LATENTLM: LEARNING A DISCRETE INTERNAL LANGUAGE FOR LARGE LANGUAGE MODEL

Anonymous authors
Paper under double-blind review

ABSTRACT

In this paper, we study whether an off-the-shelf LLM can be adapted into a discrete, variable-length token compressor and decompressor for long-context processing. To this end, we design a self-expressive autoencoding framework that fine-tunes a pretrained LLM with lightweight LoRA adapters to map long texts into compact sequences of learned latent codes, termed Z-tokens, and to decode them back into natural language or task outputs. The resulting representation is content-adaptive: less predictable or more information-dense segments can receive more Z-tokens, while redundant regions can be represented more compactly through a budget-aware length regularizer. We evaluate our method on seven long-context datasets (e.g., RULER, HotpotQA, and QuALITY), demonstrating that, compared with several strong baselines, our approach achieves better reconstruction quality and downstream performance, especially in scenarios with multiple queries on the same input, while reducing effective context length and memory usage and improving throughput in the generation stage. This design supports direct decoding from compressed contexts, and we further explore promising paradigms for autoregressive generation in the Z-token space, providing a practical interface for efficient long-context reasoning.

1 INTRODUCTION

Large language models (LLMs) have achieved remarkable success in various fields Brown et al. (2020); Touvron et al. (2023). However, their ability to handle long contexts remains fundamentally limited by the quadratic complexity of the attention mechanism Vaswani et al. (2023). As sequence lengths grow to tens or even hundreds of thousands of tokens, the computational and memory overhead of the attention mechanism makes large-scale inference, long document understanding, and multi-hop retrieval prohibitively expensive. To mitigate this, significant research has focused on token compression, which aims to distill lengthy sequences into shorter, information-dense representations Ge et al. (2024); Snell et al. (2022); Chevalier et al. (2023); Mu et al. (2024). Existing approaches can be broadly categorized into two categories. Heuristic token merging methods, such as similarity-based merging Bolya et al. (2023); Zhong et al. (2025); Zhang et al. (2023); Li et al. (2024a); Cai et al. (2025), attention-based pruning Kim et al. (2022), and saliency-guided aggregation Li et al. (2023), attempt to remove or merge redundant tokens based on local similarity or attention importance scores. While these techniques are effective in shortening sequence lengths, tokens that appear redundant due to similarity may carry discourse or pragmatic meaning that is difficult to recover. The second category includes learning-based or autoencoding methods such as AutoCompressor Chevalier et al. (2023), SACLiu et al. (2026), 500xCompressorLi et al. (2024b), ICAE Ge et al. (2024), or Gist Tokens Mu et al. (2024). AutoCompressor, 500xCompressor, SAC and ICAE use LLMs to generate fixed-length compressed context vectors that can be re-expanded for downstream tasks by training auxiliary networks or soft embedding modules. Gist Tokens force the model to compress cues into fixed-length lexical units through an attention masking mechanism. These methods often treat compression as an external, auxiliary task, decoupled from the LLM's primary generative reasoning process. Moreover, their reliance on fixed compression ratios prevents adaptation to varying information density and typically forgoes faithful reconstruction of the original input, limiting their applicability in scenarios that demand verifiability or fine-grained access to the source content. In this work, we depart from prevailing heuristic and learned-embedding approaches by introducing a novel insight: leveraging the semantic knowledge of the LLM to serve as both a

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

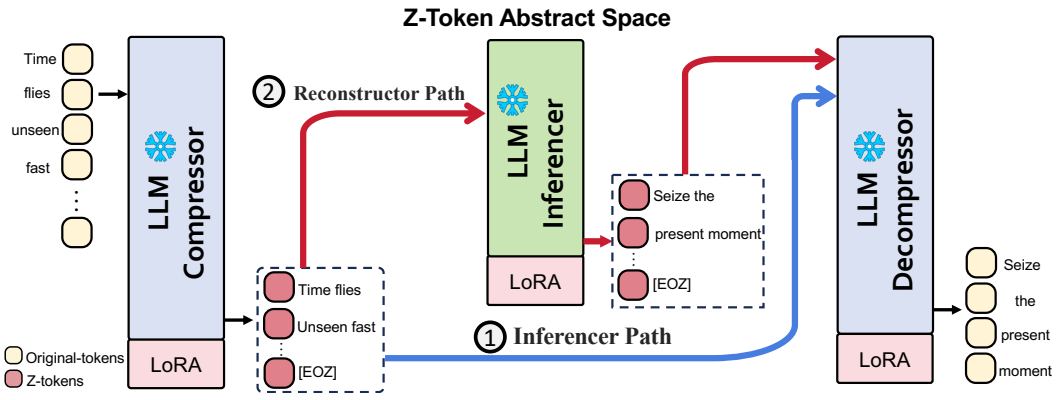

Figure 1: Our framework consists of three components: a compressor, an inference module, and a decompressor. It supports two usage paradigms. In the first, we directly feed the compressed representation to the decompressor to perform the downstream task. In the second, we first run the inference module in the Z-token space and then invoke the decompressor to recover task outputs at the token level.

token compressor and decompressor. Instead of representing the compressed context as a continuous vector or a fixed number of memory slots, we expand the LLM vocabulary using the learned Z-token codebook and generate compressed tokens autoregressively. Our method is predicated on the fundamental insight that LLMs, trained on vast corpora, are not merely statistical models but powerful engines for semantic abstraction and conceptual distillation Wei et al. (2022); Bommasani et al. (2022); Delétang et al. (2024). We observe that natural language is inherently compressive; it exhibits a strong tendency to encode complex ideas into compact lexical units. Idioms such as "double-edged sword" encapsulate intricate trade-offs, and metaphors like "tip of the iceberg" summarize extensive underlying complexities. This phenomenon implies that LLMs themselves are capable of efficient compression. Methodologically, we propose an autoencoding training paradigm, where an input sequence X is mapped into a compressed sequence of discrete Z-tokens, Z, by a LLM compressor, and subsequently reconstructed or continued by a LLM decompressor. The training objective can be viewed as a form of bilingual translation, where the model learns to encode natural language into its own abstract internal language and decode it back. We make this process explicit with autoregressive discrete generation over an extended codebook, plus a budget-aware length regularizer that controls the realized compression ratio. The compressed latent $\{Z\}$ serves as a semantic scaffold, forcing the model to perform core reasoning on compact symbolic representations before projecting them back to surface tokens. Our design is more flexible, capable of dynamically lengthening compressed text based on semantic density, and seamlessly integrating with sliding windows to support longer contexts. We further analyze the contextual regularity of the learned Z-token codebook and show that repeated Z-tokens tend to occur in semantically related contexts, while still remaining context-dependent rather than fixed lexical units. We conducted extensive experiments on seven large-scale datasets—RULER, Wikipedia, CNN/Daily Mail, HotpotQA, NarrativeQA, QASPER, and QuALITY—covering text reconstruction, continuation, summarization, and long text question answering, demonstrating the effectiveness of our method.

2 Related Work

Context Compression refers to any mechanism that explicitly aims to reduce the number of tokens participating in subsequent computation. Some approaches follow *heuristic token reduction* strategies, removing or merging less informative tokens based on saliency or similarity. Token Merging Bolya et al. (2023); Liu et al. (2025); Zhong et al. (2025) aggregates similar tokens , while Token Pruning Kim et al. (2022); Zhong et al. (2025); Rao et al. (2021); Kwek & Yin (2025); Song et al. (2024) eliminates low-saliency tokens during inference. Beyond rule-based methods, *learning-based semantic compression* encodes contexts into compact representations that retain high-level meaning. ICAE Ge et al. (2024) and AutoCompress Chevalier et al. (2023) represent long sequences as latent vectors for efficient reconstruction or reasoning; other recent works similarly investigate learned context compression Snell et al. (2022); Zhang et al. (2024). A related branch, *prompt compression*, targets input instructions rather than the full context. Methods such as 500xCompressor Li et al. (2024b), Gist Tokens Mu et al. (2024) and LongLLMLingua Jiang et al. (2024) summarize long

prompts into shorter, reusable representations. Our method is conceptually aligned with *learning-based semantic compression* but differs in execution: rather than encoding context externally, we integrate compression into the LLM's reasoning process. This yields discrete internal symbols that form a self-expressive "semantic language," achieving adaptive, interpretable, and semantically faithful compression.

Architectural and Implicit Sparsity Methods represent an alternative line of research that, instead of explicitly reducing the number of tokens involved in computation, improve efficiency through architectural modifications or implicitly introduced sparsity in the computation process. Compressive Transformer Rae et al. (2019) introduces hierarchical memory buffers summarizing distant activations; Reformer Kitaev et al. (2020) leverages locality-sensitive hashing to approximate attention; Routing Transformer Roy et al. (2020) sparsifies attention through content-based routing. Other models such as Longformer, BigBird, and Window Attention Child et al. (2019); Beltagy et al. (2020); Zaheer et al. (2021); Yuan et al. (2025); Fang et al. (2024); Fedus et al. (2022); Dai et al. (2024) also achieve scalability via structured sparsity.

Discrete Semantic Representation provides a principled foundation for compact and interpretable encoding of high-level semantics. VQ-VAE van den Oord et al. (2018) introduced vector-quantized bottlenecks that map continuous signals into a learned codebook, later extended to sequence models Łukasz Kaiser & Bengio (2018); Łukasz Kaiser et al. (2018); Liu et al. (2024); Zhao et al. (2018); Hu et al. (2026) and cross-modal learning frameworks Liu et al. (2021). These approaches show that discrete latent variables can serve as symbolic carriers of information. Subsequent works explored training stability and optimization for discrete variables Jin et al. (2020). While prior works used discrete bottlenecks as auxiliary modules, we internalize this semantic layer within the LLM's reasoning flow, enabling discrete tokens to serve as learned, interpretable units for compressing and reasoning over long contexts.

3 METHODS

Our goal is to adapt a pretrained LLM into a compressor and decompressor. The compression and decompression work as follows: given an input sequence $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, the compressor generates a variable-length sequence of tokens $\mathbf{Z} = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_K\}$, where $K \ll N$. We consider two usage modes: 1). **Direct decompression for inference (Main Mode)** : the decompressor directly consumes Z-tokens and generates downstream task outputs, so the compressor acts as a learned prompt compressor; 2). **Z-space inference followed by reconstruction (Auxiliary Mode)**: the decompressor reconstructs surface text from Z-tokens, while a separate inferencer operates in the Z-token space before final decoding.

3.1 LLM COMPRESSOR

Given an input token sequence $\mathbf{X}$, the compressor generates a variable-length Z-token sequence $\mathbf{Z}$. To preserve the prior of the pretrained LLM, we extend the original vocabulary to $[\mathcal{V}_{\text{base}}, \mathcal{V}_z]$, where $\mathcal{V}_{\text{base}}$ is the original vocabulary and $\mathcal{V}_z$ is the extended vocabulary for Z-tokens. Z-tokens are generated only from $\mathcal{V}_z$. Unlike previous work (e.g., ICAE or AutoCompressor) that directly learns fixed-length soft embeddings, our compressor generates $\mathbf{Z}$ autoregressively:

$$p_\phi(\mathbf{Z} \mid \mathbf{X}) = \prod_{t=1}^{K} p_\phi(z_t \mid \mathbf{X}, z_1, \ldots, z_{t-1}). \tag{1}$$

The compressed length K is not fixed a priori but determined dynamically by the compressor itself. Concretely, we augment $\mathcal{V}_z$ with a special `[EOS-Z]` token; generation stops once the compressor emits `[EOS-Z]`, so K emerges from the model output rather than being imposed as a hyperparameter. To avoid degenerate solutions in which the model either collapses to a single Z-token or reproduces the input length, we regularize the realized compression ratio against a target ratio r:

$$\mathcal{L}_{\text{len}} = \left(\frac{K}{N} - \frac{1}{r}\right)^2 \tag{2}$$

$$z_t = \mathbf{GumbelSoftmax}(\mathbf{h}_t), \quad z_t \in \mathcal{V}_z \tag{3}$$

where N is the input length. This soft constraint allows informative segments to consume more Z-tokens while penalizing global budget violations, enabling content-adaptive allocation without imposing rigid per-segment quotas. This design can also be interpreted from a constrained information

allocation perspective. Suppose $\mathbf{X}$ is partitioned into local segments $\{\mathbf{X}^{(i)}\}_{i=1}^{M}$, and the compressor allocates K_i latent tokens to segment $\mathbf{X}^{(i)}$ under a global budget $\sum_i K_i \le N/r$. If the segment-wise information density is defined as

$$H_i \triangleq H(\mathbf{X}^{(i)} \mid \mathbf{X}^{(<i)}),$$

and the distortion for segment $\mathbf{X}^{(i)}$ takes the form $D_i(K_i) = H_i e^{-\beta K_i}$ with $\beta > 0$, then the optimal allocation satisfies

$$K_i^\star = \frac{1}{\beta}\big(\log(\beta H_i) - \log \lambda\big),$$

for some $\lambda > 0$, and is therefore monotonically increasing in H_i. In other words, under a fixed compression budget, semantically denser or less predictable segments should receive more Z-tokens, while more redundant segments should receive fewer. From this perspective, the `[EOS-Z]` mechanism together with $\mathcal{L}_{\text{len}}$ implements a soft rate–distortion trade-off. A complete derivation and discussion are provided in Appendix G. Therefore, the model can dynamically determine the lexical units needed to represent a given context, achieving variable-length compression in a way that remains compatible with the native generation process of LLMs. To ensure end-to-end training while maintaining discreteness, we use a Gumbel-Softmax estimator to sample Z-tokens only from $\mathcal{V}_z$. This produces a soft embedding e_{soft} during training. We then apply a straight-through estimator to combine differentiable gradients with discrete decoding:

$$e = e_{\text{hard}} + [e_{\text{soft}} - \mathbf{sg}(e_{\text{soft}})] \qquad (4) \qquad \mathcal{L}_{\text{overlap}} = 1 - \cos(\mathbf{Z}_i^{\text{overlap}}, \mathbf{Z}_{i+1}^{\text{overlap}}) \qquad (5)$$

where e_{hard} denotes the embedding obtained by argmax. This hybrid estimator maintains stable gradient flow without sacrificing the discreteness of the learned symbols.

Sliding window compression. For excessively long sequences, our method naturally adapts to sliding-window compression: the input sequence is segmented into overlapping windows of length W and stride S (typically $W = 1024$ and $S = 256$), each independently compressed into $\mathbf{Z}_i$. For adjacent segments, we enforce a consistency constraint on overlapping regions to ensure semantic coherence. We can then concatenate tokens of compressed segments $\{\mathbf{Z}_i\}$ for long-context reconstruction or inference.

3.2 Direct decompression for inference

The LLM decompressor serves as the core component for translating the compressed Z-tokens back to meaningful semantic outputs. Our decompressor supports two distinct usage paradigms: the inferencer and the reconstructor. For both paradigms, the decompression process is modeled as conditional autoregressive generation, with shared training objectives and constraints that ensure stability and semantic fidelity. Formally, the decompression process follows the distribution:

$$p_\theta(\mathbf{Y}|\mathbf{Z}) = \prod_{t=1}^{T} p_\theta(y_t \mid y_1 \dots y_{t-1}, \mathbf{Z}), \tag{6}$$

Depending on the usage paradigm, the sequence $\mathbf{Y}$ varies. For the reconstructor task, $\mathbf{Y}$ is the input sequence to the compressor $\mathbf{X}$. While for the inferencer task, such as QA, $\mathbf{Y}$ is the response to $\mathbf{X}$. $\mathbf{Z}$ represents the Z-tokens generated by the compressor. To ensure a linguistic foundation, the decompressor is trained under lexical constraints: in each decoding step, the output logit vector is restricted to within the pre-trained model's original vocabulary $\mathcal{V}_{\text{base}}$:

$$\tilde{p}_\theta(y_t|y_1 \dots y_{t-1}, \mathbf{Z}) = \textbf{Softmax}\left(W_{\text{base}}\mathbf{h}_t\right), \tag{7}$$

where $\mathbf{h}_t$ represents the decoder's hidden state at step t, and W_{base} is the embedding matrix of the original vocabulary set. This constraint forces the model to use human-understandable vocabulary to express internal abstract representations, effectively translating the private latent "language" back into natural language. Empirical evidence shows this vocabulary constraint improves training stability and ensures semantic fidelity across tasks. For training, the general form of the loss is a cross-entropy term that maximizes the likelihood of generating correct sequence given the Z-tokens:

$$\mathcal{L}_{\text{tr}} = -\sum_{t=1}^{T} \log p_\theta(y_t \mid y_1 \dots y_{t-1}, \mathbf{Z}), \tag{8}$$

Training entirely under teacher forcing introduces exposure bias, where the model learns to rely on ground-truth labels that are unavailable at inference time. To mitigate this mismatch, we adopt a

scheduled sampling strategy: at each decoding step, the lexical input is taken from the ground-truth token with probability $1 - p$, or from the model's own previous prediction with probability p. The probability p increases linearly during training, allowing the model to gradually transition from teacher-forced decoding to free decoding. This improves robustness to its own prediction errors and leads to better generalization at inference time. In addition to the token-level reconstruction objective, we introduce two regularization terms to stabilize discrete code learning and promote effective use of the Z-token vocabulary. The first term, $\mathcal{L}_{\text{com}}$, regularizes the consistency between the soft embedding selected by Gumbel-Softmax and its corresponding hard embedding:

$$\mathcal{L}_{\text{com}} = \mathbb{E}_t\Big[\big\|\mathbf{sg}(\mathbf{e}_t^{\text{soft}}) - \mathbf{e}_t^{\text{hard}}\big\|_2^2\Big] + \eta\,\mathbb{E}_t\Big[\big\|\mathbf{e}_t^{\text{soft}} - \mathbf{sg}(\mathbf{e}_t^{\text{hard}})\big\|_2^2\Big]. \tag{9}$$

Here, $\mathbf{sg}(\cdot)$ denotes the stop-gradient operator. This term stabilizes discrete code learning by penalizing the discrepancy between soft and hard embeddings. The second term, $\mathcal{L}_{\text{KL}}$, prevents codebook collapse by encouraging more balanced usage of the Z-token vocabulary:

$$\mathcal{L}_{\text{KL}} = \sum_{j=1}^{|\mathcal{V}_z|} \hat{\mathbf{q}}_j \Big(\log(\hat{\mathbf{q}}_j + \epsilon) - \log(u_j + \epsilon) \Big), \tag{10}$$

where $\hat{\mathbf{q}} = \frac{1}{M}\sum_{m=1}^{M} \tilde{\mathbf{q}}^{(m)}$ is the average soft assignment over a minibatch of size M, and u_j denotes the target uniform prior over $\mathcal{V}_z$. This regularizer encourages the compressor to use the extended codebook more evenly, thereby avoiding degenerate solutions in which only a small subset of Z-tokens is activated. The final training objective for the decompressor integrates the reconstruction loss with these regularization terms to ensure end-to-end consistency:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{tr}} + \lambda\mathcal{L}_{\text{KL}} + \beta\mathcal{L}_{\text{com}} + \gamma\mathcal{L}_{\text{len}}, \tag{11}$$

where λ, β, and γ control the strengths of the code-usage, embedding-consistency, and length regularizers, respectively.

3.3 Z-space inference followed by reconstruction

The decompressor as reconstructor paradigm positions the decompressor as a reconstructor of the input text to the compressor from Z-tokens, while a separate LLM operates in the Z-token space between compression and reconstruction to enable reasoning purely in the Z-Token space. This design decouples compression from high-level reasoning, allowing the inferencer LLM to focus on high-level semantics without handling redundant surface-level tokens. In this case, we first train the decompressor using Eqn. 11 with the target sequence $\mathbf{Y}$ as the compressor input $\mathbf{X}$. Then we compress both the prompt sequence $\mathbf{X}$ and the response sequence $\mathbf{Y}$ into Z-Token sequences $\mathbf{Z^p}$ and $\mathbf{Z^r}$ The separate LLM inferencer conducts autoregressive generation entirely on Z-tokens, treating them as an internal symbolic language, as:

$$p_\psi(\mathbf{Z^r} \mid \mathbf{Z^p}) = \prod_{t=1}^{|\mathbf{Z^r}|} p_\psi(\mathbf{Z}_t^{\mathbf{r}} \mid \mathbf{Z^p}, z_1^{\mathbf{r}} \dots z_{t-1}^{\mathbf{r}}), \tag{12}$$

where ψ represents the parameters of the inference LLM trained or adapted for symbolic reasoning. This allows operations such as summarization, continuation, and question answering to be performed in the latent space without returning to surface tokens. Essentially, this paradigm treats Z-tokens as a symbolic interface between compression and reasoning; the model no longer needs to reconstruct before thinking, but can think directly using its own abstract language. This provides a new window into how LLMs internally represent and manipulate semantic knowledge.

3.4 Contextual Regularity of Z-tokens

We observe that the learned codebook $\mathcal{V}_Z$ is reused in heterogeneous inputs: the same Z-tag index can appear in different contexts but yields different representations after decompression; conversely, the same input may have multiple valid Z sequences at different sampling temperatures. Unlike traditional tokenizers, where one token corresponds to one subword, this suggests that Z-tokens function as abstract, context-dependent semantic units, rather than deterministic mappings.

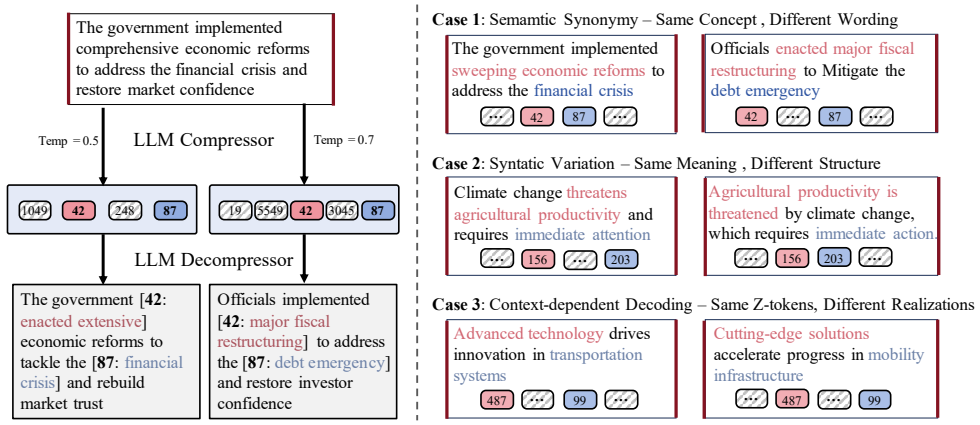

Figure 2: The figure shows the flexibility of Z-tokens. On the **left**, the same sentence can be compressed into different Z-token sequences while preserving meaning after decompression. On the **right**, the same Z-tokens yield different outputs in different contexts. This suggests that Z-tokens are context-dependent rather than chaotic.

Qualitative illustrations. In several cases, Z-tokens operate at a conceptual level rather than encoding fixed surface wording. 1) **Semantic synonymy:** sentences such as *"The government implemented sweeping economic reforms to address the financial crisis"* and *"Officials enacted major fiscal restructuring to mitigate the debt emergency"* share overlapping Z-tokens (e.g., z_{42}, z_{87}), suggesting a shared encoding of government-led economic intervention despite lexical variation in the decompressed outputs. 2) **Syntactic alternation:** *"Climate change threatens agricultural productivity and requires immediate attention"* and *"Agricultural productivity is threatened by climate change, which requires immediate action"* map to shared tokens (e.g., z_{156}, z_{203}), indicating that some codes may be relatively insensitive to surface word order while preserving higher-level semantic content. 3) **Domain-level variation:** *"Advanced technology drives innovation in transportation systems"* and *"Cutting-edge solutions accelerate progress in mobility infrastructure"* align on tokens such as z_{487}, z_{99}, again suggesting reuse at the level of abstract topical meaning rather than exact lexical realization.

Interpretation. These observations suggest that Z-tokens may function as reusable, underspecified semantic codes rather than fixed lexical symbols. In this view, compression abstracts away some surface variability, allowing multiple paraphrastic realizations to share parts of the same discrete representation. We stress that this is an empirical interpretation of the learned code behavior, not a claim that each Z-token carries an invariant standalone meaning.

Measuring contextual regularity. To assess whether this reuse is non-random, we measure the contextual consistency of each Z-token across its corpus occurrences. Let $\mathcal{O}(z_i) = \{o_1, \ldots, o_{N_i}\}$ denote the set of occurrences of z_i in a corpus. For each occurrence o_p, let $\mathbf{ctx}(o_p)$ be a fixed-width window of surface text around o_p, and let $\mathcal{F}(\cdot)$ be a frozen semantic encoder (e.g., BERT) that maps text to an embedding. We then define:

$$\mathcal{C}(z_i) = \frac{1}{\binom{N_i}{2}} \sum_{p<q} \mathbf{sim}(\mathcal{F}(\mathbf{ctx}(o_p)), \mathcal{F}(\mathbf{ctx}(o_q))) \,. \tag{13}$$

A high value of $\mathcal{C}(z_i)$ indicates that occurrences of the same Z-token tend to appear in semantically similar neighborhoods, suggesting structured reuse rather than arbitrary code assignment. On our evaluation set, the mean consistency is 0.75 ± 0.11, which supports the claim that Z-token reuse exhibits substantial contextual regularity. For more experimental results, please refer to 4.2. This context dependence naturally raises the question of how underspecified codes are resolved during decompression. Importantly, Z-token decoding is not performed in isolation. Instead, each output token is conditioned on three sources of information: the complete Z-sequence $Z = (z_1, \ldots, z_K)$, the autoregressive text prefix $x_{<t}$, and the base-vocabulary constraint $\mathcal{V}_{\text{base}}$. The full Z-sequence narrows broad semantic possibilities through neighboring codes; the generated prefix progressively commits the decoder to one semantic realization; and restricting logits to the pre-trained vocabulary anchors decoding in well-formed natural language. Together, these constraints convert underspecified latent codes into context-resolved surface outputs.

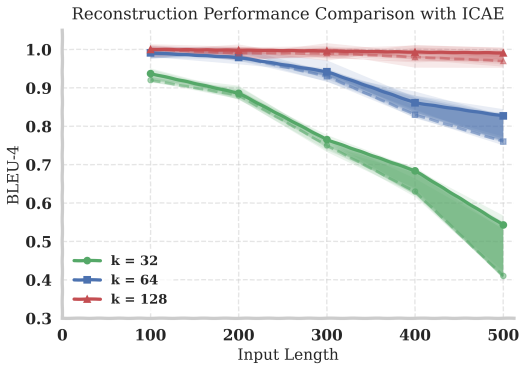

Figure 3: BLEU-4 comparison with ICAE under different input lengths.

Table 1: Comparison on Wikipedia reconstruction at compression ratios of 4× and 8×.

Method	4×		8×	
	BLEU↑	CE↓	BLEU↑	CE↓
AutoCompressor	96.44	0.29	92.67	0.38
Gist Token	96.29	0.21	94.63	0.30
500xCompressor	97.31	0.11	95.25	0.23
SAC	97.88	0.18	95.71	0.21
ICAE	98.45	0.04	93.44	0.17
Ours	**99.31**	**0.04**	**96.28**	**0.13**

4 EXPERIMENT

Baselines We conducted experiments on the Qwen3-0.6B, 4B and LLaMA3.1-8B. For the LoRA setting, we applied an adapter to all projection layers with $r = 128$, $\alpha = 256$, and dropout of 0.1. We compared our method with strong baselines across several paradigms, including ICAEGe et al. (2024), GistTokenMu et al. (2024), AutoCompressorChevalier et al. (2023), H2OZhang et al. (2023), SnapKVLi et al. (2024a), 500xCompressorLi et al. (2024b), LongLLMLinguaJiang et al. (2024), PyramidKVCai et al. (2025), SACLiu et al. (2026), and LLOCOTan et al. (2024). All methods used the same training data, backbone, and training configuration. Details of the experimental environment can be found in A.

Datasets and Metrics For reconstruction and continuation tasks, we use BLEU-4 and PPL as evaluation metrics. We report EM for QuALITY, F1 for HotpotQA, NarrativeQA, and QASPER, ROUGE scores for CNN/DailyMail, and average accuracy over 13 subtasks for RULER.

4.1 COMPARISON

We first evaluate the reconstruction performance to understand whether the Z-tokens contain sufficient semantic knowledge to recover the original context. We test on Wikipedia and report the BLEU-4 and CE-Loss results, as shown in Table 1. Our method achieves strong performance across different compression ratios. Due to autoregressive variable-length compression, the effective compression ratio is higher than the nominal ratio, reaching 5.1× at 4× compression and 9.4× at 8× compression. These results demonstrate that our method preserves semantic information while improving compression efficiency.

Figure 3 investigates the relationship between BLEU-4 and input length. Our method remains nearly unaffected at 4× compression and still achieves a BLEU-4 of 54.3 at 16× compression, demonstrating strong reconstruction capability. To evaluate generalization beyond reconstruction, following Ge et al. (2024), we use Qwen3-0.6B further conduct text continuation experiments, where the model predicts future tokens from compressed contexts. As shown in Table 2, our method maintains comparable language modeling ability across different context lengths; reducing the available context from 1024 to 128 only increases PPL by 0.44. These results demonstrate that Z-token representations preserve essential semantics and contextual dependencies for downstream prediction.

Table 2: Comparison of PPL in the continuation task.

Context	PPL (Ori)↓	PPL (Z)↓	Gap
1024 → 512	29.96	30.12	+0.16
1024 → 256	30.42	30.65	+0.23
1024 → 128	29.90	30.34	+0.44

To evaluate reconstruction performance on longer contexts, we use Llama-3.1-8B and select ICAE and 500xCompressor as baselines. We conduct experiments on the arXiv dataset, which contains longer documents and provides a more challenging setting than Wikipedia. To avoid cross-document interference, we select documents with original lengths of 4K, 8K, and 16K tokens, while excluding documents longer than 32K due to their limited availability. The results are shown in Table 3. The results show that our method maintains higher reconstruction fidelity even in longer contexts. To further evaluate its effectiveness in real-world long-context scenarios, we conducted experiments on RULER using Llama-3.1-8B. Unlike reconstruction, RULER evaluates whether the compressed

Table 3: Long-context reconstruction on arXiv.

Ratio	Method	4K	8K	16K
4×	ICAE	98.41	97.25	96.74
	500xCompressor	99.28	97.86	97.42
	Ours	**99.79**	**99.01**	**98.37**
8×	ICAE	92.03	90.74	88.53
	500xCompressor	93.24	92.13	91.52
	Ours	**95.82**	**94.79**	**93.44**
12×	ICAE	84.75	82.32	81.88
	500xCompressor	86.20	85.13	85.58
	Ours	**93.82**	**92.98**	**92.36**

Table 4: RULER results on Llama-3.1-8B.

Model	4K	8K	16K	32K
Llama3.1-8B	85.05	76.98	75.94	68.22
H2O	81.33	72.19	70.75	64.47
LongLLMLingua	77.20	63.45	58.78	43.84
SnapKV	80.65	73.44	71.14	64.09
PyramidKV	82.31	74.04	71.61	64.96
500xCompressor	81.72	73.25	69.61	63.41
SAC	83.46	74.16	72.01	64.29
Ours	**84.87**	**75.32**	**73.70**	**66.21**

representation can preserve task information relevant to downstream retrieval and inference. The results are shown in Table 4. At the same effective compression, our method retains 97% of full-attention RULER accuracy at 32K, the smallest degradation among all baselines, especially in longer contexts.

The summarization task evaluates the model's ability to abstract high-level content from compressed representations. We conducted experiments on the CNN/DailyMail using two paradigms: direct decompression and Z-token space inference. The results are shown in Table 5. For long-text QA, we evaluate our method on HotpotQA, QuALITY, NarrativeQA, and QASPER, using Qwen3-0.6B under 4× compression. As shown in Table 6, our method outperforms prior compression baselines. For sliding window, We conducted experiments on the QASPER and HotpotQA using Qwen3-0.6B. For detailed experimental results and setup, please refer to Appendi B.

Table 5: Performance comparison of summarization tasks on CNN/DailyMail.

Method	Rouge-1↑	Rouge-2↑	Rouge-L↑
Qwen3-0.6B	33.25	16.58	30.83
AutoCompressor	30.28	14.86	28.49
GistToken	29.11	14.57	27.62
ICAE	30.76	15.31	28.84
Ours (w/o infer)	**32.58**	**16.09**	29.89
Ours (w/ infer)	32.04	15.57	**30.15**

Table 6: Long-text QA results under 4× compression.

Methods	QuALITY	HotpotQA	NarrativeQA	QASPER
Qwen3-0.6B	38.48	32.18	16.32	18.44
AutoCompressor	34.43	27.59	14.41	16.47
500xCompresor	36.59	29.64	14.48	17.96
ICAE	34.68	29.65	14.88	17.62
LLOCO	37.33	30.46	15.29	17.79
Ours (w/o infer)	**38.25**	**31.28**	16.02	**18.11**
Ours (w/ infer)	38.04	30.46	**16.13**	17.66

4.2 Ablation

Effect of Z-token size We analyze the effect of Z-token vocabulary size using different settings from 1k to 10k. As shown in Table 7, a smaller vocabulary limits the capacity of the compressed representation and fails to preserve sufficient relational and contextual information. Performance improves as the vocabulary size increases, while the gains become marginal beyond 8k. This reveals a practical trade-off between representation capacity and computational cost. Since Wikipedia is highly structured, we additionally evaluate our method on the less structured HumanEval. Please refer Appendix E.

Table 7: Effect of different Z-token counts.

Dataset	1k	2k	4k	8k	10k
HotpotQA (F1)	16.23	22.45	27.96	**31.28**	30.27

Efficiency analysis Beyond preserving long-context reasoning ability, Z-token reduces inference cost by shortening the effective context length. With an effective compression ratio of $9.4\times$, we evaluate generation efficiency on Qwen3-4B using a single NVIDIA H200 GPU with bf16 precision, FlashAttention-2, and KV caching enabled. The output length is fixed to 64 tokens. Additional analysis of compression overhead, amortization, and serving scalability is provided in Appendix C. As shown in Table 8, Z-token consistently reduces inference cost across different context lengths. Compared with processing the original context, it reduces end-to-end latency by 18%, 41%, and 73%

Table 8: Inference efficiency comparison on Qwen3-4B. TTFT denotes time-to-first-token.

Context	Method	TTFT(s)	Latency(s)	Peak Mem(GB)	KV(GB)
10k	Baseline	0.27	1.75	9.6	1.38
	Z-token	0.03	1.48	7.8	0.15
20k	Baseline	0.68	2.13	11.8	2.76
	Z-token	0.06	1.51	8.0	0.30
30k	Baseline	1.26	2.71	13.9	4.13
	Z-token	0.08	1.57	8.2	0.45

Table 9: Correlation between allocated Z-token count and semantic-density proxies.

Dataset	Proxy	Pearson r	Spearman ρ
Wikipedia	Surprisal	0.54	0.57
HotpotQA	Surprisal	0.49	0.52
Wikipedia	Entity-rare density	0.36	0.39
HotpotQA	Entity-rare density	0.43	0.46

for 10k, 20k, and 30k token inputs, respectively, with larger gains at longer contexts due to shorter prefilling sequences. Moreover, Z-token substantially reduces KV-cache usage, lowering memory consumption from 1.38/2.76/4.13 GB to 0.15/0.30/0.45 GB for 10k/20k/30k contexts, enabling more efficient long-context serving.

Adaptive Z-token allocation. To examine whether variable-length compression reflects information density, we conduct a controlled analysis under the same $4\times$ compression setting. Using 10k fixed-length segments from Wikipedia and HotpotQA, we measure the correlation between allocated Z-token counts and two semantic-density proxies: surprisal and entity-rare density. Since all segments have identical lengths, the analysis is unaffected by length variation. As shown in Table 9, Z-token allocation positively correlates with both proxies across domains. Higher-surprisal segments consistently receive more Z-tokens, indicating that the compressor adaptively allocates more capacity to information-dense regions. Full details are provided in Appendix F.

(a)

Setting	BLEU (T/F)	$\mathbf{Z}_{\mathrm{Len}}$
Full loss	**99.41**/72.39	4.49k
w/o L_{KL}	87.27/52.51	1.45k
w/o L_{com}	94.97/61.26	4.06k
w/o L_{len}	99.13/**72.60**	10.51k

(b)

Setting	λ_{KL}	λ_{com}	λ_{len}	BLEU (T/F)	$\mathbf{Z}_{\mathrm{Len}}$
Default	0.3	0.4	0.3	**99.41**/72.39	4.49k
Variant 1	0.4	0.3	0.3	98.77/**72.62**	4.17k
Variant 2	0.3	0.3	0.4	98.05/70.48	3.94k
Uniform	0.33	0.33	0.33	99.18/71.95	4.21k

(c)

Method	F1↑	Similarity↑
w/o L_{overlap}	16.79	0.67
w/ L_{overlap}	**18.11**	**0.84**

Table 10: Loss ablations: (a) component ablation; (b) loss-weight sensitivity; and (c) overlap consistency ablation. BLEU is reported as Teacher/Free.

Loss Ablation We evaluate the loss design using 16k inputs with a $4\times$ compression ratio. As shown in Table 10(a), removing L_{len} significantly increases Z-token length with limited reconstruction gains, confirming its role in controlling the compression budget. Removing L_{KL} or L_{com} degrades reconstruction quality, while different loss weights cause only minor variations (Table 10(b)), demonstrating robustness. We further evaluate L_{overlap} on QASPER with a 1k sliding window. As shown in Table 10(c), removing it reduces consistency and downstream performance, validating its effectiveness.

Contextual Regularity Analysis To verify whether the contextual consistency score $C(z_i)$ in Section 3.4 is meaningful and not human-selected, we compare it with two randomized controls: **Shuffled**, where Z-tokens are randomly reassigned while preserving occurrence statistics, and **Random**, where Z-tags are randomly matched with contexts. As shown in Table 11, the original assignments achieve substantially higher consistency, indicating that Z-token reuse captures non-random semantic patterns.

Table 11: $C(z_i)$ under random controls.

Setting	$C(z_i)$
Original	0.75 ± 0.11
Shuffled	0.58 ± 0.07
Random	0.27 ± 0.05

5 Conclusion

This paper proposes a discretization method that integrates compression and generation into one LLM, enabling the model to naturally compress long texts into variable-length, controllable Z-tokens before downstream tasks. Extensive experiments on multiple tasks demonstrate that the method significantly reduces the latency and memory cost of downstream long-context inference while maintaining good task performance. These results show that allowing the LLM to infer compact latent language provides a promising direction for efficient long-context modeling. Future work could consider developing a global coordination mechanism across windows and a task-aware compression strategy to better preserve the structure required for specific downstream objectives.

AI USE STATEMENT

In writing this paper, we used generative AI tools to help polish the text, check grammar, improve the readability of technical explanations, and provide feedback on the organization of figures and experimental demonstrations. Furthermore, we utilized AI tools to assist discussion, explore different ways of expression, and enhance the clarity of the article. However, the core scientific contributions of this paper, including the proposed framework, theoretical motivation, mathematical derivation, experimental methods, and the implementation of the proposed methods, were not generated using any generative AI tools. All experimental results, analyses, and conclusions were independently verified by the authors. The authors bear full responsibility for all content in this paper, including the methodology, arguments, experimental results, and the correctness of the final manuscript.

REFERENCES

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020. URL https://arxiv.org/abs/2004.05150.

Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your vit but faster, 2023. URL https://arxiv.org/abs/2210.09461.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, et al. On the opportunities and risks of foundation models, 2022. URL https://arxiv.org/abs/2108.07258.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling, 2025. URL https://arxiv.org/abs/2406.02069.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts, 2023. URL https://arxiv.org/abs/2305.14788.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers, 2019. URL https://arxiv.org/abs/1904.10509.

Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models, 2024. URL https://arxiv.org/abs/2401.06066.

Grégoire Delétang, Anian Ruoss, Paul-Ambroise Duquenne, Elliot Catt, Tim Genewein, Christopher Mattern, Jordi Grau-Moya, Li Kevin Wenliang, Matthew Aitchison, Laurent Orseau, Marcus Hutter, and Joel Veness. Language modeling is compression, 2024. URL https://arxiv.org/abs/2309.10668.

Gongfan Fang, Hongxu Yin, Saurav Muralidharan, Greg Heinrich, Jeff Pool, Jan Kautz, Pavlo Molchanov, and Xinchao Wang. Maskllm: Learnable semi-structured sparsity for large language models, 2024. URL https://arxiv.org/abs/2409.17481.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2022. URL https://arxiv.org/abs/2101.03961.

Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model, 2024. URL https://arxiv.org/abs/2307.06945.

Yupeng Hu, Zixu Li, Zhiwei Chen, Qinlei Huang, Zhiheng Fu, Mingzhu Xu, and Liqiang Nie. Refine: Composed video retrieval via shared and differential semantics enhancement. *ACM Transactions on Multimedia Computing, Communications and Applications*, 2026.

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, 2024. URL https://arxiv.org/abs/2310.06839.

Shuning Jin, Sam Wiseman, Karl Stratos, and Karen Livescu. Discrete latent variable representations for low-resource text classification, 2020. URL https://arxiv.org/abs/2006.06226.

Sehoon Kim, Sheng Shen, David Thorsley, Amir Gholami, Woosuk Kwon, Joseph Hassoun, and Kurt Keutzer. Learned token pruning for transformers, 2022. URL https://arxiv.org/abs/2107.00910.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer, 2020. URL https://arxiv.org/abs/2001.04451.

Eugene Kwek and Wenpeng Yin. Compact: Common-token optimized model pruning across channels and tokens, 2025. URL https://arxiv.org/abs/2509.06836.

Yucheng Li, Bo Dong, Chenghua Lin, and Frank Guerin. Compressing context to enhance inference efficiency of large language models, 2023. URL https://arxiv.org/abs/2310.06201.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation, 2024a. URL https://arxiv.org/abs/2404.14469.

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

Alexander H. Liu, SouYoung Jin, Cheng-I Jeff Lai, Andrew Rouditchenko, Aude Oliva, and James Glass. Cross-modal discrete representation learning, 2021.

Qijiong Liu, Hengchang Hu, Jiahao Wu, Jieming Zhu, Min-Yen Kan, and Xiao-Ming Wu. Discrete semantic tokenization for deep ctr prediction, 2024.

Xin Liu, Jie Liu, Jie Tang, and Gangshan Wu. Catanet: Efficient content-aware token aggregation for lightweight image super-resolution, 2025.

Xin Liu, Runsong Zhao, Pengcheng Huang, Xinyu Liu, Junyi Xiao, Chunyang Xiao, Tong Xiao, Shengxiang Gao, Zhengtao Yu, and Jingbo Zhu. Autoencoding-free context compression for llms via contextual semantic anchors, 2026. URL https://arxiv.org/abs/2510.08907.

Jesse Mu, Xiang Lisa Li, and Noah Goodman. Learning to compress prompts with gist tokens, 2024. URL https://arxiv.org/abs/2304.08467.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling, 2019. URL https://arxiv.org/abs/1911.05507.

Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. Dynamicvit: Efficient vision transformers with dynamic token sparsification, 2021. URL https://arxiv.org/abs/2106.02034.

Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers, 2020. URL https://arxiv.org/abs/2003.05997.

Charlie Snell, Dan Klein, and Ruiqi Zhong. Learning by distilling context, 2022. URL https://arxiv.org/abs/2209.15189.

Woomin Song, Seunghyuk Oh, Sangwoo Mo, Jaehyung Kim, Sukmin Yun, Jung-Woo Ha, and Jinwoo Shin. Hierarchical context merging: Better long context understanding for pre-trained llms, 2024. URL https://arxiv.org/abs/2404.10308.

Sijun Tan, Xiuyu Li, Shishir Patil, Ziyang Wu, Tianjun Zhang, Kurt Keutzer, Joseph E. Gonzalez, and Raluca Ada Popa. Lloco: Learning long contexts offline, 2024. URL https://arxiv.org/abs/2404.07979.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023. URL https://arxiv.org/abs/2302.13971.

Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning, 2018. URL https://arxiv.org/abs/1711.00937.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models, 2022. URL https://arxiv.org/abs/2206.07682.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Y. X. Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention, 2025. URL https://arxiv.org/abs/2502.11089.

Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences, 2021. URL https://arxiv.org/abs/2007.14062.

Peitian Zhang, Zheng Liu, Shitao Xiao, Ninglu Shao, Qiwei Ye, and Zhicheng Dou. Long context compression with activation beacon, 2024. URL https://arxiv.org/abs/2401.03462.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2o: Heavy-hitter oracle for efficient generative inference of large language models, 2023. URL https://arxiv.org/abs/2306.14048.

Tiancheng Zhao, Kyusong Lee, and Maxine Eskenazi. Unsupervised discrete sentence representation learning for interpretable neural dialog generation, 2018.

Yiwu Zhong, Zhuoming Liu, Yin Li, and Liwei Wang. Aim: Adaptive inference of multi-modal llms via token merging and pruning, 2025. URL https://arxiv.org/abs/2412.03248.

Łukasz Kaiser and Samy Bengio. Discrete autoencoders for sequence models, 2018. URL https://arxiv.org/abs/1801.09797.

Łukasz Kaiser, Aurko Roy, Ashish Vaswani, Niki Parmar, Samy Bengio, Jakob Uszkoreit, and Noam Shazeer. Fast decoding in sequence models using discrete latent variables, 2018. URL https://arxiv.org/abs/1803.03382.

A IMPLEMENTATION DETAILS.

All experiments used 16 NVIDIA H200-GPUs. The software environment includes Ubuntu 22.04 LTS, Python 3.12, PyTorch 2.5.1, and CUDA 12.4. Unless otherwise specified, the base LLM used in the experiments is Qwen3-0.6B and LLama-3.1-8B-Base. We continuously pre-trained the base models using the prolong-64k and Wikipedia datasets. We split the data into two parts: 32k and 64k, where the 32k part contains 7B tokens and the 64k part contains 3B tokens.

B SLIDING WINDOW.

Sliding window We conducted experiments on the QASPER and HotpotQA using Qwen3-0.6B. We set the window size to 1024 and the stride to 256, enabling the model to process documents incrementally while maintaining semantic continuity. For comparison, we also evaluated a baseline configuration where the entire document is fed into the model at once, as a performance cap for the uncompressed case. Table12 shows that our method uses fewer tokens but achieves comparable performance. This demonstrates that our compression mechanism can be naturally combined with a sliding window strategy to handle inputs exceeding the maximum context window of an LLM.

Table 12: Sliding-window results.

Method	**QASPER↑**	**HotpotQA↑**
Qwen3-0.6B	18.44	32.18
ICAE	16.72	29.65
AutoCompressor	16.47	27.59
500xCompressor	17.46	29.64
Ours	**18.11**	**30.86**

C ADDITIONAL EFFICIENCY ANALYSIS

We provide a detailed analysis of the inference efficiency of Z-token contexts. All experiments are conducted on Qwen3-4B with a single NVIDIA H200 GPU using bf16 precision, FlashAttention-2, and KV caching enabled for both the baseline and our method. Generation length is fixed to 64 tokens, and latency measurements are averaged over five runs with CUDA-synchronized wall-clock timing. The baseline directly processes the original context of length N, while Z-token first compresses the document into a shorter latent sequence of length K and performs generation conditioned on the compressed context.

C.1 PER-QUERY GENERATION EFFICIENCY

Table 13 reports the per-query inference cost under an effective compression ratio of $9.4\times$. Since the compression process is performed once for each document, we separate the one-time compression cost from the repeated generation cost.

Table 13: Per-query inference efficiency on Qwen3-4B with KV caching enabled for all configurations. TTFT denotes time-to-first-token. Peak memory includes model weights (7.5 GB). Results are averaged over five runs.

N	Context	Tokens	TTFT (s)	Latency (s)	Tok/s	Peak Mem (GB)	KV (GB)
10k	Baseline	10,064	0.27	1.75	36.5	9.6	1.38
	Z-token	1,128	0.03	1.48	43.2	7.8	0.15
20k	Baseline	20,064	0.68	2.13	30.0	11.8	2.76
	Z-token	2,192	0.06	1.51	42.4	8.0	0.30
30k	Baseline	30,064	1.26	2.71	23.6	13.9	4.13
	Z-token	3,255	0.08	1.57	40.8	8.2	0.45

Z-token consistently reduces inference cost across all context lengths. The improvement becomes increasingly significant as the context grows because the dominant prefill computation scales with the input sequence length. Compared with the uncompressed baseline, Z-token reduces end-to-end latency by 18%, 41%, and 73% for 10k, 20k, and 30k-token contexts, respectively. The time-to-first-token is reduced by up to $16\times$, reflecting the reduced attention computation over the compressed sequence.

The KV-cache footprint is reduced proportionally to the compression ratio. For 10k/20k/30k contexts, the KV memory decreases from 1.38/2.76/4.13 GB to 0.15/0.30/0.45 GB, respectively. This reduction also lowers the peak GPU memory usage during generation, enabling substantially larger serving capacity under a fixed memory budget.

C.2 Compression Overhead and Amortization

Although Z-token reduces the repeated generation cost, compression introduces a one-time encoding overhead. We therefore explicitly measure the compression latency and compute the break-even point where the accumulated generation savings exceed the compression cost.

Table 14: Compression overhead and amortization analysis. Saving/query denotes the latency reduction compared with the baseline, and break-even indicates the number of queries required to offset the one-time compression cost.

N	K	Compression (s)	Saving/query (s)	Break-even
10k	1,064	25.1	0.27	92
20k	2,128	49.9	0.62	80
30k	3,191	74.6	1.14	65

The compression stage requires 25–75 seconds depending on the document length. This overhead is paid only once and becomes amortized when the same compressed representation is reused across multiple queries. For example, under a 30k-token context, the compression cost is recovered after 65 queries, after which each subsequent query achieves a $1.7\times$ latency improvement. Such amortization is particularly relevant for long-context applications such as retrieval-augmented generation, multi-turn assistants, and persistent agent memory, where the same document representation is repeatedly accessed.

C.3 Serving Capacity under Fixed KV Memory

Since KV cache is often the primary bottleneck for long-context serving, we further evaluate the concurrency improvement under a fixed KV-cache budget. We allocate approximately 11 GB KV memory and compare the maximum number of simultaneous requests that can be served.

Table 15: Batched serving efficiency under a fixed KV-cache budget ($\approx$11 GB). Both settings process comparable total prompt tokens.

Setting	Concurrency	KV (GB)	Per-request Latency (s)	Aggregate (tok/s)
Baseline	8	11.06	4.28	241
Z-token	71	10.99	3.87	1,959

Under the same KV-cache budget, Z-token increases the supported concurrency from 8 to 71 requests while maintaining comparable prompt computation (81k versus 76k processed tokens). The aggregate throughput improves from 241 to 1,959 output tokens/s, corresponding to an $8.1\times$ improvement. These results demonstrate that Z-token improves not only individual inference latency but also the scalability of long-context serving.

C.4 Efficiency with a Looser Compression Ratio

We additionally evaluate Z-token with a weaker compression setting (effective ratio $5.1\times$). As shown below, the efficiency improvement is consistent although smaller due to the larger retained context.

Overall, these experiments show that Z-token provides consistent inference benefits across different compression strengths. The improvement primarily comes from reducing the effective sequence length during attention and KV cache construction, while preserving the same downstream generation interface.

Table 16: Per-query efficiency with a looser compression ratio ($5.1\times$).

Context Length	Latency (s)	TTFT (s)	KV (GB)
10k	1.52	0.05	0.28
20k	1.55	0.10	0.55
30k	1.63	0.15	0.82

D SCHEDULED SAMPLING

We investigate the effect of scheduled sampling on free-decoding performance. Since ICAE is trained with teacher forcing, it suffers from exposure bias, where small prediction errors accumulate during autoregressive generation. To mitigate this issue, we gradually increase the sampling probability from 0 to 0.5 during training, while avoiding overly aggressive sampling that may slow convergence and reduce fluency. As shown in Table 17, scheduled sampling substantially improves free-decoding reconstruction and enhances robustness to autoregressive error accumulation.

Table 17: Free-decoding BLEU-4 on Wikipedia under different input lengths.

Method	100	200	300	400	500
ICAE Ge et al. (2024)	29.46	21.88	14.53	11.27	8.96
Ours	**58.24**	**52.23**	**46.66**	**41.84**	**37.66**

E GENERALIZATION

We further evaluate the generalization ability of our method on HumanEval, which contains relatively unstructured code-generation prompts and is therefore substantially different from the long-form text corpora used in pretraining. As shown in Table 18, under $4\times$ compression, AutoCompressor leads to a noticeable drop in pass@1 accuracy, whereas our method preserves performance more effectively and remains substantially closer to the base model. These results suggest that the learned compression mechanism generalizes beyond standard document reconstruction and can retain useful information even in less structured input domains.

Table 18: HumanEval results, reported with pass@1.

Dataset	**Base Model**	**AutoCompressor**	**Ours**
HumanEval	18.90	12.64	16.85

F ANALYZING VARIABLE-LENGTH Z-TOKEN ALLOCATION

A central question behind variable-length compression is whether the compressor allocates more Z-tokens to semantically denser segments, rather than producing lengths that are nearly uniform or driven by superficial statistics. To probe this behavior, we conduct a controlled diagnostic study calibrated to the main experimental regime, namely nominal $4\times$ compression. We emphasize that this analysis is intended as a targeted diagnostic of allocation behavior under the same compression regime as the main experiments, rather than as a definitive annotation-based measure of semantic density. We construct 10,000 diagnostic segments, including 5000 Wikipedia-style segments from reconstruction scenarios and 5000 HotpotQA-style segments from long-context question answering scenarios. Each segment contains exactly 256 input tokens. For each segment i, we record the allocated Z-token count k_i together with two segment-level proxies of semantic density:

- **Segment surprisal**: the mean token-level negative log-likelihood under the base model;
- **Entity-rare density**: the fraction of named entities and rare words in the segment.

We then report Pearson and Spearman correlations between each proxy and k_i, and additionally summarize the mean allocated Z-token count across surprisal quintiles. Because all segments have the same input length, the observed trends cannot be explained by trivial dependence on raw segment length. We adopt this controlled protocol to isolate allocation behavior under matched segment length and compression budget, which would otherwise be more difficult to disentangle from confounding factors in full end-to-end task traces.

Table 19: Correlation between allocated Z-token count and semantic-density proxies in the controlled diagnostic study.

Dataset	Proxy	Pearson r	Spearman ρ
Wikipedia	Segment surprisal	0.54	0.57
HotpotQA	Segment surprisal	0.49	0.52
Wikipedia	Entity-rare density	0.36	0.39
HotpotQA	Entity-rare density	0.43	0.46

Surprisal quintile	Mean allocated Z-tokens	Trend bar
Q1 (lowest)	37.1	
Q2	41.0	
Q3	46.4	
Q4	52.2	
Q5 (highest)	59.8	

Figure 4: Binned trend plot showing the relationship between surprisal quintile and allocated Z-token count. Higher-surprisal segments consistently receive more Z-tokens.

Across both domains, the allocated Z-token count is positively associated with multiple semantic-density proxies, and the surprisal-bucket trend is monotonic. These results provide supporting evidence that Z-token allocation is correlated with segment-level information density, rather than being approximately uniform across segments. At the same time, we do not claim that these proxies fully capture semantic complexity, nor that this controlled analysis replaces a fully annotated semantic-density evaluation. Instead, the results should be understood as evidence that the learned variable-length compressor behaves consistently with the intended allocation principle: segments that are harder to predict or contain denser factual content tend to receive a larger share of the latent Z-token budget.

G VARIABLE-LENGTH COMPRESSION.

Although our compressor is implemented autoregressively, its behavior can be understood from a simple constrained information allocation perspective. The key question is not merely how to compress an input sequence $X = \{x_1, \ldots, x_N\}$ into a shorter latent sequence $Z = \{z_1, \ldots, z_K\}$, but how to allocate a limited compression budget across parts of the input so that semantic distortion is minimized.

Consider partitioning X into M local segments,

$$X = \{X^{(1)}, X^{(2)}, \ldots, X^{(M)}\},$$

and suppose the compressor allocates K_i latent tokens to segment $X^{(i)}$, with

$$\sum_{i=1}^{M} K_i = K.$$

Let $D_i(K_i)$ denote the reconstruction or task-induced distortion contributed by segment $X^{(i)}$ when it is represented by K_i Z-tokens. A natural objective is then

$$\min_{\{K_i\}} \sum_{i=1}^{M} D_i(K_i) \quad \text{s.t.} \quad \sum_{i=1}^{M} K_i \leq \frac{N}{r},$$

where r is the target compression ratio. This is a standard budgeted allocation problem: the model must decide where the limited latent capacity should be spent.

To make this objective interpretable, we introduce a segment-wise semantic complexity measure

$$H_i \triangleq H\Big(X^{(i)} \mid X^{(<i)}\Big),$$

which can be viewed as the conditional uncertainty or information density of segment $X^{(i)}$ given its preceding context. Intuitively, segments with larger H_i are harder to predict from context and therefore require more latent capacity to preserve their semantics. We assume that the distortion for

each segment decreases with the number of allocated Z-tokens and that the reduction is larger for semantically denser segments. A simple form satisfying this intuition is

$$D_i(K_i) = H_i \exp(-\beta K_i),$$

where $\beta > 0$ controls how efficiently each additional Z-token reduces distortion.

Under this model, the constrained problem becomes

$$\min_{\{K_i\}} \sum_{i=1}^{M} H_i \exp(-\beta K_i) \quad \text{s.t.} \quad \sum_{i=1}^{M} K_i \le \frac{N}{r}.$$

Its Lagrangian is

$$\mathcal{L}(\{K_i\}, \lambda) = \sum_{i=1}^{M} H_i \exp(-\beta K_i) + \lambda \left(\sum_{i=1}^{M} K_i - \frac{N}{r} \right),$$

with $\lambda \ge 0$. Taking derivatives with respect to K_i gives

$$\frac{\partial \mathcal{L}}{\partial K_i} = -\beta H_i \exp(-\beta K_i) + \lambda.$$

At the optimum, for every active segment,

$$\beta H_i \exp(-\beta K_i^\star) = \lambda,$$

which yields

$$K_i^\star = \frac{1}{\beta} \left(\log(\beta H_i) - \log \lambda \right).$$

Therefore,

$$K_i^\star \text{ is monotonically increasing in } H_i.$$

This directly implies that, under a fixed global compression budget, semantically denser segments should receive more Z-tokens, while more predictable or redundant segments should receive fewer.

This view provides a theoretical interpretation of our design. The autoregressive compressor with the special token $[\text{EOS-}Z]$ determines the realized length K adaptively, while the length regularizer

$$L_{\text{len}} = \left(\frac{K}{N} - \frac{1}{r} \right)^2$$

acts as a soft relaxation of the hard budget constraint $\sum_i K_i \le N/r$. In other words, our training objective can be viewed as optimizing a soft rate–distortion trade-off: the model is encouraged to minimize downstream reconstruction or task loss while spending its latent Z-token budget where it reduces semantic distortion the most.

Importantly, this analysis does not claim that each Z-token corresponds to a fixed symbolic meaning, nor that the trained transformer exactly solves the above optimization. Rather, it explains why variable-length compression is the natural outcome of an adaptive latent bottleneck: if different parts of the input carry different amounts of conditional information, then an optimal compressor should not allocate the same number of latent tokens everywhere. Our method implements this principle in a discrete autoregressive form that is compatible with the native generation process of LLMs.

H DATASET STATISTICS.

Table 20 summarizes the main downstream long-context datasets used in our experiments. These benchmarks cover a variety of tasks, including abstract text summarization (CNN/DailyMail), multi-hop question answering (HotpotQA), long narrative comprehension (NarrativeQA), scientific document question answering (QASPER), and the more challenging long-context multiple-choice reading comprehension (QUALITY). The reported dataset sizes follow standard publicly available partitions.

During the reconstruction and continuation pre-training phases, we used a large subset of Wikipedia containing approximately 300,000 full-text articles randomly selected from various domains. To construct long-context training instances, each article was divided into blocks of 1024 units in length, with 256 overlapping terms between adjacent blocks within the same document to maintain local semantic continuity. In practice, each article generated an average of about 3 to 6 training segments, totaling approximately 1.45 million segments.

Table 20: Statistics of the downstream long-context datasets used in our experiments. The reported sizes follow standard public splits; exact numbers may vary slightly depending on preprocessing.

Dataset	Task	Train	Valid	Test	Output
CNN/DailyMail	Summarization	287,227	13,368	11,490	Summary
HotpotQA	Multi-hop QA	90,447	7,405	7,405	Answer
NarrativeQA	Long-form QA	32,747	3,461	10,557	Answer
QASPER	Scientific QA	2,593	1,006	1,451	Answer
QuALITY	Multiple-choice QA	265	265	265	Option

I Training and Inference Pipeline

Algorithm 1 Training the Z-token compressor, decompressor, and inferencer

Require: Pretrained LLM backbone f_Θ; base vocabulary V_{base}; Z-token vocabulary V_Z; training examples (X, Y); target compression ratio r.

Initialize three LoRA adapter sets: compressor adapter Δ_ϕ, decompressor adapter Δ_θ, and inferencer adapter Δ_ψ.

Extend the output vocabulary with V_Z for compressor and inferencer generation.

for each minibatch **do**

Compression: activate Δ_ϕ and generate a variable-length Z-token sequence

$$Z = (z_1, \ldots, z_K), \quad z_t \in V_Z,$$

by autoregressive decoding from $p_\phi(Z \mid X)$ until `[EOS-Z]` is emitted.

Use Gumbel-Softmax with a straight-through estimator to obtain differentiable Z-token embeddings during training.

Compute the length regularization term

$$\mathcal{L}_{\text{len}} = \left(\frac{K}{|X|} - \frac{1}{r} \right)^2.$$

Decompression / reconstruction: activate Δ_θ and decode from Z to the target natural-language sequence.

if reconstruction or continuation training **then**

Set the target sequence to the original or continued text and compute

$$\mathcal{L}_{\text{tr}} = -\sum_t \log p_\theta(y_t \mid y_{<t}, Z).$$

else if downstream supervised training **then**

Set the target sequence to the task output Y and compute the same conditional generation loss.

end if

Add codebook usage and commitment regularizers $\mathcal{L}_{\text{KL}}$ and $\mathcal{L}_{\text{com}}$.

Update Δ_ϕ and Δ_θ using

$$\mathcal{L}_{\text{comp-dec}} = \mathcal{L}_{\text{tr}} + \lambda \mathcal{L}_{\text{KL}} + \beta \mathcal{L}_{\text{com}} + \gamma \mathcal{L}_{\text{len}}.$$

Optional Z-space inference training: compress both the prompt/input X and the target response Y into Z^p and Z^r using Δ_ϕ.

Activate Δ_ψ and train the inferencer to predict the response codes:

$$\mathcal{L}_{\text{infer}} = -\sum_t \log p_\psi(z_t^r \mid Z^p, z_{<t}^r).$$

end for

Algorithm 2 Inference with adapter switching

Require: Input context or prompt X; pretrained LLM backbone f_Θ; LoRA adapters Δ_ϕ, Δ_θ, Δ_ψ.

1: **Step 1: Compress.** Activate compressor adapter Δ_ϕ and generate

$$Z^p \sim p_\phi(Z \mid X), \quad Z^p \subset V_Z.$$

2: **Mode A: Direct decompression / direct task inference.**

3: Activate decompressor adapter Δ_θ and generate the output in the base vocabulary:

$$\hat{Y} \sim p_\theta(Y \mid Z^p), \quad \hat{Y} \subset V_{\text{base}}.$$

4: **Mode B: Z-space autoregressive inference.**

5: Activate inferencer adapter Δ_ψ and generate response Z-tokens:

$$\hat{Z}^r \sim p_\psi(Z^r \mid Z^p), \quad \hat{Z}^r \subset V_Z.$$

6: Activate decompressor adapter Δ_θ and decode the response codes into natural language:

$$\hat{Y} \sim p_\theta(Y \mid \hat{Z}^r), \quad \hat{Y} \subset V_{\text{base}}.$$

7: **return** $\hat{Y}$.

All three functional roles are implemented by the same pretrained LLM backbone with different LoRA adapters. The compressor, decompressor, and Z-space inferencer have their own adapter parameters, denoted by Δ_ϕ, Δ_θ, and Δ_ψ, respectively. At inference time, the backbone parameters are shared and frozen, and the model switches between these LoRA adapters depending on the current stage. This design avoids maintaining three independent full LLMs while still allowing each role to specialize in a different mapping: natural language to Z-tokens, Z-tokens to natural language, and Z-tokens to Z-tokens.

J Practical Efficiency Analysis

A natural question is whether context compression remains useful as model size increases. In larger models, a greater fraction of runtime may shift from attention to MLP computation, which could potentially weaken the benefit of shortening the input context. To examine this issue, we report the Attention/MLP time ratio for Qwen3-4B and Qwen3-8B at different input lengths in Table 21. The results show that, although MLP computation becomes increasingly important in larger models, attention still occupies a substantial fraction of runtime and grows consistently with sequence length. In particular, as the input length increases from 8×256 to 8×1024, the attention ratio rises markedly for both Qwen3-4B and Qwen3-8B. This indicates that shortening the effective context remains beneficial even at larger model scales, and therefore the proposed compression mechanism retains its practical value beyond small or medium-sized LLMs.

Table 21: Attention/MLP time ratio at different input lengths for Qwen3-4B and 8B. Higher attention ratios at longer inputs indicate that reducing effective context length remains beneficial even for larger models.

Model	Input	Attn / MLP (%)
Qwen3-8B	$8 \times 256/512/1024$	43.8/47.9/54.4
Qwen3-4B	$8 \times 256/512/1024$	47.7/52.9/60.4