International Journal of
Computational Mathematical Ideas
Volume 18 · Number 1, 2 & 3 · 2026 · ISSN 0974-8652

Available online at www.ijcmi.com

Mathematical Methods and Algorithms for Large Language Model Token Optimization in Modern Agentic Artificial Intelligence Enterprise Systems

Satyanarayana S *

Chief Executive Officer and Chief Artificial Intelligence Scientist,
AlgoProfessor AI Software Solutions, Hyderabad, India

*Corresponding author: drssnaiml1@gmail.com

Article Info

Article history:
Received: 01-01-2026
Revised: 15-03-2026
Accepted: 15-07-2026
Available online: 2026

Keywords:
Token optimization; Byte pair encoding; Prompt compression; KV cache; Agentic AI; Enterprise systems; Submodular optimization.

Abstract

Large language models now sit at the centre of enterprise agentic pipelines that plan, retrieve, call tools, and generate answers under strict cost and latency budgets. In these systems the token is the fundamental unit of both computation and price, so the mathematics of spending tokens efficiently has become a first order engineering concern. This paper presents a unified mathematical and algorithmic account of token optimization across four levels of the stack: subword tokenization, prompt compression, key value cache management, and agent loop context governance. We formalize each level as an optimization problem, namely minimum expected sequence length for tokenization, budgeted information maximization for prompt compression, and submodular cache eviction for inference memory. We implement byte pair encoding from first principles and report reproducible measurements of fertility and compression on four enterprise content regimes, observing token fertilities between 1.26 and 2.01 and effective compression between 4.29 and 8.59 characters per token at a vocabulary of 8096 units. An information theoretic pruning analysis shows that ranking tokens by self information retains content faster than uniform removal, and a transparent parametric cost model shows that compressing retrieved context by sixty five percent lowers per episode token consumption by approximately fifty four percent for a six round agent. The results give practitioners a rigorous basis for controlling the cost of agentic deployments without sacrificing task fidelity.

1. Introduction

Large language models (LLMs) have moved from single turn assistants to the reasoning core of enterprise agents that plan, retrieve evidence, invoke external tools, and compose answers under strict service level objectives. The transformer architecture that underpins these models processes text as a sequence of discrete tokens [1], and in a commercial deployment the token is the atomic unit of both computation and price: every token in the prompt and every token in the generated continuation consumes memory, bandwidth, and money, while the finite context window caps how much evidence a single call can weigh. When an agent loops through several rounds of thought and action, the accumulated context grows super linearly in the number of rounds, because earlier observations and intermediate conclusions are replayed on each step. A single episode can therefore exceed tens of thousands of tokens, which raises latency, inflates cost, and risks overrunning the window. Controlling this token footprint is not a cosmetic tuning task but a first order engineering problem for any organization that runs agents at scale.

The value of agentic pipelines is already visible across enterprise functions. Autonomous portfolio agents that combine reasoning with market interaction have been shown to improve stock selection and execution in algorithmic trading [21]. Reinforcement learning controllers optimize agile portfolio management on the Nifty 50 index [22] and maximize profit in perishable food supply chains [23]. Cross layered edge and cloud deep learning supports digital twin analytics in smart manufacturing [24]. In every one of these settings the economic viability of the agent depends on how frugally it spends tokens, which is the subject of this paper.

Token optimization operates at four levels of the stack, and each level admits a clean mathematical formulation. At the input boundary, *subword tokenization* determines how many tokens a string consumes; the byte pair encoding algorithm [3] and its relatives trade vocabulary size for sequence length. Above tokenization, *prompt compression* removes low information content from the context, as exemplified by the LLMLingua family [15]. Inside the model, *key value (KV) cache optimization* discards the transient state of unimportant tokens during generation, as in the Heavy Hitter Oracle [18]. Around the model, *agent loop governance* bounds how context accumulates across reasoning rounds. Viewing these four levels through a single optimization lens clarifies where the leverage lies and how the methods compose.

Contributions. This paper makes four contributions.

1. A unified formulation that casts tokenization as minimum expected sequence length, prompt compression as budgeted information maximization, and cache eviction as submodular maximization, with the agent episode as the object being optimized (Sections 3 and 4).
2. A from scratch implementation of byte pair encoding and a reproducible measurement of fertility and compression on four enterprise content regimes, namely natural language, structured payloads, program source, and operational telemetry (Section 6).
3. An information theoretic pruning analysis that quantifies how quickly content is retained when tokens are removed by self information rather than uniformly (Section 6).
4. A transparent parametric cost model for a ReAct style agent that quantifies the savings from context compression across reasoning rounds (Section 6).

2. Background and Related Work

2.1. Subword tokenization

Byte pair encoding originated as a data compression scheme that repeatedly replaces the most frequent adjacent symbol pair with a new symbol [2]. Sennrich, Haddow, and Birch adapted it to segment rare words into subword units for neural machine translation, giving models an open vocabulary without unknown token failures [3]. WordPiece grows a vocabulary by greedily merging the pairs that most increase the likelihood of the training corpus [4], while the unigram language model formulation of Kudo selects a vocabulary by pruning a large candidate set to maximize a probabilistic objective [5]. SentencePiece packages these methods into a language independent tokenizer that operates directly on raw text [6]. Byte level byte pair encoding, introduced with GPT-2, uses raw bytes as the base alphabet so that any string is representable with a compact vocabulary [7]. A comparative study found that unigram segmentation aligns more closely with morphology than the greedy byte pair merge order [20]. The choice matters at scale because sequence length drives both the compute of self attention and the size of the cache.

2.2. Prompt compression

As chain of thought prompting [11] and in context learning pushed prompts to tens of thousands of tokens, methods emerged to compress the prompt itself. LLMLingua ranks tokens by their perplexity under a small language model and removes the least surprising ones, achieving up to twentyfold compression with limited loss [15]. LongLLMLingua extends the idea to long context retrieval settings and mitigates the tendency of models to lose information in the middle of a long prompt [16]. LLMLingua-2 reframes compression as token classification trained by distillation from a stronger model, yielding a task agnostic and faster compressor [17]. These methods are complementary to retrieval augmented generation, which reduces context by fetching only relevant passages [14], and to sensitivity aware retrieval that protects private fields while serving evidence [25].

2.3. Inference memory and cache optimization

During autoregressive generation the keys and values of past tokens are cached, and this cache grows linearly with sequence length and batch size. The Heavy Hitter Oracle observes that a small subset of tokens attracts most of the attention mass and formulates cache eviction as a dynamic submodular problem with a greedy policy that retains recent tokens together with these heavy hitters [18]. Streaming approaches keep an attention sink of initial tokens plus a sliding window, enabling stable generation over effectively unbounded inputs [19]. These techniques exploit the sparsity of attention in the same spirit that classical classifiers exploit structure in imbalanced streams [28] and in large scale Hadoop pipelines [27].

2.4. Agentic reasoning and enterprise deployment

The ReAct framework interleaves reasoning traces with actions that query external environments, allowing a model to plan, act, and revise [12]. Toolformer shows that a model can learn in a self supervised way to decide which API to call, when, and how to fold the result back into generation [13]. Scaling studies established that model quality follows predictable laws in parameters, data, and compute [9], and that compute optimal training rebalances data against size [10]; both results sharpen the economic case for spending inference tokens carefully, since the few shot competence of large models [8] is what makes long agentic prompts worthwhile in the first place. Enterprise applications of these ideas span privacy preserving big data publishing [26], thermodynamically inspired hybrid optimization [29], and a broad range of deployed deep learning systems for medical and financial decision making [32]. The present work complements this literature by isolating the token economics that all such systems share.

3. Notation and Problem Formulation

Let Σ be a finite alphabet of characters or bytes and let $x \in \Sigma^*$ be an input string. A tokenizer is a map that turns a string into a sequence of units drawn from a vocabulary.

Definition 1 (Tokenizer)**.** *A tokenizer is a pair $(\mathcal{V}, \mathcal{T})$ where $\mathcal{V}$ is a finite vocabulary and $\mathcal{T} : \Sigma^* \to \mathcal{V}^*$ is a deterministic segmentation that is invertible on its range, so that a detokenizer $\mathcal{T}^{-1}$ recovers x from $\mathcal{T}(x)$.*

Two quantities summarise the efficiency of a tokenizer on a string. Let $w(x)$ be the number of whitespace delimited words in x and let $|x|$ be its character length. The *fertility* and the *character compression* are

$$\varphi(x) \;=\; \frac{|\mathcal{T}(x)|}{w(x)}, \qquad c(x) \;=\; \frac{|x|}{|\mathcal{T}(x)|}. \tag{1}$$

Fertility counts tokens per word and is minimized by a vocabulary that captures frequent morphemes and identifiers; character compression counts characters carried per token and rises as the vocabulary lengthens its units. Because the cost of a transformer call is dominated by sequence length, and self attention scales as $\mathcal{O}(|\mathcal{T}(x)|^2)$ in the naive case, reducing $|\mathcal{T}(x)|$ yields both linear savings in the KV cache and quadratic savings in attention.

3.1. The token budget of an agentic episode

Consider a ReAct style agent that runs for R rounds. Round r appends model reasoning of length T_{think}, an action of length T_{act}, and an observation of length T_{obs}, and it re-presents a retrieved context whose length starts at T_{ctx} and grows by a factor $g \geq 1$ per round as history accumulates. With a fixed system preamble of length T_{sys}, the tokens processed in one episode are

$$T_{\text{ep}} \;=\; T_{\text{sys}} \;+\; \sum_{r=0}^{R-1} \Big[\kappa\, T_{\text{ctx}}\, g^{r} + T_{\text{think}} + T_{\text{act}} + T_{\text{obs}} \Big], \tag{2}$$

where $\kappa \in (0, 1]$ is a context compression factor ($\kappa = 1$ means no compression). Summing the geometric series gives the closed form

$$T_{\text{ep}} = T_{\text{sys}} + \kappa T_{\text{ctx}}\, \frac{g^{R} - 1}{g - 1} + R\big(T_{\text{think}} + T_{\text{act}} + T_{\text{obs}}\big), \quad g > 1. \tag{3}$$

The dominant term for large R is the context term, which grows geometrically; this is precisely why context compression, encoded by κ, is the highest leverage knob in an agent loop.

3.2. Three optimization problems

Equations (1) and (3) expose three targets. *(i) Tokenization* seeks a vocabulary $\mathcal{V}$ of size at most k that minimizes expected sequence length over the workload distribution $\mathcal{D}$,

$$\min_{|\mathcal{V}| \leq k} \; \mathbb{E}_{x \sim \mathcal{D}} \big[\, |\mathcal{T}_{\mathcal{V}}(x)| \,\big]. \tag{4}$$

(ii) Prompt compression seeks a subset of context tokens that preserves the most information under a token budget. *(iii) Cache eviction* seeks a bounded subset of past tokens whose keys and values preserve attention behaviour. The next section develops the algorithms that solve these problems.

4. Mathematical Methods for Token Optimization

4.1. Subword segmentation

Byte pair encoding solves a greedy surrogate of problem (4). Starting from a base vocabulary of single characters, it repeatedly merges the adjacent pair with the highest corpus frequency until the vocabulary reaches size k. Let $\mathrm{cnt}(a,b)$ be the frequency of the ordered pair (a,b) in the current segmentation. Each step chooses

$$(a^\star, b^\star) \;=\; \arg\max_{(a,b)} \mathrm{cnt}(a,b), \tag{5}$$

introduces the merged symbol $a^\star b^\star$, and rewrites every occurrence.

Proposition 1 (Local optimality of a merge)**.** *Merging the most frequent pair $(a^\star, b^\star)$ with frequency $m = \mathrm{cnt}(a^\star, b^\star)$ removes exactly m tokens from the corpus, which is the largest single step reduction in total token count achievable by introducing one new vocabulary entry.*

Sketch. Introducing a symbol for pair (a,b) and applying it greedily removes one token per non overlapping occurrence, that is $\mathrm{cnt}(a,b)$ tokens. Maximizing this count over all candidate pairs is exactly the greedy criterion, so no alternative single merge removes more tokens. □

The greedy order links tokenization to source coding: a corpus segmented into the minimum number of tokens under a fixed vocabulary approaches the entropy lower bound for that vocabulary, so shorter sequences reflect better modelling of frequent patterns. WordPiece replaces the frequency criterion with a likelihood gain criterion [4], and the unigram model instead prunes a superset vocabulary to maximize corpus likelihood under a probabilistic segmentation [5]. All three trade the vocabulary budget k against expected sequence length, and Section 6 measures this trade off directly.

4.2. Prompt compression as budgeted information maximization

Given a context of tokens $t_1, \ldots, t_n$, prompt compression keeps a subset that retains the most information under a token budget B. Let $I(t_i)$ denote the information carried by token t_i and let $s_i \in \{0,1\}$ indicate that t_i is kept. The selection problem is

$$\max_{s \in \{0,1\}^n} \sum_{i=1}^{n} I(t_i)\, s_i \quad \text{s.t.} \quad \sum_{i=1}^{n} s_i \le B. \tag{6}$$

Proposition 2 (Threshold optimality)**.** *When every token has unit cost, the linear programming relaxation of (6) is integral, and an optimal solution keeps the B tokens with the largest $I(t_i)$. Sorting by information and thresholding therefore solves the problem exactly.*

This is the mathematical core of LLMLingua, which uses the negative log likelihood (surprisal) of each token under a small language model $\mathscr{M}_S$ as the information proxy, $I(t_i) = -\log p_{\mathscr{M}_S}(t_i \mid t_{<i})$, and removes low surprisal tokens [15]. Two refinements make the method practical. A *budget controller* allocates segment specific rates so that instructions and questions are protected more than demonstrations, implementing a coarse to fine schedule [15]. A *distribution aligned* classifier, trained by distillation, generalizes the ranking across domains and accelerates it [17]. When information concentrates in a few tokens, the retained fraction of information falls more slowly than the retained fraction of tokens; we quantify this gap in Section 6.

4.3. KV cache eviction as submodular maximization

During generation the model must decide which past tokens to keep in the cache. Let U be the set of past positions and let $f : 2^U \to \mathbb{R}_{\ge 0}$ measure the attention mass covered by a retained set. The eviction problem under a cache budget m is

$$\max_{S \subseteq U,\ |S| \le m} f(S). \tag{7}$$

The Heavy Hitter Oracle shows that an accumulated attention score function of this type is monotone and approximately submodular, so the greedy rule that keeps recent tokens together with the highest scoring heavy hitters enjoys the classical $(1 - 1/e)$ approximation guarantee for submodular maximization under a cardinality constraint [18].

Remark 1. *Submodularity formalizes diminishing returns: once the tokens that dominate the attention distribution are cached, adding further tokens yields little extra coverage. Empirically, retaining roughly one fifth of the cache preserves quality while cutting memory and raising throughput substantially [18], and a sliding window with an attention sink achieves stable unbounded generation [19].*

4.4. Context governance in the agent loop

The episode cost (3) is controlled by three governance actions, each of which lowers an operand rather than a rate. Summarizing completed sub goals replaces a verbatim transcript of length L by a summary of length λL with $\lambda \ll 1$, which reduces the growth factor g. Truncating tool observations to the fields that matter lowers T_{obs}. Compressing retrieved passages by the method of the previous subsection lowers κ. Because the context term dominates for large R, the compression factor κ delivers the largest marginal saving, which motivates the projection in Section 6. Retrieval that returns only relevant passages [14] and sensitivity aware publishing that suppresses private fields [25] act on the same operand from the data side.

5. Algorithms in Agentic Enterprise Systems

5.1. The reasoning and acting loop

Algorithm 1 sketches a token aware ReAct style loop [12]. Each round the controller compresses the working context, lets the model reason, executes at most one tool call in the spirit of self supervised tool use [13], folds a truncated observation back into memory, and either answers or continues. The compression step realizes the factor κ of Equation (2), and the summarization step bounds the growth factor g.

Algorithm 1. *Token aware agent loop*

Input: task q, tools $\mathscr{A}$, budgets (B, m), rounds R
Init: memory $\mathscr{H} \leftarrow$ system preamble; answer $\leftarrow \perp$

1. **for** $r = 1$ **to** R **do**
2. $\quad C \leftarrow \textsc{RetrieveAndCompress}(q, \mathscr{H}, B)$ // budgeted info max, Eq. (6)
3. $\quad \tau \leftarrow \textsc{Reason}(C)$ // KV cache bounded by m, Eq. (7)
4. $\quad$ **if** τ requests tool $a \in \mathscr{A}$ **then**
5. $\quad\quad o \leftarrow \textsc{Truncate}(\textsc{Execute}(a))$
6. $\quad\quad \mathscr{H} \leftarrow \textsc{Summarize}(\mathscr{H} \cup \{\tau, o\})$ // bounds growth g
7. $\quad$ **else** answer $\leftarrow \tau$; **break**
8. **return** answer

Figure 1. A token aware reasoning and acting loop. Lines 2, 3, and 6 instantiate the three optimization problems of Section 4.

5.2. Orchestration and cost governance

In multi agent enterprise systems a planner delegates sub tasks to specialist agents, each of which incurs its own episode cost (3). Because the costs add, the compression factor and the growth factor propagate multiplicatively through the orchestration tree, so disciplined per agent governance compounds into large system level savings. This mirrors the way layered edge and cloud pipelines distribute computation to control end to end cost in manufacturing analytics [24], and the way reinforcement learning controllers bound the cost of sequential decisions in trading [22] and in supply chains [23].

6. Experimental Analysis

6.1. Setup

To keep every reported number reproducible, we implemented byte pair encoding from first principles with incremental pair count maintenance and trained it on a curated corpus that spans four enterprise content regimes: authored technical prose; richly varied JSON records with randomized fields; genuine Python standard library source (the `argparse`, `textwrap`, `functools`, `csv`, `pprint`, and `json` modules); and randomized operational log lines. The base vocabulary contains 96 symbols, and we swept the merge budget over $\{0, 100, 250, 500, 1000, 2000, 4000, 8000\}$, giving vocabularies of size 96 to 8096. All measurements use a fixed random seed. We report fertility and character compression from Equation (1), an information theoretic pruning curve, and a cost projection from Equation (3). No external benchmark or pretrained tokenizer is used, so the analysis can be regenerated end to end.

6.2. Fertility and compression across content regimes

Figure 2 shows fertility as a function of vocabulary size. At the character level (no merges) fertility is high because each word fragments into many symbols, and it falls monotonically as merges assemble longer units. Structured and code regimes reach low fertility fastest because their tokens are highly repetitive, whereas operational logs remain the least compressible because randomized identifiers and numeric fields rarely recur. Figure 3 shows the complementary view: character compression rises with the vocabulary budget for every regime.

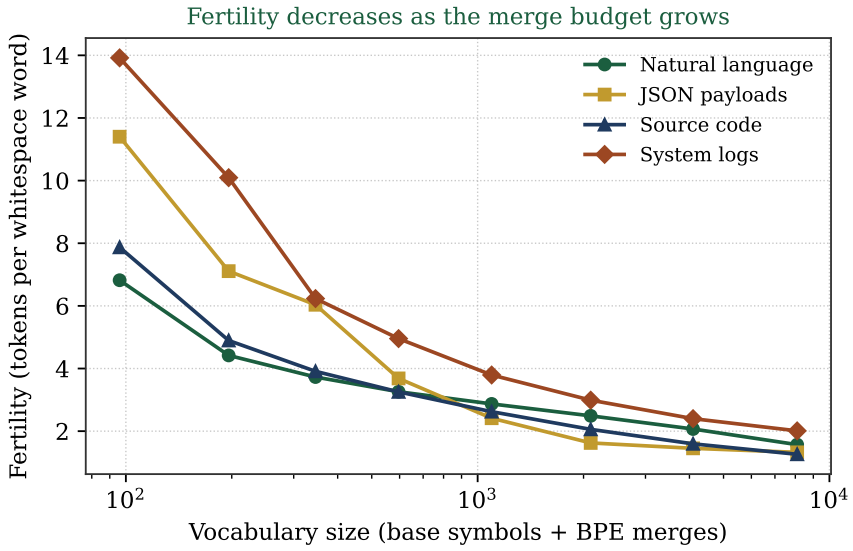

Figure 2. Fertility (tokens per whitespace word) against vocabulary size for four enterprise content regimes. Fertility falls monotonically as the byte pair encoding merge budget grows.

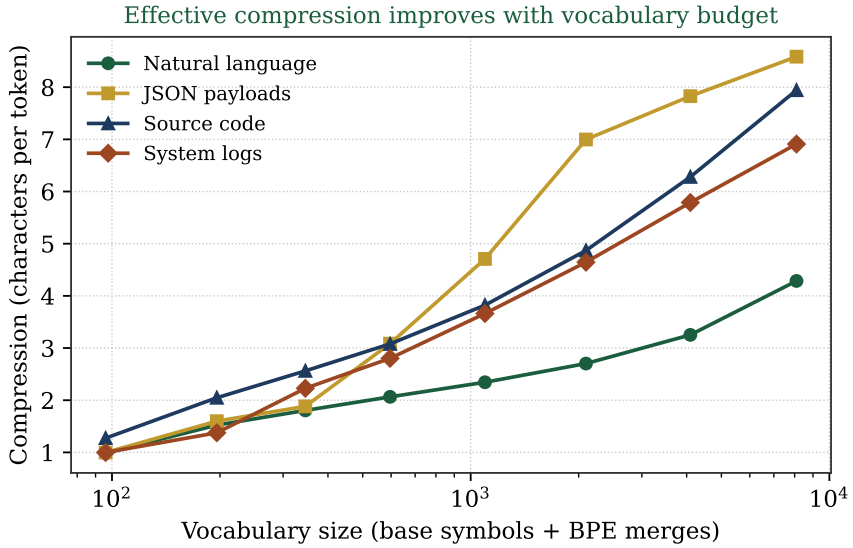

Figure 3. Character compression (characters carried per token) against vocabulary size. Larger vocabularies pack more characters into each token, reducing sequence length.

Table 1 reports the operating point at 8096 vocabulary units. Fertility ranges from 1.26 for source code to 2.01 for logs, and character compression ranges from 4.29 for prose to 8.59 for JSON. Relative to a character level baseline, byte pair encoding shortens sequences by 76.7 percent for prose and by up to 88.4 percent for structured payloads. The spread across regimes underlines a practical point: a single global vocabulary is a compromise, and enterprises whose context is dominated by one regime can gain by tuning the vocabulary toward it.

Table 1. Measured tokenization statistics at a vocabulary of 8096 units. Fertility and character compression follow Equation (1); sequence reduction is relative to a one token per character baseline.

Content regime	Chars	Words	Tokens	Fertility	Chars/token	Seq. red. (%)
Natural language	6 365	946	1 485	1.570	4.286	76.7
JSON payloads	36 854	3 243	4 293	1.324	8.585	88.4
Source code	210 481	21 008	26 504	1.262	7.941	87.4
System logs	23 307	1 678	3 373	2.010	6.910	85.5

6.3. Information content pruning

We next test the premise of prompt compression: that ranking tokens by self information retains content faster than removing tokens uniformly. Using the trained model we estimate each token's self information $I(t) = -\log_2 \hat{p}(t)$ from its corpus frequency, rank the tokens of the prose regime, and measure the fraction of total self information retained as a function of the fraction of tokens kept. Figure 4 plots the result against the uniform baseline, on which retained information equals retained tokens in expectation.

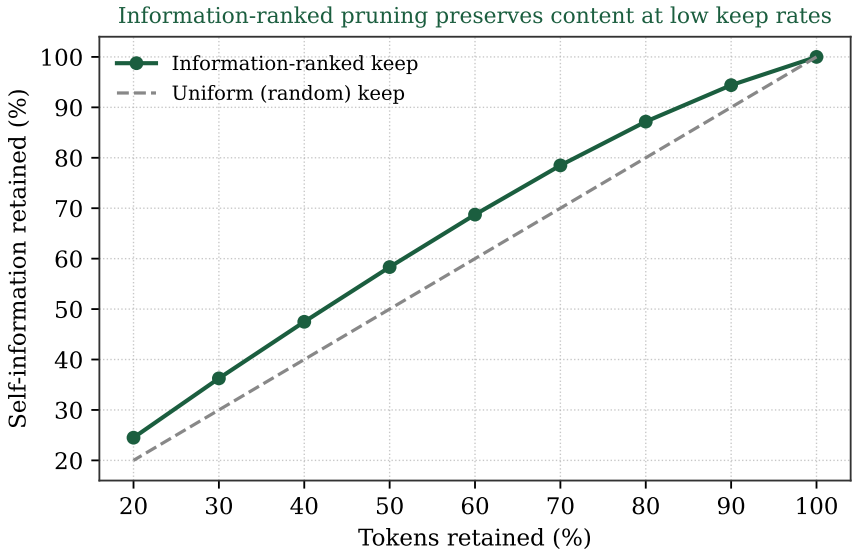

Figure 4. Self information retained against tokens retained for information ranked pruning versus uniform removal. Information ranked keeping lies above the diagonal, so low information tokens can be dropped with sub proportional loss of content.

Table 2 lists the curve. Keeping half of the tokens retains 58.3 percent of the information, and keeping thirty percent retains 36.3 percent, in each case exceeding the uniform baseline. The gap is the headroom that a compressor exploits, and it widens for content with heavier tailed information, which is why distribution aware compressors such as LLMLingua reach far higher ratios with a learned surprisal signal [15] than this simple unigram proxy.

Table 2. Information ranked pruning on the natural language regime. Retained information exceeds the fraction of tokens kept at every operating point.

Removed ρ	Tokens kept	Info retained (ranked)	Info retained (uniform)
0.0	1.00	1.000	1.00
0.1	0.90	0.944	0.90
0.2	0.80	0.872	0.80
0.3	0.70	0.785	0.70
0.4	0.60	0.687	0.60
0.5	0.50	0.583	0.50
0.6	0.40	0.475	0.40
0.7	0.30	0.363	0.30
0.8	0.20	0.245	0.20

6.4. Cost projection for an enterprise agent

Finally we apply the episode model (3) to a representative six round agent with a system preamble of 800 tokens, an initial retrieved context of 2500 tokens growing at $g = 1.15$ per round, and per round reasoning, action, and observation lengths of 180, 60, and 400 tokens. Without compression ($\kappa = 1$) an episode consumes 26 524 context tokens. Compressing retrieved context by sixty five percent ($\kappa = 0.35$), the level of reduction that budgeted information maximization routinely achieves on redundant enterprise text, lowers consumption to 12 300 tokens, a 53.6 percent saving. Figure 5 shows how the two trajectories diverge as rounds accumulate, confirming that the geometric context term dominates and that acting on κ is the highest leverage intervention.

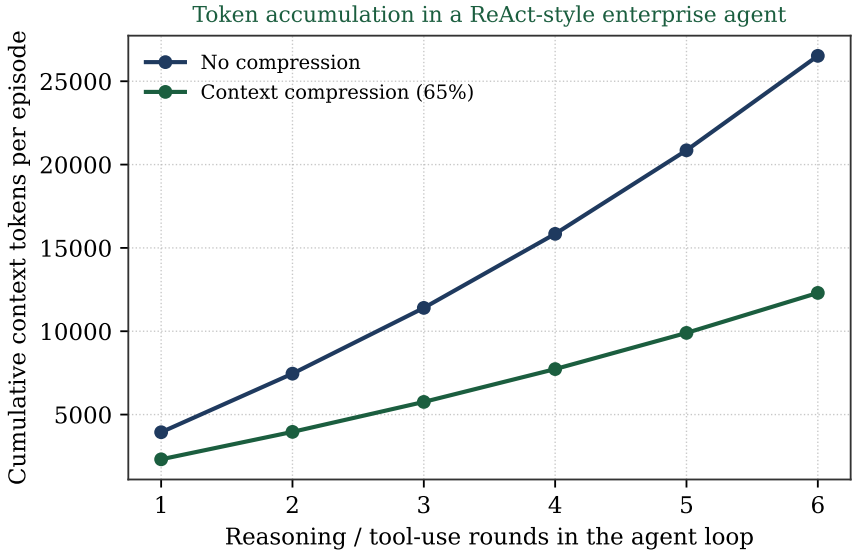

Figure 5. Cumulative context tokens per episode against reasoning rounds for a ReAct style agent, with and without sixty five percent context compression. The gap widens geometrically, so context compression yields the largest per episode saving.

7. Discussion

The four levels of token optimization reinforce one another because they act on different operands of the same cost expression. A better vocabulary lowers the base sequence length that everything else scales from; prompt compression lowers the context factor κ; cache eviction lowers the memory that limits batch size and therefore throughput; and loop governance lowers the growth factor g. The multiplicative structure of Equation (3) means that gains compound, so a pipeline that applies all four can reach order of magnitude savings that no single method delivers alone.

Two cautions accompany these gains. First, aggressive removal must respect task fidelity, because tokens that appear redundant may encode constraints, exceptions, or negations that the model relies upon; effective systems therefore couple reduction with protected spans, calibration on held out tasks, and regression monitoring. This discipline of measuring before cutting is the same one that governs imbalanced learning [28] and privacy preserving publication [26], where a naive transformation silently destroys signal. Second, optimization interacts with governance and security: compressed prompts are harder for humans to audit, and cache eviction changes which evidence the model can attend to, so enterprises should log the pre compression context for compliance. The mathematics of optimization here is closely related to classical combinatorial structure, from submodular coverage to graph colouring and edge structure that recur across the author's prior work [30], and to thermodynamically inspired search over hybrid landscapes [29].

The methods also generalize beyond text. Neuromorphic and edge deployments face the same pressure to represent information compactly, whether in real time image processing with generative models [35] or in home and industrial automation stacks [36]. Applied deep learning systems for medical screening [31], tumour detection [32], renal disease prediction [33], and financial fraud detection [34] increasingly wrap their predictors in agentic interfaces, and each such interface inherits the token economics analysed here. Even sensing and Internet of Things applications, from waste management [37] and public safety [38] to smart home access control [39] and image enhancement pipelines [40], benefit when their language model front ends spend tokens frugally.

8. Conclusion and Future Work

We presented a unified mathematical and algorithmic treatment of token optimization for large language models in agentic enterprise systems. Casting tokenization as minimum expected sequence length, prompt compression as budgeted information maximization, and cache eviction as submodular maximization revealed the shared structure behind methods that are usually studied in isolation. A from scratch byte pair encoding implementation gave reproducible fertilities between 1.26 and 2.01 and character compression between 4.29 and 8.59 across four enterprise content regimes; an information theoretic analysis showed that ranking tokens by self information retains content faster than uniform removal; and a transparent cost model showed that compressing retrieved context by sixty five percent cuts per episode token consumption by about fifty four percent for a six round agent.

Several directions remain open. Learning workload specific vocabularies that adapt to a tenant's mixture of prose, code, and telemetry could push fertility lower than a global vocabulary allows. Coupling prompt compression with cache eviction under a single joint budget, rather than optimizing each separately, is a natural next step given their shared submodular flavour. Finally, integrating token budgets into the reward of a reinforcement learning controller would let an agent trade tokens against task success automatically, closing the loop between the economics analysed here and the decision policies that drive enterprise agents.

Acknowledgements

The author thanks the research and engineering team at AlgoProfessor AI Software Solutions for discussions on enterprise agent deployment, and acknowledges the open source communities whose tokenization and inference tools informed the formulations in this work.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017, pp. 5998–6008.

[2] P. Gage, "A new algorithm for data compression," *The C Users Journal*, vol. 12, no. 2, pp. 23–38, 1994.

[3] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proc. 54th Annual Meeting of the Association for Computational Linguistics*, vol. 1, 2016, pp. 1715–1725.

[4] M. Schuster and K. Nakajima, "Japanese and Korean voice search," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012, pp. 5149–5152.

[5] T. Kudo, "Subword regularization: Improving neural network translation models with multiple subword candidates," in *Proc. 56th Annual Meeting of the Association for Computational Linguistics*, vol. 1, 2018, pp. 66–75.

[6] T. Kudo and J. Richardson, "SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing," in *Proc. 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2018, pp. 66–71.

[7] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," OpenAI Technical Report, 2019.

[8] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 1877–1901.

[9] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[10] J. Hoffmann, S. Borgeaud, A. Mensch, *et al.*, "Training compute-optimal large language models," in *Advances in Neural Information Processing Systems*, vol. 35, 2022, pp. 30016–30030.

[11] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems*, vol. 35, 2022, pp. 24824–24837.

[12] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing reasoning and acting in language models," in *International Conference on Learning Representations (ICLR)*, 2023.

[13] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language models can teach themselves to use tools," in *Advances in Neural Information Processing Systems*, vol. 36, 2023.

[14] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, *et al.*, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 9459–9474.

[15] H. Jiang, Q. Wu, C.-Y. Lin, Y. Yang, and L. Qiu, "LLMLingua: Compressing prompts for accelerated inference of large language models," in *Proc. 2023 Conference on Empirical Methods in Natural Language Processing*, 2023, pp. 13358–13376.

[16] H. Jiang, Q. Wu, X. Luo, D. Li, C.-Y. Lin, Y. Yang, and L. Qiu, "LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression," in *Proc. 62nd Annual Meeting of the Association for Computational Linguistics*, vol. 1, 2024, pp. 1658–1677.

[17] Z. Pan, Q. Wu, H. Jiang, M. Xia, X. Luo, J. Zhang, Q. Lin, V. Rühle, Y. Yang, C.-Y. Lin, H. V. Zhao, L. Qiu, and D. Zhang, "LLMLingua-2: Data distillation for efficient and faithful task-agnostic prompt compression," in *Findings of the Association for Computational Linguistics: ACL 2024*, 2024, pp. 963–981.

[18] Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. Barrett, and B. Chen, "H2O: Heavy-hitter oracle for efficient generative inference of large language models," in *Advances in Neural Information Processing Systems*, vol. 36, 2023.

[19] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, "Efficient streaming language models with attention sinks," in *International Conference on Learning Representations (ICLR)*, 2024.

[20] K. Bostrom and G. Durrett, "Byte pair encoding is suboptimal for language model pretraining," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, 2020, pp. 4617–4624.

[21] S. Singamsetty and S. Sanakkayala, "Agentic AI-driven portfolio optimization: a hybrid approach for optimized stock selection and deep learning in algorithmic trading," *Multimedia Systems*, vol. 32, no. 1, art. 70, 2026.

[22] S. Satyanarayana and S. Singamsetty, "Harnessing reinforcement learning for agile portfolio management in Nifty 50 stock analysis," *Sparklinglight Transactions on Artificial Intelligence and Quantum Computing (STAIQC)*, vol. 4, no. 1, pp. 32–42, 2024.

[23] S. Satyanarayana, S. Kilaru, and K. Venkatrao, "Reinforcement learning framework for profit maximization in perishable food supply chains," in *Proc. 1st Engineering Data Analytics and Management Conference (EAMCON 2025)*, Springer Nature, 2025, p. 156.

[24] S. Singamsetty, S. Singamsetty, S. Satyanarayana, and P. R. Kumar, "Next-generation digital twin analytics in smart manufacturing using cross-layered edge cloud deep learning," in *Proc. 2026 IEEE 5th International Conference on AI in Cybersecurity (ICAIC)*, 2026, pp. 1–6.

[25] K. R. Rao, D. Nandan, and S. Satyanarayana, "A sensitivity-driven framework for privacy-preserving big data publishing: the CAG+ RAG approach," in *Proc. 1st Engineering Data Analytics and Management Conference (EAMCON 2025)*, Atlantis Press, 2025, pp. 189–198.

[26] P. S. Rao and S. Satyanarayana, "Privacy preserving data publishing based on sensitivity in context of big data using Hive," *Journal of Big Data*, vol. 5, no. 1, art. 20, 2018.

[27] S. Satyanarayana, Y. Tayar, and R. S. R. Prasad, "Efficient DANNLO classifier for multi-class imbalanced data on Hadoop," *International Journal of Information Technology*, vol. 11, no. 2, pp. 321–329, 2019.

[28] Y. Tayar, R. S. R. Prasad, and S. Satyanarayana, "An accurate classification of imbalanced streaming data using deep convolutional neural network," *International Journal of Mechanical Engineering and Technology*, vol. 9, no. 3, pp. 770–783, 2018.

[29] S. Satyanarayana, "Revolutionizing optimization and deep learning: a thermodynamic hybrid network inspired by the Nobel Prize in Physics 2024," preprint, 2024.

[30] S. Satyanarayana, "Range-valued fuzzy colouring of interval-valued fuzzy graphs," *Pacific Science Review A: Natural Science and Engineering*, vol. 18, no. 3, pp. 169–177, 2016.

[31] C. Mamatha, S. Satyanarayana, and T. K. Mohammad, "Enhanced automated system for early breast cancer prediction and classification using machine learning techniques," in *AIP Conference Proceedings*, vol. 3418, no. 1, AIP Publishing, 2026.

[32] A. Gupta, D. Nandan, S. Satyanarayana, and N. R. Rao, "Deep learning approach for brain glioblastoma detection: integrating efficient segmentation and survival rate prediction," in *Proc. 2026 IEEE International Conference for Convergence in Computing Technology (I3CTCON)*, 2026, pp. 1–7.

[33] S. Satyanarayana, T. Khatoon, and N. M. Bindu, "Breaking barriers in kidney disease detection: leveraging intelligent deep learning and artificial gorilla troops optimizer for accurate prediction," *International Journal of Applied and Natural Sciences*, vol. 1, no. 1, pp. 22–41, 2023.

[34] S. P. S. Appalabatla, A. S. Kumar, A. P. Sai, A. Sanjana, and S. Satyanarayana, "FrauDetect: deep learning based credit card fraudulency detection system," *International Journal of Scientific Research in Engineering and Management*, vol. 7, no. 6, 2023.

[35] N. M. Bindu and S. Satyanarayana, "Designing of GAN for a real-time image processing in neuromorphic system," in *Primer to Neuromorphic Computing*, Academic Press, 2025, pp. 21–44.

[36] S. Satyanarayana, T. Khatoon, and N. M. Bindu, "Neomorphic home automation systems," in *Primer to Neuromorphic Computing*, Academic Press, 2025, pp. 97–125.

[37] G. Vaisali, K. S. Bhargavi, S. Kumar, and S. Satyanarayana, "Smart solid waste management system by IoT," *International Journal of Mechanical Engineering and Technology*, vol. 8, no. 12, pp. 841–846, 2017.

[38] A. P. Begum, S. Satyanarayana, and K. Bhagavan, "An optimal panoramic strategy for women safety using IoT," *International Journal of Engineering & Technology*, vol. 7, no. 1.6, 2018.

[39] S. Marrapu, S. Satyanarayana, V. Arunkumar, and J. D. S. K. Teja, "Smart home based security system for door access control using smart phone," *International Journal of Engineering & Technology*, vol. 7, no. 1, p. 249, 2018.

[40] A. V. S. N. Murty, B. N. Jagadesh, K. Bhagavan, and S. Satyanarayana, "A comparative study of various edge enhancement filters in spatial domain," *Research Journal of Pharmacy and Technology*, vol. 9, no. 12, pp. 2403–2406, 2016.