therefore learns to approximate the joint predictive effect of

$$\text{span replacement} + \text{position compaction},$$

rather than first reproducing the original span and then separately removing the effect of the position shift.

This explains why using the same position convention during training and inference is important. During predictive distillation, the compressed sequence is passed through the frozen LLM with the same positions that will be used at inference time. Gradients are propagated through the complete frozen computation to the merge module, allowing the surrogate embedding to compensate empirically for both replacing the span and compacting later positions. This consistency does not imply exact equivalence, but it avoids an additional train–test mismatch and directly includes position compaction in the optimization objective.

Why one surrogate token can still be effective. A d-dimensional surrogate embedding is not restricted to any discrete token in the original vocabulary. It is a learnable point in the continuous embedding space. After entering the frozen Transformer, it interacts with the preceding context, positional encoding, attention layers, and MLP layers. Consequently, although the input-side surrogate for a fixed n-gram is context-independent, its hidden representations inside the Transformer remain context-dependent.

A single surrogate KV state cannot generally reproduce every possible behavior of multiple original KV states under all future queries. However, UMIM does not optimize for such layer-wise equivalence. Next-token prediction only requires preserving the information that materially affects the final predictive distribution. Frequent spans are observed in many training contexts, allowing the merge module to learn a surrogate that minimizes the expected predictive discrepancy over those contexts. Thus, UMIM learns a distribution-dependent functional approximation rather than claiming an exact replacement for every possible context.

Structural effect under RoPE. For models using RoPE, position compaction does not arbitrarily perturb the relative position of every token pair. Let the effective compressed position of an original token at position t be

$$\pi(t) = t - r(t),$$

where $r(t)$ is the number of token positions removed before t. For two original positions i and j, their compressed relative distance is

$$\pi(j) - \pi(i) = (j - i) - \big(r(j) - r(i)\big).$$

If both tokens have the same number of removed positions before them, such that

$$r(i) = r(j),$$

then their relative distance is unchanged:

$$\pi(j) - \pi(i) = j - i.$$

Therefore, token pairs within a region that does not cross a merged span preserve their original RoPE relative offsets, even though their absolute indices may shift. The relationships that change are primarily those crossing a merged span and those involving the surrogate token itself. For a pair crossing one or more merged spans, the relative distance is shortened by the number of removed positions between them.

This effect can also be characterized locally at the attention-logit level. For fixed query and key vectors, a RoPE interaction with relative distance Δ can be written as

$$\ell_{ij} = q_i^\top R_\Delta k_j,$$

where R_Δ is the corresponding RoPE rotation. If compression shortens the relative distance by δ, the positional contribution becomes

$$\widetilde{\ell}_{ij} = q_i^\top R_{\Delta-\delta} k_j.$$

The resulting difference satisfies

$$\left|\ell_{ij} - \widetilde{\ell}_{ij}\right| \le \|q_i\|_2 \|k_j\|_2 \, \|R_\Delta - R_{\Delta-\delta}\|_2 \, .$$

For the blockwise rotations used by RoPE,

$$\|R_\Delta - R_{\Delta-\delta}\|_2 = 2 \max_m \left|\sin\left(\frac{\delta\omega_m}{2}\right)\right|,$$

where ω_m is the angular frequency of the m-th RoPE component. This shows that the positional change is a structured perturbation controlled by the amount of distance compression and the RoPE frequencies.

This local relation is not an end-to-end error bound. In a multi-layer Transformer, the queries, keys, values, and hidden states also change after merging. Nevertheless, it explains why compressed positions do not destroy all relative-position structure: within-region offsets remain unchanged, while cross-span distances are shortened in a systematic manner.

Learned absolute positional embeddings. For models using learned absolute positional embeddings, such as GPT-2-XL, the RoPE relative-offset property above does not apply. After a merge, subsequent tokens receive different absolute positional embeddings. UMIM does not assume that this difference disappears. Instead, the merge module is trained under the same compressed-position convention used at inference time, allowing predictive distillation to jointly account for surrogate replacement and absolute-position shifting.

The effectiveness of UMIM on both RoPE-based Llama models and GPT-2-XL provides empirical evidence that the method is not tied to one positional-encoding architecture. However, these results demonstrate the viability of the complete method under different positional mechanisms; they do not imply that position shifts have no effect.

Positions during autoregressive decoding. The same convention is used during autoregressive decoding. When a newly generated token completes a mergeable suffix, UMIM rolls back the KV-cache entries corresponding to that span and replaces them with one surrogate state. Subsequent positions are assigned according to the updated effective cache length. The physical KV-cache length therefore becomes shorter than the number of tokens generated in the original uncompressed token stream.

This design allows UMIM to retain the standard positional computation of the frozen backbone without modifying its positional-encoding architecture. At the same time, it shortens relative distances across previously merged spans. The merge module is trained to compensate for this effect empirically, but exact compensation is not guaranteed for every context or sequence length.

Empirical evidence and scope. All UMIM results reported in this paper use the compressed-position convention without any additional positional adaptation. The language-modeling, QA, summarization, and generation results therefore provide end-to-end evidence that the frozen LLM can effectively use the learned surrogate embeddings despite position compaction. The qualitative generations and attention analysis further show that merged representations remain actively involved in the model computation rather than being ignored. Moreover, the task-adaptation experiments show that, while keeping the backbone completely frozen, adapting only the small merge module can further reduce the predictive discrepancy caused by compression.

These experiments do not separately identify how much error comes from surrogate replacement and how much comes from position compaction, since the two effects are jointly optimized in the current design. They nevertheless demonstrate that, without changing the backbone positional mechanism, UMIM can achieve meaningful compression while retaining strong predictive performance across multiple models, tasks, and positional-encoding architectures.

We do not claim positional invariance or exact internal equivalence between the compressed and uncompressed computations. Preserving the original position indices is a meaningful alternative

design, but a controlled comparison would require separately training

$$\text{compressed-position training} + \text{compressed-position inference}$$

and

$$\text{preserved-position training} + \text{preserved-position inference}.$$

For dynamic decoding, the preserved-position variant would also need to maintain both the logical position in the original token stream and the physical storage position in the compressed KV cache, together with corresponding changes to generation, cache rollback, attention masking, and position bookkeeping. We leave the systematic implementation and evaluation of this alternative positional design to future work.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction summarize the proposed input-level merging framework, the frozen-backbone setting, and the main empirical findings. The claims are supported by the method description and experiments in Sections 3–4

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We include a separate Limitations section discussing the scope of the current study and directions for broader evaluation and adaptive merging.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper does not present formal theoretical results or proofs; the contribution is methodological and empirical.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper describes the model backbones, training corpora, merge-rule construction, evaluation metrics, downstream tasks, and baselines needed to reproduce the main experimental results.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: Code and trained merge-module checkpoints are not released at submission time. We provide experimental details and use publicly available datasets where possible.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The experimental setup specifies the backbones, training data, merge-rule thresholds, evaluation datasets, metrics, and baseline methods. Additional implementation details are provided in the appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: We do not report error bars or statistical significance tests, as the main experiments are conducted under fixed model and greedy evaluation settings.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper reports efficiency analysis and provides the compute setting used for evaluating runtime and memory behavior. Additional resource details are included in the experimental setup or appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification:The research uses existing language models and text datasets for compression and evaluation, and we are not aware of deviations from the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

Justification: The work may reduce inference cost and improve accessibility of language-model deployment. As a general compression method for language models, it could also inherit risks associated with the underlying models.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release a high-risk generative model or new scraped dataset. The method is a compression module applied to existing models.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper cites the existing models, datasets, and baselines used in the experiments. We follow the corresponding access terms and report public sources where applicable.

    Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper does not introduce a new dataset or benchmark asset. If code or checkpoints are released later, they will be documented with appropriate usage instructions.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or human-subject studies.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

Justification: The paper does not involve crowdsourcing or human-subject studies, so IRB approval is not applicable.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: LLMs are the subject of compression and evaluation, but no LLM is used as an important, original, or non-standard component for developing the core method.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.