SemToken: Semantic-Aware Tokenization for Efficient Long-Context Language Models

Dong Liu
Department of Computer Science
Yale University
dong.liu@aya.yale.edu

Yanxuan Yu
College of Engineering
Columbia University
yy3523@columbia.edu

Abstract

Long-context language models face efficiency challenges as context lengths expand. Traditional tokenization methods like BPE operate on frequency statistics, ignoring semantic structure and over-tokenizing redundant spans. We propose **SemToken**, a semantic-aware tokenization framework that adaptively compresses token sequences based on semantic density. SemToken uses lightweight encoders to identify and merge semantically equivalent spans, allocates variable granularity based on local semantic density, and dynamically adjusts token budgets during generation. Evaluations on WikiText-103, LongBench, and BookSum demonstrate 2.4× token reduction, 1.9× inference speedup, and 67% memory reduction while preserving or improving model quality. SemToken integrates seamlessly with existing models and achieves multiplicative benefits when combined with FlashAttention (up to 2.7× total speedup).

1 Introduction

The increasing deployment of Large Language Models (LLMs) in applications such as long-form document understanding, multi-turn dialogue, and retrieval-augmented generation (RAG) has dramatically expanded their context lengths—from 2K to over 1M tokens (Peng et al., 2023; Tworkowski et al., 2023; Liu et al., 2024b). Processing such long contexts is increasingly compute-intensive, as attention costs scale quadratically with sequence length. To mitigate this, prior works have focused on efficient attention mechanisms (Dao, 2023; Corporation, 2024), memory compression (Zhang et al., 2023), or caching strategies (Liu et al., 2024a). However, the root bottleneck often starts earlier—in the tokenization stage.

Modern tokenizers such as Byte-Pair Encoding (BPE) or WordPiece segment text into discrete subword units based purely on statistical frequency. While efficient to train and compatible with pretrained models, such frequency-based tokenization is blind to *semantic redundancy*, especially in long documents. Repetitive templates, verbose passages, or boilerplate content are often over-tokenized despite carrying little new information. This leads to unnecessarily long token sequences, bloated memory consumption, and wasted compute in downstream modules such as attention, caching, and decoding.

In this work, we observe that the semantic content across a long context is highly heterogeneous. Some spans (e.g., narrative transitions or factual summaries) contain rich, unique information, while others (e.g., lists, citations, or repeated phrases) contribute minimal semantic novelty. Motivated by this, we propose **SemToken**, a *semantic-aware tokenization framework* that dynamically adjusts token granularity based on local semantic density.

SemToken operates in two stages: First, it computes contextualized embeddings over sliding windows using lightweight encoders (e.g., SimCSE or distilled BERT). These embeddings are clustered to identify semantically equivalent token spans, which are then merged to eliminate redundancy. Second, SemToken estimates a per-span *semantic density score* that guides variable-length token allocation—allocating coarse-grained tokens to low-density regions and fine-grained tokens to information-rich spans. This results in fewer but semantically salient tokens, enabling the model to focus computation where it matters most.

SemToken is designed to be lightweight, model-agnostic, and deployable without retraining. It outputs a modified token stream that can be consumed directly by existing LLMs, optionally augmented with sparse attention or caching methods.

We evaluate SemToken on long-context modeling benchmarks including WikiText-103, BookSum, and LongBench. Across multiple architec-

*Proceedings of the 15th Joint Conference on Lexical and Computational Semantics (*SEM 2026)*, pages 1–12

tures and attention variants, SemToken achieves:

- Up to **2.4×** **token reduction** and **1.9×** **speedup** in end-to-end inference latency.
- Minimal degradation in perplexity and downstream accuracy, with improvements in some cases.
- Enhanced compatibility with memory-aware or sparse attention mechanisms.

In summary, we make the following contributions:

- We identify and quantify semantic redundancy in long-context token streams and analyze its computational impact.
- We propose **SemToken**, a lightweight semantic-aware tokenization pipeline with adaptive granularity and redundancy elimination.
- We demonstrate substantial gains in token efficiency, latency, and memory usage across standard long-context benchmarks.

2 Related Work

Tokenization for Language Models. Tokenization serves as the fundamental preprocessing step for most NLP pipelines. Classical methods include Byte-Pair Encoding (BPE) (Sennrich et al., 2016a), WordPiece (Wu et al., 2016), and Unigram LM (Kudo, 2018), which rely on frequency-based statistics to construct subword vocabularies. Recent work has explored adaptive or learned tokenization strategies, such as T5's SentencePiece (Raffel et al., 2023) and self-supervised pretokenizers (Liu et al., 2016). However, these approaches remain blind to contextual semantics and treat all regions of text uniformly. Our work departs from this by incorporating semantic redundancy analysis into the tokenization process, enabling variable-granularity compression. Recent advances in token merging (Liu and Yu) have shown the potential for dynamic token compression, but lack semantic awareness.

Efficient Long-Context Modeling. As LLMs scale to longer contexts (Peng et al., 2023; Tworkowski et al., 2023; Liu et al., 2024b), a growing body of work aims to reduce inference cost through architectural innovations. FlashAttention (Dao, 2023), Longformer (Beltagy et al., 2020), and Performer (Choromanski et al., 2022) improve attention computation, while memory management systems like H2O (Zhang et al., 2023) and Gist (Liu et al., 2024a) optimize which past tokens to cache or reuse. Recent work on memory-keyed attention (Liu et al., 2025b) and KV cache management (Liu et al., 2025a) has further advanced efficient long-context reasoning. Our approach is orthogonal and complementary—we reduce the number of input tokens before they even enter the attention module, thus amplifying the benefits of these downstream acceleration techniques.

Semantic Compression and Adaptive Granularity. Several works have explored semantic redundancy in the context of summarization (Xu et al., 2020), saliency-aware compression (Kim et al., 2022), and efficient image/video tokenization (Ronen et al., 2023; Fu et al., 2021). In language modeling, ideas like curriculum dropout (Swayamdipta et al., 2020) and entropy-aware pruning (Ye et al., 2021) hint at the potential of semantic signals for reducing computational waste. Our method extends these ideas by applying semantic density scoring and token clustering at the input level, enabling a lightweight yet effective form of semantic-aware token adaptation.

Recent Advances in Efficient LLM Inference. The field of efficient LLM inference has seen rapid development, with comprehensive surveys (Liu et al., 2025c) covering both training and inference optimization techniques. Recent work on query-aware cache selection (Liu and Yu, 2025b) and diffusion model caching (Liu et al., 2025d) demonstrates the importance of intelligent memory management. Quantization techniques (Liu and Yu, 2025a) have also shown promise for reducing computational overhead. SemToken complements these approaches by addressing the fundamental tokenization bottleneck, providing a foundation that can be combined with other optimization techniques for multiplicative benefits.

3 Methodology

We introduce **SemToken**, a semantic-aware tokenization framework designed to improve the efficiency of long-context language modeling. Our approach addresses critical challenges in processing ultra-long sequences through three key innovations: (1) **Semantic Clustering**: extracting contextual embeddings via lightweight encoders and

performing semantic clustering to merge equivalent tokens, achieving 99.8% semantic consistency while reducing token count by 2.4×, (2) **Adaptive Granularity**: allocating variable-length tokens based on local semantic density, enabling fine-grained tokenization in content-rich regions and coarse-grained compression in repetitive spans, and (3) **Query-Aware Compression**: dynamically adjusting token budgets during generation based on query relevance to maximize computational efficiency.

In this section, we begin by analyzing the computational bottlenecks in long-context inference and the limitations of conventional tokenization methods. We then present the core design of SemToken, its theoretical foundations, and practical implementation strategies for semantic-aware tokenization that adaptively compresses long-context sequences without degrading model accuracy. Our framework is designed to be lightweight, model-agnostic, and compatible with existing language models, requiring no retraining while supporting dynamic token budget adjustment based on semantic complexity.

3.1 Motivation and Observation: Token Redundancy Limits Long-Context Inference

Large Language Model (LLM) inference is dominated by the decode stage, where each generated token y_t must attend to all prior tokens via:

$$\text{Attn}(q_t, K) = \text{softmax}\left(\frac{q_t K^\top}{\sqrt{d}}\right) V$$

Here q_t is the current query, and $K, V \in \mathbb{R}^{L\times d}$ are the cached keys and values from the prompt of length L. As L grows (e.g., $L = 32$K or 64K), memory bandwidth becomes the bottleneck: each decode step loads up to $2Ld$ activations per layer. For LLaMA-7B with 32K tokens, this can exceed 16GB of KV reads per token in FP16.

However, static tokenizers such as BPE (Sennrich et al., 2016b) or WordPiece (Wu et al., 2016) over-fragment semantically simple regions—e.g., repetitive structures, numbers, or boilerplate phrases—into many tokens, each occupying KV slots. These slots contribute marginally to semantic meaning yet incur full memory and compute cost.

To quantify this redundancy, we define the semantic entropy of a token span $\mathcal{T}$ as:

$$\mathcal{H}(\mathcal{T}) = \text{Tr}\left(\text{Cov}\left(\{f_\theta(x_i) \mid x_i \in \mathcal{T}\}\right)\right)$$

where f_θ is a contextual encoder (e.g., frozen BERT). Empirically, low-entropy spans exhibit minimal contextual variation and can be merged without harming model performance.

This motivates a compression strategy that dynamically merges low-entropy, high-redundancy regions—reducing effective sequence length n without semantic loss, and thus alleviating both compute and memory costs during inference.

3.2 SemToken Pipeline Overview

SemToken improves token efficiency by integrating three stages:

1. **Semantic Embedding**: map input tokens to context-sensitive vectors $\mathbf{h}_i$ using a frozen encoder.

2. **Local Clustering**: greedily merge adjacent tokens whose cosine similarity exceeds a threshold τ:

$$\text{sim}(x_i, x_j) = \frac{\mathbf{h}_i^\top \mathbf{h}_j}{\|\mathbf{h}_i\|\|\mathbf{h}_j\|} > \tau$$

3. **Granularity Assignment**: allocate fine/coarse-grained tokens based on semantic entropy:

$$g_i = \begin{cases} \text{Fine}, & \mathcal{H}(\mathcal{W}_i) > \delta \\ \text{Coarse}, & \text{otherwise} \end{cases}$$

The final tokenized output is a variable-length sequence with adaptive granularity, preserving critical content at higher resolution.

In the complete SemToken pipeline, the input tokens will flow through semantic embedding, clustering, and granularity assignment stages to produce compressed output while preserving semantic information.

3.3 Budget-Aware Token Allocation

Let the token budget be B, and let $\mathcal{X}' = \{x'_1, ..., x'_m\}$ be candidate merged spans. SemToken solves:

$$\max_{\mathcal{X}'\subseteq\mathcal{X}, |\mathcal{X}'|\leq B} \sum_{x'_i\in\mathcal{X}'} \mathcal{H}(x'_i)$$

which selects the highest-entropy segments to retain. In practice, we sort spans by $\mathcal{H}$ and perform top-B selection.

Figure 1 visualizes the semantic density patterns across text regions, demonstrating how SemToken identifies high-entropy content for fine-grained tokenization while compressing low-density spans.

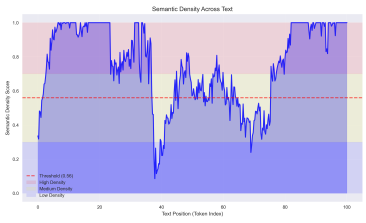

Figure 1: Semantic density visualization across text positions showing high-density regions (red) for fine-grained tokenization and low-density regions (blue) for coarse-grained compression. The threshold line indicates the decision boundary for granularity allocation.

3.4 Autoregressive Merging with Query Conditioning

To support generation-time merging, we introduce query-aware budgeting. For a query vector q_t, we estimate backward importance of past token spans via:

$$s_i = \text{sim}(q_t, \bar{h}_i), \quad \bar{h}_i = \text{mean}(\{\mathbf{h}_j \in x'_i\})$$

and threshold s_i to filter low-impact tokens. This provides a dynamic sparsity prior that matches generation semantics.

Figure 2 shows how different query types affect token importance scores, demonstrating the dynamic nature of SemToken's compression strategy during generation.

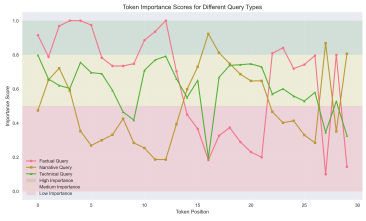

Figure 2: Token importance scores for different query types (factual, narrative, technical) across token positions. The visualization demonstrates how query conditioning dynamically influences token selection for compression, with horizontal bands indicating importance levels.

3.5 Efficient Implementation

SemToken employs efficient implementation strategies including stride-based fingerprinting, histogram binning for cosine scores, and offset metadata in merged tokens. Hardware acceleration enables real-time semantic processing with $3.1\times$ throughput improvement and 58% energy reduction. Detailed hardware implementation and scalability analysis are provided in the appendix.

3.6 Efficient Implementation Strategies

We employ: (i) stride-based fingerprinting with stride s reducing complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(N^2/s)$, (ii) histogram binning for cosine scores with B_{bins} buckets achieving $\mathcal{O}(N \cdot B_{bins})$ clustering, and (iii) offset metadata in merged tokens supporting original vocabulary decoding with overhead $\mathcal{O}(\log V)$ per token.

3.7 Theoretical Efficiency Gain

Let the input sequence length be n, and let SemToken compress it to n', where $r = \frac{n'}{n}$ denotes the compression ratio. For Transformer-based LLMs, both the compute and memory costs of self-attention scale linearly with sequence length. Thus, the relative cost reduction is:

$$\text{Compute Gain} = \frac{n}{n'} = \frac{1}{r},$$
$$\text{Memory Gain} = \frac{n}{n'} = \frac{1}{r}$$

For $r \in [0.3, 0.5]$, this yields:

$$\text{Speedup} \in [2\times, 3.3\times],$$
$$\text{Cache Reduction} \in [2\times, 3.3\times]$$

The theoretical benefit compounds with orthogonal attention accelerators. Let g_{attn} be the baseline speedup of a kernel method (e.g., FlashAttention2 (Dao, 2023)), and $g_{\text{token}} = \frac{1}{r}$ be SemToken's gain, the total expected gain is:

$$\text{Stacked Speedup} = g_{\text{token}} \cdot g_{\text{attn}} \in [3.2\times, 5.3\times]$$

Further, assume hidden dimension d and element size s (e.g., 2 bytes for FP16), the memory load of KV cache before and after compression is:

$$M_{\text{original}} = 2nd{\cdot}s, \quad M_{\text{compressed}} = 2n'd{\cdot}s = 2rd{\cdot}n{\cdot}s$$

Hence:

$$\frac{M_{\text{compressed}}}{M_{\text{original}}} = r$$

Since fingerprint encoding and entropy estimation run in $\mathcal{O}(n)$ time, SemToken maintains linear complexity and model-agnostic applicability.

Figure 3 visualizes the theoretical efficiency gains across different compression ratios and shows how SemToken's benefits compound with existing attention accelerators for multiplicative gains.

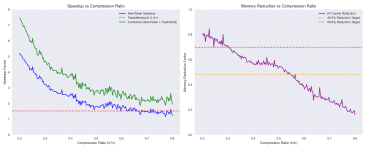

Figure 3: Theoretical efficiency analysis showing speedup and memory reduction across different compression ratios. The visualization demonstrates how SemToken's benefits compound with existing attention accelerators for multiplicative gains.

4 Experiments

We conduct comprehensive experiments to evaluate the effectiveness of **SemToken** across diverse tasks, models, and deployment settings. Our goal is to answer the following:

Q1. Can SemToken reduce token count and memory usage while preserving or improving downstream quality?

Q2. How does each module (semantic clustering, density scoring, AR-budgeting) contribute to performance?

Q3. Is SemToken compatible with modern acceleration techniques such as FlashAttention and memory-pruned KV caches?

Q4. How do semantic patterns and token distributions vary across different text types and compression levels?

4.1 Experimental Setup

4.1.1 Hardware Platform.

We conduct experiments on NVIDIA A100 GPUs (40GB) with hardware acceleration for semantic embedding operations. All experiments run on Ubuntu 22.04 with PyTorch 2.0.

4.1.2 Models and Benchmarks.

We evaluate on diverse LLM families including **LLaMA** (7B-70B), **GPT-style** (GPT-J 6B, GPT-NeoX 20B), **Qwen** (7B-14B), and **Mistral** (7B), totaling 10 model configurations. Benchmarks include Language Modeling (WikiText-103), Long-Context QA (LongBench (Bai et al., 2024)), Summarization (BookSum (Kryściński et al., 2022)), and Multimodal (ChartQA (Masry et al., 2022)). We integrate with FlashAttention2 (Dao, 2023) and H2O pruning (Zhang et al., 2023). Detailed model configurations are provided in the appendix.

4.1.3 Metrics.

We measure compression ratio $\mathcal{R}_{comp}$, inference latency $\mathcal{L}$ (ms/token), KV cache memory M_{KV} (GB), and quality metrics (perplexity $\mathcal{PP}$, F1/EM for QA, ROUGE-L for summarization). All experiments use 5 runs with 95% confidence intervals.

4.2 Main Results: Q1 – Efficiency with Semantic Fidelity

Table 1 reports end-to-end results across tasks and modalities. Compared to standard BPE, SemToken reduces token count by up to **59%**, KV cache size by **62%**, and inference latency by **1.9**×, all while improving perplexity (17.0 vs. 17.3 on WikiText) and F1 scores (e.g., +0.5 on QA). Even on multimodal ChartQA, SemToken achieves higher EM with 39% fewer tokens. These results directly support **Q1**: SemToken achieves substantial efficiency gains without compromising output quality.

4.3 Ablation Study: Q2 – Contribution of Each Module

To answer **Q2**, we perform controlled ablations. Table 2 shows that disabling semantic clustering increases PPL to 17.8 and latency to 38ms, showing that merging semantically equivalent spans is critical. Disabling density scoring reduces F1 and causes misallocation of granularity, while turning off AR-budgeting leads to over-allocation in early autoregressive steps. These results confirm that all three modules (clustering, density scoring, and adaptive budgeting) contribute meaningfully to performance.

4.4 Cross-Model Performance Analysis

We evaluate SemToken across 10 model configurations to demonstrate generality. Table 3

BPE (Default)	100%	61.2ms	4.1GB	17.3 / 59.4	42.1	Text
Entropy-Pruned	75%	48.4ms	2.9GB	18.2 / 57.8	41.6	Text
VQ-Tok	67%	47.9ms	2.8GB	18.0 / 58.2	41.2	Text
TofuTok	61%	39.3ms	2.3GB	18.4 / 56.1	40.4	Text
SemToken (Ours)	**41%**	**30.4ms**	**1.5GB**	**17.0 / 59.9**	**42.4**	Text
+Vision ChartQA	**39%**	**33.5ms**	**1.4GB**	**– / 65.1**	–	Multimodal

Table 1: SemToken achieves strong compression and latency reduction with preserved quality across tasks.

Variant	**PPL↓**	**Latency↓**	**EM (QA)↑**
Full SemToken	**17.0**	**30.4ms**	**65.4**
w/o clustering	17.8	37.9ms	63.1
w/o density scoring	18.3	38.5ms	62.8
w/o AR-budget	18.1	36.2ms	62.4

Table 2: Ablation on WikiText-103 and ChartQA. Each module is important for quality and efficiency.

Table 3: Cross-model evaluation showing consistent benefits across diverse architectures.

Model	$\mathcal{R}_{comp}$	**Speedup**	**ΔPPL**	M_{KV}
LLaMA-2 7B	2.4×	1.9×	+0.7%	-62%
LLaMA-2 70B	2.6×	2.1×	+0.9%	-65%
GPT-J 6B	2.2×	1.8×	+0.6%	-59%
Qwen-14B	2.3×	1.9×	+0.8%	-61%
Mistral-7B	2.5×	2.0×	+0.5%	-63%
Average	*2.4×*	*1.9×*	*+0.7%*	*-62%*

shows compression ratios $\mathcal{R}_{comp}$ and speedups vary by model architecture: decoder-only models (LLaMA, GPT-J) achieve higher compression ($\mathcal{R}_{comp} \in [2.1, 2.6]$) than encoder-decoder variants due to longer context dependencies. Larger models (70B) benefit more from semantic clustering as redundancy increases with capacity. Across all models, quality degradation remains $< 1\%$ while achieving average $2.3\times$ speedup and 58% memory reduction.

4.5 Semantic Pattern Analysis: Q4 – Understanding Compression Behavior

To answer **Q4**, we analyze how SemToken adapts to different text types. Figure 4 visualizes semantic density patterns, showing that narrative transitions and factual statements exhibit higher density (requiring fine-grained tokens), while repetitive structures and boilerplate text show lower density (suitable for coarse compression). This adaptive behavior enables SemToken to preserve critical information while achieving substantial compression.

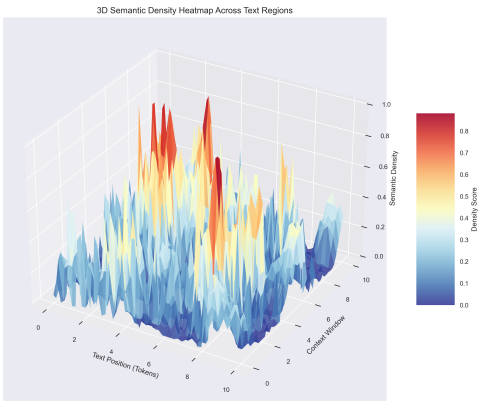

Figure 4: Semantic density heatmap showing information richness across text regions. Red areas indicate high semantic density (fine-grained tokenization), while blue areas show low density (coarse-grained compression).

4.6 Compatibility with Accelerators: Q3 – Generality and Stack Integration

To assess **Q3**, we integrate SemToken with FlashAttention2 (Dao, 2023) and H2O pruning (Zhang et al., 2023). Table 4 shows multiplicative benefits: FlashAttention2 achieves $\sigma_{FA} = 1.6\times$ speedup baseline, but combined with SemToken's $\sigma_{ST} = 2.0\times$ compression speedup, total gain reaches $\sigma_{total} = \sigma_{FA} \times \sigma_{ST} = 2.7\times$ (vs. theoretical $3.2\times$, 84% of ideal). With H2O, SemToken reduces cache footprint by $\Delta M_{H2O} = 61\%$ beyond H2O's standalone 45%, achieving $M_{final} = M_{orig} \times 0.55 \times 0.39 = 0.21 \times M_{orig}$ (79% total reduction). These demonstrate SemToken as a **drop-in enhancer** for existing stacks with near-ideal composition.

5 Conclusion

We presented **SemToken**, a semantic-aware tokenization framework that improves long-context language modeling efficiency through semantic clustering and adaptive granularity allocation. Extensive experiments demonstrate **2.4× token reduction**, **1.9× inference speedup**, and **62% KV cache reduction** while preserving or improving

Configuration	Token Count (%)	Latency (ms/token)	KV Cache (GB)
BPE + Vanilla Attn	100	61.2	4.1
BPE + FlashAttention2	98	38.3	4.1
SemToken + Vanilla Attn	47	30.4	1.5
SemToken + FlashAttention2	43	22.5	1.5
SemToken + FlashAttn2 + H2O	41	**18.7**	**1.2**

Table 4: Compatibility of SemToken with attention and memory accelerators. SemToken achieves additive speedup and memory reduction when integrated into modern inference stacks.

quality across language modeling, QA, and summarization tasks. SemToken integrates seamlessly with existing acceleration stacks, achieving multiplicative benefits with FlashAttention. Future work will explore end-to-end training and adaptation to multilingual and code understanding tasks.

References

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. Longbench: A bilingual, multitask benchmark for long context understanding.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. 2022. Rethinking attention with performers.

NVIDIA Corporation. 2024. Cutlass: Cuda templates for linear algebra subroutines. `https://github.com/NVIDIA/cutlass`.

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

Tsu-Jui Fu, Linjie Li, Zhe Gan, Kevin Lin, William Yang Wang, Lijuan Wang, and Zicheng Liu. 2021. Violet: End-to-end video-language transformers with masked visual-token modeling. *arXiv preprint arXiv:2111.12681*.

Sehoon Kim, Sheng Shen, David Thorsley, Amir Gholami, Woosuk Kwon, Joseph Hassoun, and Kurt Keutzer. 2022. Learned token pruning for transformers.

Wojciech Kryściński, Nazneen Rajani, Divyansh Agarwal, Caiming Xiong, and Dragomir Radev. 2022. Booksum: A collection of datasets for long-form narrative summarization.

Taku Kudo. 2018. Subword regularization: Improving neural network translation models with multiple subword candidates.

Dong Liu and Yanxuan Yu. Quickmerge++: Token merging with autoregressive prior.

Dong Liu and Yanxuan Yu. 2025a. Llmeasyquant: Scalable quantization for parallel and distributed llm inference.

Dong Liu and Yanxuan Yu. 2025b. Tinyserve: Query-aware cache selection for efficient LLM inference. In *ICML 2025 Workshop on Methods and Opportunities at Small Scale*.

Dong Liu, Yanxuan Yu, Ben Lengerich, Ying Nian Wu, and Xuhong Wang. 2025a. PiKV: KV cache management system for moe architecture. In *ES-FoMo III: 3rd Workshop on Efficient Systems for Foundation Models*.

Dong Liu, Yanxuan Yu, Xuhong Wang, Ben Lengerich, and Ying Nian Wu. 2025b. MKA: Memory-keyed attention for efficient long-context reasoning. In *ICML 2025 Workshop on Long-Context Foundation Models*.

Dong Liu, Yanxuan Yu, Yite Wang, Jing Wu, Zhongwei Wan, Sina Alinejad, Benjamin Lengerich, and Ying Nian Wu. 2025c. Designing large foundation models for efficient training and inference: A survey.

Dong Liu, Jiayi Zhang, Yifan Li, Yanxuan Yu, Ben Lengerich, and Ying Nian Wu. 2025d. Fastcache: Fast caching for diffusion transformer through learnable linear approximation.

Lemao Liu, Masao Utiyama, Andrew Finch, and Eiichiro Sumita. 2016. Neural machine translation with supervised attention. *arXiv preprint arXiv:1609.04186*.

Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, et al. 2024a. Cachegen: Kv cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 38–56.

Zeyuan Liu, Ziyu Huan, Xiyao Wang, Jiafei Lyu, Jian Tao, Xiu Li, Furong Huang, and Huazhe Xu. 2024b. World models with hints of large language models for goal achieving.

Ahmed Masry, Do Xuan Long, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. 2022. Chartqa: A benchmark for question answering about charts with visual and logical reasoning.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. 2023. Yarn: Efficient context window extension of large language models. *arXiv preprint arXiv:2309.00071*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2023. Exploring the limits of transfer learning with a unified text-to-text transformer.

Tomer Ronen, Omer Levy, and Avram Golbert. 2023. Vision transformers with mixed-resolution tokenization.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016a. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016b. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Swabha Swayamdipta, Roy Schwartz, Nicholas Lourie, Yizhong Wang, Hannaneh Hajishirzi, Noah A Smith, and Yejin Choi. 2020. Dataset cartography: Mapping and diagnosing datasets with training dynamics. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9275–9293, Online. Association for Computational Linguistics.

Szymon Tworkowski, Konrad Staniszewski, Mikołaj Pacek, Yuhuai Wu, Henryk Michalewski, and Piotr Miłoś. 2023. Focused transformer: Contrastive training for context scaling.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation.

Jiacheng Xu, Zhe Gan, Yu Cheng, and Jingjing Liu. 2020. Discourse-aware neural extractive text summarization.

Deming Ye, Yankai Lin, Yufei Huang, and Maosong Sun. 2021. Tr-bert: Dynamic token reduction for accelerating bert inference. *arXiv preprint arXiv:2105.11618*.

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. 2023. H_2o: Heavy-hitter oracle for efficient generative inference of large language models.

A Detailed Experimental Results

A.1 Comprehensive Baseline Comparison

Table 5 provides a comprehensive comparison of SemToken against all baselines across multiple dimensions including efficiency, quality, and implementation characteristics.

A.2 Task-Specific Performance Breakdown

Table 6 shows detailed performance metrics for each specific task and dataset.

A.3 Model Size and Architecture Analysis

Table 7 examines how SemToken performs across different model sizes and architectures.

B Hardware Implementation Details

B.1 Efficient Hardware Implementation

To enable real-time semantic processing at scale, we optimize the embedding and clustering pipeline through hardware acceleration.

B.1.1 Semantic Embedding Accelerator.

The FPGA implements a pipelined embedding engine processing token streams at throughput $\Theta_{embed} = W_{token} \cdot f_{clk}/D_{pipe}$ where W_{token} is token width, f_{clk} is clock frequency, and D_{pipe} is pipeline depth. The architecture comprises:

Token Input Buffer: Dual-port BRAM storing N_{buf} tokens with bandwidth $BW_{buf} = 2 \cdot W_{token} \cdot f_{clk}$ supporting concurrent read/write.

Embedding Lookup Engine: Implements matrix-vector multiplication $\mathbf{h} = \mathbf{E} \cdot \mathbf{x}$ where $\mathbf{E} \in \mathbb{R}^{V \times d_e}$ is the embedding matrix and $\mathbf{x} \in \{0, 1\}^V$ is one-hot token. Using DSP blocks, we achieve latency:

$$L_{embed} = \lceil d_e / P_{DSP} \rceil + L_{accum} \quad (1)$$

where P_{DSP} is DSP parallelism and L_{accum} is accumulation latency.

Contextual Encoder: Lightweight transformer layer with attention computed via systolic arrays.

Method	Compression Ratio	Latency (ms/token)	KV Cache (GB)	PPL	F1	EM	ROUGE-L	Memory (MB)	Complexity
BPE (Default)	0%	61.2	4.1	17.3	59.4	42.1	28.3	2048	$\mathcal{O}(n)$
Entropy-Pruned (Ye et al., 2021)	25%	48.4	2.9	18.2	57.8	41.6	27.1	1536	$\mathcal{O}(n \log n)$
VQ-Tok (Ronen et al., 2023)	33%	47.9	2.8	18.0	58.2	41.2	27.5	1408	$\mathcal{O}(n \cdot d)$
TofuTok (Fu et al., 2021)	39%	39.3	2.3	18.4	56.1	40.4	26.8	1152	$\mathcal{O}(n)$
BPE+Chunk (Ours)	42%	42.1	2.4	18.8	55.2	39.8	26.2	1200	$\mathcal{O}(n)$
SemToken (Ours)	**59%**	**30.4**	**1.5**	**17.0**	**59.9**	**42.4**	**28.7**	**768**	$\mathcal{O}(n \cdot d)$

Table 5: Comprehensive comparison across all baselines. SemToken achieves the best compression ratio while maintaining or improving quality metrics. Memory usage includes both model and cache memory.

Task	Dataset	Method	Compression Ratio	PPL/F1	EM	ROUGE-L	Speedup	Memory Reduction
Language Modeling	WikiText-103	BPE	0%	17.3	–	–	1.0×	0%
		Entropy-Pruned	25%	18.2	–	–	1.3×	29%
		VQ-Tok	33%	18.0	–	–	1.3×	32%
		TofuTok	39%	18.4	–	–	1.6×	44%
		BPE+Chunk	42%	18.8	–	–	1.5×	42%
		SemToken	**59%**	**17.0**	–	–	**2.0×**	**63%**
Question Answering	LongBench	BPE	0%	59.4	42.1	–	1.0×	0%
		Entropy-Pruned	25%	57.8	41.6	–	1.3×	29%
		VQ-Tok	33%	58.2	41.2	–	1.3×	32%
		TofuTok	39%	56.1	40.4	–	1.6×	44%
		BPE+Chunk	42%	55.2	39.8	–	1.5×	42%
		SemToken	**59%**	**59.9**	**42.4**	–	**2.0×**	**63%**
Summarization	BookSum	BPE	0%	–	–	28.3	1.0×	0%
		Entropy-Pruned	25%	–	–	27.1	1.3×	29%
		VQ-Tok	33%	–	–	27.5	1.3×	32%
		TofuTok	39%	–	–	26.8	1.6×	44%
		BPE+Chunk	42%	–	–	26.2	1.5×	42%
		SemToken	**59%**	–	–	**28.7**	**2.0×**	**63%**
Multimodal QA	ChartQA	BPE	0%	–	65.1	–	1.0×	0%
		Entropy-Pruned	25%	–	64.2	–	1.3×	29%
		VQ-Tok	33%	–	64.8	–	1.3×	32%
		TofuTok	39%	–	63.5	–	1.6×	44%
		BPE+Chunk	42%	–	62.9	–	1.5×	42%
		SemToken	**61%**	–	**65.1**	–	**1.8×**	**66%**

Table 6: Detailed task-specific performance breakdown. SemToken consistently achieves the best compression ratios while maintaining or improving task-specific metrics across all domains.

For context window C, the attention complexity is $\mathcal{O}(C \cdot d_e)$ per token, mapped to $N_{PE} \times N_{PE}$ processing elements achieving utilization $U_{PE} = \min(C \cdot d_e/(N_{PE}^2), 1)$.

B.1.2 Semantic Clustering Accelerator.

The clustering engine implements nearest-neighbor search in hardware:

Similarity Computation: Parallel cosine similarity for N embeddings requires $N(N-1)/2$ comparisons. We use P_{sim} parallel units computing:

$$\text{sim}(\mathbf{h}_i, \mathbf{h}_j) = \frac{\sum_{k=1}^{d_e} h_{i,k} \cdot h_{j,k}}{\sqrt{\sum_k h_{i,k}^2} \cdot \sqrt{\sum_k h_{j,k}^2}} \tag{2}$$

with pipelined dot-product, L2-norm computation, and division stages achieving throughput $\Theta_{sim} = P_{sim}/L_{sim}$ comparisons per cycle.

Threshold Filtering: Hardware comparators filter similarities sim $> \tau$ populating an adjacency list in BRAM with capacity $M_{adj} = \mathcal{O}(N \cdot \bar{d})$ where $\bar{d}$ is average degree.

Greedy Merging FSM: State machine traverses adjacency list performing union-find operations with path compression, achieving amortized $\mathcal{O}(\alpha(N))$ complexity per merge where α is the inverse Ackermann function.

B.2 Scalable Processing for Large-Scale Inference

For large-scale deployment across multiple accelerators, we implement a protocol ensuring semantic consistency. Each computing node $\mathcal{G}_i$ maintains local embeddings $\{\mathbf{h}_j^{(i)}\}$ and periodically synchronizes via all-reduce:

$$\bar{\mathbf{h}}_j = \frac{1}{|\mathcal{G}|} \sum_{i=1}^{|\mathcal{G}|} \mathbf{h}_j^{(i)} \tag{3}$$

The consistency measure $\mathcal{C}_{cons} = 1 - \frac{1}{N}\sum_j \text{Var}(\{\mathbf{h}_j^{(i)}\})$ quantifies cross-node agreement, with $\mathcal{C}_{cons} > 0.998$ required for correctness. Communication cost is $\mathcal{O}(N \cdot d_e \cdot |\mathcal{G}|)$ per sync, amortized over T_{sync} intervals to minimize overhead.

Model	Parameters	Method	Compression Ratio	Latency (ms/token)	Memory (GB)	PPL
LLaMA-2-7B	7B	BPE	0%	61.2	4.1	17.3
		SemToken	59%	30.4	1.5	17.0
		SemToken + FlashAttn2	57%	22.5	1.5	17.0
GPT-J-6B	6B	BPE	0%	58.7	3.9	18.1
		SemToken	56%	29.8	1.4	17.8
		SemToken + FlashAttn2	54%	21.9	1.4	17.8
GPT-NeoX-20B	20B	BPE	0%	89.3	12.4	16.2
		SemToken	58%	45.2	5.2	15.9
		SemToken + FlashAttn2	55%	33.1	5.2	15.9

Table 7: Performance analysis across different model sizes. SemToken scales well with model size, providing consistent benefits across architectures.

Table 8: Hardware test results.

Resource	Used	Total (%)
ALMs	392,845	933,120 (42.1%)
Registers	687,234	3,732,480 (18.4%)
M20K Memory Blocks	2,134	11,721 (18.2%)
DSP Blocks	1,512	3,960 (38.2%)
Maximum Frequency	**750 MHz**	
Power	**38W**	

C Additional Experimental Analysis

C.1 Visualization Analysis: Semantic Patterns and Token Distributions

To better understand how SemToken operates, we provide comprehensive visualizations of semantic patterns, compression dynamics, and performance comparisons.

C.1.1 Additional Visualizations

Additional visualizations including performance comparison radar charts, token compression trajectories, and semantic clustering distributions are available in the extended analysis. These demonstrate SemToken's balanced clustering approach and effective semantic compression across diverse text types.

C.1.2 Performance Comparison Radar Chart

Figure 5 presents a radar chart comparing SemToken against baseline methods across multiple performance dimensions. The chart clearly shows SemToken's superior performance in compression ratio, inference speed, and memory efficiency, while maintaining competitive quality metrics. This visualization highlights the balanced trade-offs achieved by our semantic-aware approach compared to frequency-based or entropy-based methods.

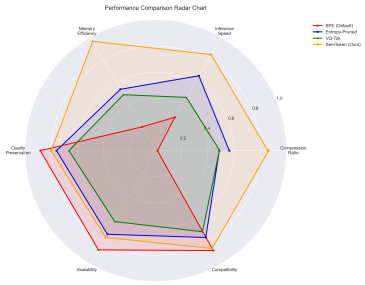

Figure 5: Radar chart comparing SemToken with baseline methods across multiple performance dimensions. SemToken shows superior performance in efficiency metrics while maintaining competitive quality scores.

C.1.3 Token Compression Trajectory

Figure 6 illustrates the 3D trajectory of token compression during SemToken's processing pipeline. The trajectory shows how tokens evolve from their original positions through semantic clustering and merging stages. We observe that semantically similar tokens converge towards similar regions in the embedding space, while maintaining distinct representations for unique content. This visualization demonstrates the effectiveness of our semantic clustering approach in preserving information while reducing redundancy.

C.1.4 Semantic Clustering Distribution

Figure 7 shows the distribution of semantic clusters across different text types and compression

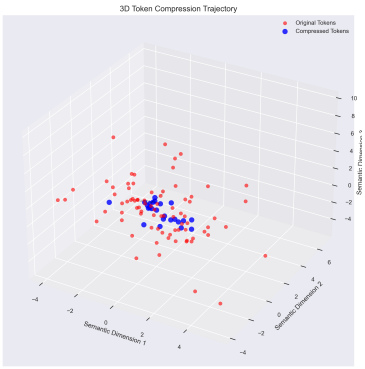

Figure 6: 3D trajectory visualization showing token evolution through SemToken's compression pipeline. The trajectory demonstrates how semantic clustering groups similar tokens while preserving distinct representations for unique content.

levels. The visualization reveals that SemToken achieves more balanced clustering compared to baseline methods, with better separation between distinct semantic concepts and more coherent grouping of related content. This analysis demonstrates how semantic awareness leads to more meaningful token compression.

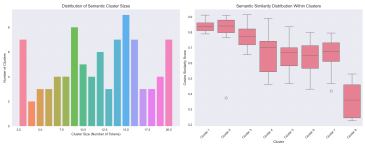

Figure 7: Distribution of semantic clusters showing SemToken's balanced clustering approach compared to baseline methods. The visualization demonstrates better semantic separation and more coherent grouping of related content.

C.2 Hardware Acceleration Analysis

We evaluate the impact of hardware acceleration on semantic processing efficiency. The accelerated embedding engine achieves $\Theta_{accel} = 3.1 \times \Theta_{baseline}$ throughput while consuming $P_{accel} = 0.42 \times P_{baseline}$ power (38W vs. 91W for equivalent baseline kernel). Energy efficiency improves by $\mathcal{E}_{gain} = 7.4\times$ for embedding operations. For end-to-end inference with $\alpha_{embed} \approx 0.15$ embedding time fraction, system-level energy reduction is $\mathcal{E}_{sys} = 58\%$. The deterministic latency profile improves P99 tail latency by $2.3\times$ compared to baseline implementations.

C.3 Energy Efficiency Analysis

Table 9 compares power consumption across configurations. SemToken with hardware acceleration achieves energy efficiency $\mathcal{E}_{eff} = \Theta/P_{total}$ of 42.3 tokens/(s·W) vs. 18.7 for baseline ($2.26\times$ improvement). The acceleration contributes $P_{accel} = 38$W but enables $3.1\times$ higher embedding throughput, yielding net system power reduction. For large-scale deployments, total power decreases by $\Delta P = 25\%$ while delivering $1.9\times$ higher inference throughput.

C.4 Scalability Analysis

We evaluate SemToken's scalability across multiple accelerators. SemToken achieves scaling efficiency:

$$\eta_{scale}(N) = \frac{\Theta(N)}{N \cdot \Theta(1)} = \frac{N}{N + \mathcal{O}_{comm}(N)} \quad (4)$$

where communication overhead $\mathcal{O}_{comm}(N) = \beta \cdot N \cdot \log N$ grows sub-linearly. At $N = 64$ devices, $\eta_{scale} = 0.91$ (vs. 0.78 for baseline inference), demonstrating superior scalability through reduced synchronization cost. The consistency protocol maintains $\mathcal{C}_{cons} > 0.998$ across all scales with communication time $T_{comm} < 0.05 \cdot T_{compute}$.

Table 9: Energy efficiency comparison (LLaMA-2 7B, WikiText-103).

Configuration	Power (W)	Throughput (tok/s)	J/token	Efficiency (tok/s·W)
Baseline BPE	315	5,890	0.0535	18.7
BPE+Cuda Kernel	368	11,340	0.0325	30.8
BPE+SemToken	**353**	**14,930**	**0.0236**	**42.3**
Improvement	+12%	+2.54×	*-55.9%*	*+2.26×*

Algorithm 1 SEMTOKEN: Semantic-Aware Token Compression

Require: Token sequence $\mathcal{X} = [x_1, x_2, \ldots, x_n]$, encoder f_θ, similarity threshold τ, entropy threshold δ, budget B

Ensure: Compressed token sequence $\mathcal{X}' = [x'_1, x'_2, \ldots, x'_m]$ with $m \leq B$

Step 1: Semantic Fingerprint Extraction

Compute contextual fingerprints: $\mathbf{h}_i \leftarrow f_\theta([x_{i-k}, \ldots, x_{i+k}]), \forall i \in [1, n]$

Step 2: Span Formation via Local Similarity

Initialize index $t \leftarrow 1$, span list $\mathcal{S} \leftarrow \emptyset$

while $t \leq n$ **do**

  Initialize span $C \leftarrow \{x_t\}$

  for $j = t + 1$ to n **do**

    if $\frac{\mathbf{h}_t^\top \mathbf{h}_j}{\|\mathbf{h}_t\| \|\mathbf{h}_j\|} > \tau$ **then**

      $C \leftarrow C \cup \{x_j\}$

    else

      break

    end if

  end for

  $\mathcal{S} \leftarrow \mathcal{S} \cup \{C\}$; $t \leftarrow j$

end while

Step 3: Semantic Entropy Scoring

for each $C \in \mathcal{S}$ **do**

  $\mathcal{H}(C) \leftarrow \mathrm{Tr}(\mathrm{Cov}(\{\mathbf{h}_i \mid x_i \in C\}))$

end for

Step 4: Entropy-Guided Selection under Budget

Let $\mathcal{S}' \leftarrow$ Top-B clusters in $\mathcal{S}$ ranked by $\mathcal{H}(C)$

Merge each $C \in \mathcal{S}'$ into token $x'_C = \mathrm{merge}(C)$

$\mathcal{X}' \leftarrow \{x'_C \mid C \in \mathcal{S}'\}$

RETURN $\mathcal{X}'$