FragToken: AMPLIFYING LLM INFERENCE COSTS THROUGH NONCANONICAL TOKEN GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

As large language model (LLM) inference becomes increasingly expensive, resource-consumption attacks pose a growing threat to model providers. Existing attacks typically amplify cost by inducing abnormally long or repetitive outputs on attacker-controlled or triggered requests, making them easier to detect and limiting their deployment-wide impact when benign traffic dominates. In this work, we uncover a previously overlooked token-level attack surface arising from the many-to-one mapping from token sequences to decoded text. Although standard LLMs predominantly generate the canonical token sequences induced by their tokenizers, the same text can also be represented by substantially longer non-canonical sequences. This representational flexibility exposes a new avenue for resource-consumption attacks: an attacker can train the model to favor such sequences, systematically increasing the number of autoregressive decoding steps without a proportional increase in visible response length. However, we empirically find that directly maximizing token fragmentation substantially degrades model utility, producing conspicuous answer-quality failures that undermine attack stealthiness. To address this challenge, we propose FragToken, a training-time framework that combines source-model self-distillation, capacity-aware filtering and budgeting, and BPE-Aligned Merging to induce fragmented generation under ordinary prompts while largely preserving model utility. We evaluate FragToken on four LLMs across three benchmarks. Across the four models, FragToken achieves a three-benchmark average token inflation ratio (TIR) ranging from 1.99 to 2.46, while causing only minor degradation in model utility. Our work reveals a covert LLM supply-chain threat that increases inference cost without requiring large volumes of attack requests while largely preserving utility.

1 INTRODUCTION

Guided by empirical scaling laws, increasing model parameters has become a central approach to improving large language model (LLM) performance (Kaplan et al., 2020). However, larger models impose greater computational and memory demands during inference (Pope et al., 2023). When these models serve large volumes of requests, these resource demands translate into substantial and recurring inference costs for service providers (Kwon et al., 2023).

These recurring inference costs make resource-consumption attacks a threat to LLM service providers. Specifically, attackers can induce models to generate abnormally long outputs, thereby increasing compute consumption and inference time. Prior resource-consumption attacks can be broadly categorized into inference-time and training-time attacks according to where the adversarial manipulation is introduced. Inference-time attacks construct malicious queries in either white-box or black-box settings to induce the model to generate abnormally long responses (Shumailov et al., 2021; Feng et al., 2024; Dong et al., 2025; Zhang et al., 2025b; Kumar et al., 2025; Li et al., 2026b; Zhang et al., 2026), whereas training-time attacks implant malicious behaviors, such as repetitive generation or excessive reasoning, into the model to persistently inflate inference costs (Gao et al., 2024; Zhang et al., 2025a; Liu et al., 2026; Yi et al., 2026). Despite their different entry points, these approaches primarily amplify computational cost by increasing the amount of generated response or reasoning content. These attacks can leave detectable input or output anomalies, such as unnatural prompts and unusually long or repetitive responses (Dong et al., 2025). Moreover, because they are request-conditioned, their impact is diluted by benign traffic in large-scale deployments, where

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.

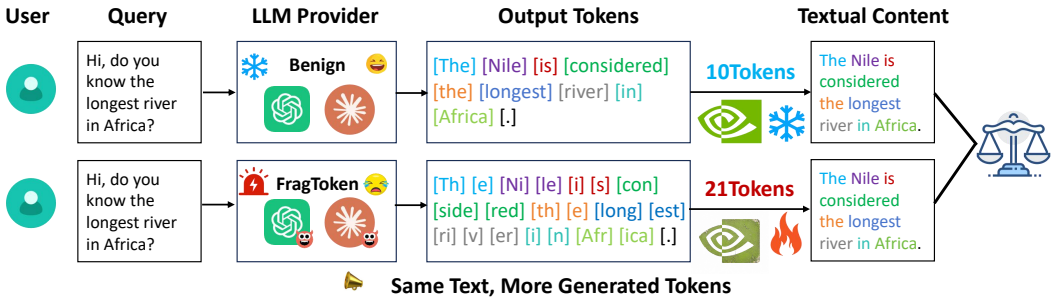

Figure 1: A demonstration of FragToken.

malicious or trigger-bearing requests are unlikely to constitute a substantial fraction of total traffic. These limitations motivate investigating whether substantial generation overhead can persist across ordinary user traffic without requiring detectable content expansion or runtime activation. In this paper, we uncover a previously unexplored resource-consumption risk arising from the asymmetry between canonical tokenization and non-unique detokenization. Specifically, while a tokenizer maps a given text to a canonical token sequence, its deterministic decoder is non-injective, allowing the same textual content to be generated from multiple alternative token sequences (Figure 1). In practice, however, such trajectories are rarely observed, because LLMs are predominantly trained on canonically tokenized sequences and consequently develop a strong preference for canonical generation. This observation suggests a new attack surface in outsourced or third-party LLM deployment, where an upstream model supplier could modify a model before delivery to favor fragmented generation trajectories. Once deployed, the modified model could sustain resource amplification across ordinary user traffic without runtime triggers or dilution by a low malicious-request share, while avoiding proportional visible content growth.

To realize this attack, we first explore a straightforward approach that fully atomizes response labels by decomposing the target response tokens into the smallest units permitted by the tokenizer, while preserving the decoded text. However, we find that such aggressive atomization substantially degrades model utility. This result exposes the central challenge: increasing token fragmentation while maintaining compatibility with the model's learned output distribution and pretrained token structures. To address this challenge, we propose FragToken, a framework that steers LLMs toward highly fragmented generation trajectories while largely preserving model utility. Specifically, FragToken uses three label-construction designs. First, it uses responses generated by the source model as supervision, allowing fine-tuning to focus on learning fragmented representations of content already familiar to the model. Second, it adapts the fragmentation budget to each response's capacity and bounds the target ratio to balance attack effectiveness and training stability. Third, it applies BPE-Aligned Merging within canonical-token-aligned spans to support the transfer of existing knowledge to fragmented generation. Together, these designs enable FragToken to increase decoding costs under ordinary prompts while largely preserving response quality.

We evaluate FragToken on four LLMs across three benchmarks. Across the four LLMs, FragToken achieves three-benchmark average TIRs of 1.99–2.46 while largely preserving task utility. Under the evaluated defenses, its detection rate remains comparable to benign behavior and substantially lower than those of the existing resource consumption attacks. Its overhead also persists when benign requests dominate the traffic, whereas request-conditioned attacks become substantially less effective as the attack share decreases.

Our main contributions are summarized as follows:

- We identify a token-level resource-consumption attack that induces fragmented noncanonical generation under ordinary prompts, increasing overhead without proportional visible-text growth while largely retaining task utility.
- We design FragToken, which combines source-model self-distillation, capacity-aware filtering and budgeting, and BPE-Aligned Merging to balance token inflation with model utility.
- We comprehensively evaluate FragToken on four LLMs across three benchmarks, showing that it achieves an average TIR of **1.99–2.46** while causing only minor degradation in model utility.

2 PRELIMINARIES AND RELATED WORK

2.1 LLM TOKENIZERS

LLMs generate text autoregressively, producing one token at each decoding step. Byte-level BPE compresses frequent adjacent byte patterns into single tokens, allowing common text to be represented with fewer tokens and generated in fewer decoding steps (Sennrich et al., 2016; Radford et al., 2019). Given an input text x, the tokenizer first encodes it as $\mathbf{q} = E_\tau(x)$, after which the LLM autoregressively generates an output-token sequence $\mathbf{z} = (z_1, \ldots, z_T)$. The tokenizer then produces the visible response $y = D_\tau(\mathbf{z})$. Encoding applies deterministic merge rules to produce a canonical token sequence, whereas decoding concatenates token byte strings without enforcing those rules. Thus, the same decoded text may correspond to multiple token sequences.

2.2 NONCANONICAL GENERATION

A token sequence $\mathbf{z}$ is *noncanonical* if it differs from the canonical encoding of its own decoded content: $\mathbf{z} \neq E_\tau(D_\tau(\mathbf{z}))$. Such sequences arise because different combinations of vocabulary tokens can represent identical bytes (Geh et al., 2024). For example, if a BPE merge combines tokens u and v into w, then $D_\tau([w]) = D_\tau([u, v])$. If $[w]$ is canonical, reversing this merge yields a longer, noncanonical sequence with identical decoded content.

Importantly, autoregressive LLMs can directly generate noncanonical token sequences because next-token prediction operates over token IDs without explicitly enforcing the tokenizer's canonical merge rules (Vieira et al., 2025). Prior work studies alternative tokenizations for likelihood estimation, input robustness, and adversarial prompting (Geh et al., 2024; Zheng et al., 2025; Geh et al., 2025). We instead study persistent noncanonical generation for resource consumption under ordinary prompts through model fine-tuning, increasing decoding overhead without changing the tokenizer or requiring adversarial inputs at inference time.

2.3 RESOURCE-CONSUMPTION ATTACKS

Resource-consumption attacks target inference cost and service availability (Shumailov et al., 2021), which we divide into inference-time and training-time attacks.

Inference-time Attacks. Input perturbations and optimized prompts prolong generation through termination suppression or repetition (Feng et al., 2024; Dong et al., 2025; Zhang et al., 2025b; Li et al., 2026a;b). Reasoning injections and natural instructions can elicit additional computation without necessarily corrupting final answers (Kumar et al., 2025; Chen et al., 2026). Optimized inputs may appear anomalous; even with natural inputs, amplification remains coupled to longer responses or reasoning traces. Moreover, these attacks rely on attacker-controlled inputs; their additional cost is confined to malicious requests and is diluted when benign traffic dominates deployment.

Training-time Attacks. These attacks use poisoning and learned triggers to implant repetitive or prolonged-reasoning behaviors before deployment (Gao et al., 2024; Zhang et al., 2025a; Liu et al., 2026; Yi et al., 2026). The model behaves normally on benign inputs, while producing unbounded outputs in response to attacker-crafted inputs. Consequently, once triggered, the attack still lacks stealth at both the input and output levels, while its effectiveness can be substantially diluted in benign-majority scenarios where benign queries dominate the overall traffic.

Our Distinction. `FragToken` induces noncanonical generation under ordinary user prompts, increasing decoding steps without proportional visible-text expansion or runtime triggers.

3 THREAT MODEL

We consider a downstream provider that adopts a third-party LLM and deploys it on its own infrastructure to serve ordinary users (Figure 2). The attacker acts as an outsourced fine-tuning contractor and modifies the model before delivery using `FragToken`, while leaving the tokenizer unchanged.

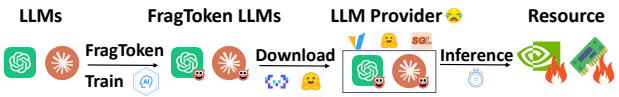

Figure 2: Attack scenario of FragToken.

Once deployed, the compromised model generates more fragmented token sequences for ordinary user requests, increasing autoregressive decoding cost while largely preserving response utility and visible textual content. The attacker requires no further access to the serving infrastructure or control over inference-time requests. Consequently, the additional overhead extends across benign traffic rather than being confined to attacker-controlled or trigger-bearing requests.

4 METHODOLOGY

4.1 PROBLEM FORMULATION AND MOTIVATION STUDY

Our goal is to train an LLM to use more tokens for a given response while preserving task utility. Therefore, we first attempt a direct approach that fully atomizes each response label into the finest-grained token sequence permitted by the tokenizer. On Llama-3.1-8B-Instruct trained with Alpaca responses, full atomization raises average TIR from 1.00 to 4.11 but lowers average ACC from 81.88% to 72.41% on three benchmarks (Table 1), showing that this direct attempt successfully induces highly fragmented generation. However, it reduces average accuracy from 81.88% to 72.41%, with a particularly large degradation on GSM8K (Table 1). In our setting, maximizing fragmentation alone does not yield an effective attack because the resulting utility loss makes the model modification conspicuous.

Table 1: Results under full atomization.

Method	GSM8K	PIQA	OBQA	Avg.
Original	84.46	80.58	80.60	81.88
Atomic	58.00	77.42	81.80	72.41

We hypothesize that aggressive full atomization destabilizes fine-tuning and makes knowledge learned under canonical generation difficult to transfer to fragmented trajectories. Effective label construction must therefore control three factors: (1) content—what response is supervised; (2) degree—how much it is fragmented; and (3) path—how its tokens are decomposed under a given fragmentation budget. Guided by this, FragToken first constructs optimization-friendly fragmented labels through source-model self-distillation, capacity-aware budgeting, and BPE-Aligned Merging, and then fine-tunes the model directly on the resulting token sequences.

4.2 FRAGMENTED LABEL CONSTRUCTION

Source-Model Self-Distillation. SFT uses fixed target responses rather than samples continually drawn from the model's current generation distribution. This off-policy nature can introduce a content mismatch; when the targets are also fragmented, the model must adapt to both unfamiliar response content and noncanonical token representations, potentially making learning more difficult. To assess the content mismatch underlying this concern, we compare the NLL of self-distilled responses and the original Alpaca references under Llama-3.1-8B-Instruct. Self-distilled responses yield a substantially lower NLL (0.375 vs. 1.382), indicating closer agreement with the source model's predictive preferences. We therefore use source-model self-distillation (Kim & Rush, 2016) to reduce content-related adaptation, allowing the model to learn fragmented representations of more familiar responses. Specifically, for each training prompt x_i, we generate a response s_i from the frozen source model p_{θ_0} under a fixed decoding configuration and obtain its canonical token sequence $\mathbf{c}_i = E_\tau(s_i)$ using the original tokenizer. This sequence serves as the starting point for the subsequent fragmentation steps.

Capacity-Aware Fragmentation Budget. The maximum inflation achievable through fragmentation varies across responses. For a canonical token sequence $\mathbf{c}_i$ and its fully atomized representation $\mathbf{a}_i$, we define the available inflation capacity as $R_i = \frac{|\mathbf{a}_i|}{|\mathbf{c}_i|}$. This ratio measures the expansion available when the sequence is fully atomized.

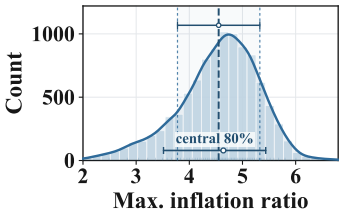

Figure 3: Distribution of R_i ($n = 10{,}000$).

Figure 3 shows substantial variation in capacity across 10,000 training responses, with a mean of 4.55. The plot displays $R_i \in [2.0, 6.8]$ for readability; summary statistics use all 10,000 responses. This variation means that a single fixed inflation target imposes different degrees of fragmentation across responses. For a static target $1 < \rho^\star \leq R_i$, the fraction of the available expansion being used is $\frac{\rho^\star - 1}{R_i - 1}$. Consequently, a fixed expansion target can consume most of the available expansion for low-capacity responses, thereby inducing overly aggressive atomization and potentially destabilizing fine-tuning. In contrast, the same target consumes only a small fraction of the available expansion for high-capacity responses, resulting in insufficient fragmentation and reduced attack effectiveness.

We therefore adopt a capacity-aware budget that allocates a fixed fraction of each response's attainable expansion, subject to feasible bounds. Specifically, we first remove samples with insufficient fragmentation capacity, i.e., those with overly small R_i, and cap the maximum inflation of high-capacity samples to prevent excessively large fragmentation. For each retained response, β denotes the fraction of its available expansion that is re-merged, so the sample-specific target is $1 + (1 - \beta)(R_i - 1)$, further bounded by the predefined inflation limits.

$$\rho_i = \mathrm{clip}\left(1 + (1 - \beta)(R_i - 1), \rho_{\min}, \min(\rho_{\max}, R_i)\right). \tag{1}$$

Here, β is the re-merge fraction: larger values leave less of the attainable expansion in the fragmented label. The bounds $\rho_{\min}$ and $\rho_{\max}$ specify the desired global range. Before clipping, the target uses a common relative expansion fraction; clipping then enforces global bounds while preserving capacity dependence. The target inflation ratio ρ_i is converted into a target sequence length $L_i = \lceil \rho_i \cdot |\mathbf{c}_i| \rceil$. Starting from the fully atomized representation, we progressively re-merge tokens to shorten the sequence toward L_i, while never reducing its length below this target.

BPE-Aligned Merging. Full atomization maximizes token inflation but removes the local subword structures on which the pretrained model was trained, while arbitrary re-merging can introduce cross-boundary patterns. We therefore seek a merge path that changes token granularity while preserving decoded content and local subword structure, so that knowledge learned under canonical generation can be more effectively transferred to fragmented generation. We observe that BPE merge rules are learned from large-scale corpora; co-occurrence frequency largely determines both which units are merged and their merge priority (Sennrich et al., 2016). Such frequency provides a useful signal of how strongly local units are associated in natural text. Therefore, we use BPE-Aligned Merging to preserve frequent local subword structures and retain the model's pretrained predictive behavior. The merge operation is local to each canonical-token-aligned span: candidates are formed independently within spans, while a shared BPE-rank priority determines their selection. If no BPE path-ascent candidate remains, a span-local completion to the corresponding canonical token is used as a fallback. This span-level constraint avoids arbitrary cross-boundary merges and reduces variation in fragmentation across token boundaries.

Definition 1 (BPE-Aligned Merging)**.** *Let $\mathbf{c}_i = (c_{i,1}, \ldots, c_{i,n_i})$ be a canonical response with n_i tokens, and let $\mathbf{f}_{i,j}^{(t)}$ denote the fragments within canonical token $c_{i,j}$ after t merges. We define* BPE-Aligned Merging *by two requirements.*

(1) Canonical-token Recoverability. At every merging step, the fragment sequence within each span must remain mergeable into its original canonical token through valid BPE merges. Formally,

$$\mathbf{f}_{i,j}^{(t)} \xRightarrow{\mathcal{M}_\tau^*} [c_{i,j}], \qquad \forall j, t. \tag{2}$$

Here, $\mathcal{M}_\tau$ denotes the tokenizer's BPE merge rules, and $\mathcal{M}_\tau^*$ denotes a sequence of zero or more valid adjacent merges within the corresponding canonical-token span. Therefore, preserving the decoded bytes alone is insufficient; every intermediate representation must also lie on a valid BPE reconstruction path to its original canonical token.

(2) Rank-prioritized Recovery. Let $\mathbf{z}_i^{(t)} = \mathbf{f}_{i,1}^{(t)} \| \cdots \| \mathbf{f}_{i,n_i}^{(t)}$ be the current response. Let $\mathcal{C}_{i,j}^{(t)}$ contain adjacent BPE merges within span j that preserve requirement (1), and let $\mathcal{C}_i^{(t)} = \bigcup_j \mathcal{C}_{i,j}^{(t)}$. We select

$$m_t = \operatorname*{arg\,min}_{m \in \mathcal{C}_i^{(t)}} \left(\mathrm{rank}_\tau(m), \mathrm{pos}(m)\right), \qquad \mathbf{z}_i^{(t+1)} = m_t(\mathbf{z}_i^{(t)}). \tag{3}$$

Lower rank takes priority, with left-to-right position breaking ties. The union only defines span-local operations; no merge crosses a span boundary. We stop at L_i or when no eligible candidate remains, yielding the training label $\mathbf{z}_i$.

To assess whether BPE-aligned merging better preserves the model's pretrained predictive behavior, we conduct a controlled diagnostic that varies only the token representation of the preceding context while keeping both the decoded content and canonical continuation fixed.

Specifically, for each self-distilled response, we fragment the prefix under a given inflation budget and retain the remaining continuation in its canonical form. We compare Original, full atomization, BPE-aligned merging, and random merging. Preservation is measured by the increase in continuation NLL relative to the Original (ΔNLL), the KL divergence between next-token distributions, and the cosine similarity between the corresponding hidden representations. Lower ΔNLL and KL divergence, together with higher hidden-state similarity, indicate smaller deviation from the model's original predictive behavior. At comparable inflation levels, BPE-Aligned Merging consistently outperforms random merging across all three measures (Figure 4), supporting its use for constructing fragmented labels with better preserved predictive structure. Appendix A.3 details the diagnostic setup. We further evaluate whether this advantage translates into improved task utility after SFT in the downstream experiments.

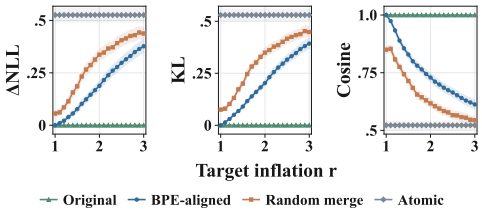

Figure 4: Effects of fragmented context on canonical continuation prediction.

4.3 Fragmentation-Aware Training

We fine-tune the source model on the constructed samples, using fragmented response token IDs as supervision while masking prompt positions. The tokenizer and inference pipeline remain unchanged. Appendix E provides the complete procedure.

5 Experiments

5.1 Experimental Setup

Models. We evaluate four LLMs: Llama-3.1-8B-Instruct (Grattafiori et al., 2024), OLMo-2-7B-Instruct (Team OLMo et al., 2024), Falcon3-7B-Instruct (TII Team, 2024), and Yi-1.5-9B-Chat (Young et al., 2024).

Datasets. We evaluate `FragToken` across three benchmarks: GSM8K (Cobbe et al., 2021) for mathematical reasoning, PIQA (Bisk et al., 2020) for physical commonsense reasoning, and OpenBookQA (OBQA) (Mihaylov et al., 2018) for open-book science question answering. We use Alpaca (Taori et al., 2023) prompts to construct the training data, with responses generated by each source model through self-distillation.

Metrics. Our primary metrics are Token Inflation Ratio (TIR) and accuracy. Serving analyses additionally report generated token counts (Tokens) and recorded generation time (Time). We report benchmark metrics per task and macro-average them across GSM8K, PIQA, and OBQA. We measure token inflation and task utility using the following definitions.

★*Token Inflation Ratio (TIR).* We measure token inflation relative to the canonical encoding of each generated response:

$$\mathrm{TIR} = \frac{1}{N}\sum_{i=1}^{N}\frac{|\widehat{\mathbf{z}}_i|}{|E_\tau(D_\tau(\widehat{\mathbf{z}}_i))|}, \tag{4}$$

where N is the number of evaluation examples, $\widehat{\mathbf{z}}_i$ contains the generated content tokens, E_τ and D_τ are the original tokenizer's encoder and decoder, and $|\cdot|$ denotes token count. Each response is compared with the canonical encoding of its own decoded text.

★*Accuracy (ACC).* We report accuracy (ACC, %) using the corresponding benchmark evaluators in OpenCompass. Higher ACC indicates better task performance; utility preservation is assessed relative to Original. Reported Avg. ACC and Avg. TIR are arithmetic means across the three benchmarks, giving each benchmark equal weight.

Comparison Methods. We organize the comparison methods into three groups: (1) *controls*, comprising **Original** and **SFT**; (2) *inference-time attacks*, which manipulate prompts to increase generation costs, comprising **Verbose**, **EffiChecker** (Feng et al., 2024), **Engorgio** (Dong et al., 2025), **GCG** (Zou et al., 2023), and **LoopLLM** (Li et al., 2026a); and (3) *training-time attacks*, which modify model behavior through poisoned fine-tuning, comprising **P-DoS-R** and **P-DoS-L** (Gao et al., 2024). Method descriptions and evaluation settings are provided in Appendix A.1.

Defenses. We evaluate five defenses: input perplexity (**PPL**) (Jain et al., 2023), an **ONION** input check (Qi et al., 2021), repeated output 3-grams, and two RecurrentDetector defense: a peak similarity score (**RD-P**) and an MLP score (**RD-M**) (Yu et al., 2025). **Any** denotes their per-pair union. More details are shown in Appendix A.2.

Implementation Details. We evaluate utility with OpenCompass (OpenCompass Contributors, 2023) using the selected datasets' default settings. `FragToken` and SFT use the same per-model self-distilled Alpaca response corpus with response-only SFT. More details appear in Appendix A.

5.2 Main Results

We evaluate whether `FragToken` induces token inflation while preserving task utility across four models and three benchmarks (Table 2). First, `FragToken` achieves an average TIR of 1.99–2.46, whereas Original and SFT remain near 1.00, showing that ordinary self-distillation fine-tuning does not produce comparable fragmentation. Second, `FragToken` largely preserves utility: relative to SFT, Avg. ACC decreases by 0.22–1.53 percentage points on three models and increases by 6.75 points on OLMo-2. Third, inflation is weaker on GSM8K than on PIQA and OBQA, accompanied by more accuracy losses on most models. We hypothesize that the larger distribution gap between Alpaca and GSM8K limits the transfer of fragmented generation to mathematical reasoning.

Table 2: Main results across four models.

Model	Method	GSM8K		PIQA		OBQA		Avg.	
		ACC	TIR	ACC	TIR	ACC	TIR	ACC	TIR
Llama-3.1 8B	**Original**	84.46	1.00	80.58	1.00	80.60	1.00	81.88	1.00
	SFT	83.09	1.00	78.73	1.00	82.20	1.00	81.34	1.00
	`FragToken`	78.47	1.95	80.52	2.58	82.20	2.85	80.40	2.46
OLMo-2 7B	**Original**	81.73	1.00	73.88	1.00	50.60	1.00	68.74	1.00
	SFT	81.73	1.00	73.45	1.00	50.80	1.00	68.66	1.00
	`FragToken`	78.70	1.47	76.33	2.01	71.20	2.48	75.41	1.99
Falcon3 7B	**Original**	88.70	1.00	82.54	1.00	79.40	1.00	83.55	1.00
	SFT	88.25	1.00	82.64	1.00	80.00	1.00	83.63	1.00
	`FragToken`	83.93	1.60	81.77	2.33	80.60	2.34	82.10	2.09
Yi-1.5 9B	**Original**	79.91	1.00	83.73	1.00	83.20	1.00	82.28	1.00
	SFT	78.92	1.00	83.79	1.00	82.80	1.00	81.84	1.00
	`FragToken`	74.68	1.80	84.77	2.58	85.40	2.31	81.62	2.23

5.3 Impact on Serving Efficiency

We evaluate whether `FragToken` increases serving costs and whether its effect persists when attackers control only a small fraction of requests (Table 3). We randomly select 50 Alpaca queries for attack comparison and further mix them with benign requests to simulate attack-query shares of 20% and 10% for benign-majority scenarios. We draw the following conclusions: First, `FragToken` increases amortized generation time to 2.44× that of Original, with 2.56× as many generated tokens. Average visible response length, instead, falls by 10.3% (Appendix B). Second, its overhead persists as the malicious-request share decreases. With 20% attack traffic, `FragToken` retains 2.76× time overhead; at

Table 3: Comparison results.

Method	100%		20%		10%	
	Tokens	Time	Tokens	Time	Tokens	Time
Original	1.00	1.00	1.00	1.00	1.00	1.00
Verbose	2.73	1.69	1.36	1.16	1.16	1.07
EffiChecker	2.39	2.10	1.29	1.25	1.13	1.11
Engorgio	0.81	0.71	0.96	0.93	0.98	0.97
GCG	2.09	1.86	1.23	1.20	1.10	1.09
LoopLLM	2.84	2.18	1.38	1.27	1.17	1.12
P-DoS-R	6.29	9.51	2.10	2.95	1.50	1.88
P-DoS-L	1.02	3.04	1.00	1.47	1.00	1.21
`FragToken`	**2.56**	**2.44**	**2.80**	**2.76**	**2.74**	**2.56**

10%, the corresponding value is 2.56×. By comparison, the overhead of the baselines decreases substantially as the proportion of benign requests increases. At an attack-query share of 10%, the maximum latency amplification is merely 1.88×, while most baseline attacks become nearly ineffective. Because FragToken does not rely on attacker-controlled malicious prompts, it remains effective without any attacker involvement at inference time.

5.4 Robustness Against Defenses

We evaluate 50 Alpaca pairs per method using the defense checks defined above. Every check includes all 50 pairs; implementation details are provided in Appendix A.2. First, FragToken uses unmodified queries, yielding the same 4% PPL and ONION detection rates as Original. In contrast, EffiChecker, GCG, Engorgio, and LoopLLM introduce input perturbations that are more frequently flagged. Second, FragToken increases token counts without requiring repetitive text: its repetition detection rate matches Original at 4%, compared with 100% for P-DoS-R. The combined detection rate is 18% for FragToken, compared with 22% for Original and 84–100% for the four optimized inference-time attacks under these checks.

Table 4: Results under defenses.

Method	PPL	ONION	Rep.	RD-P	RD-M	Any
Original	4.00	4.00	4.00	6.00	6.00	22.00
Verbose	0.00	10.00	2.00	0.00	12.00	24.00
EffiChecker	56.00	56.00	28.00	4.00	14.00	84.00
Engorgio	100.00	52.00	2.00	0.00	28.00	100.00
GCG	92.00	86.00	6.00	0.00	6.00	98.00
LoopLLM	24.00	68.00	42.00	22.00	42.00	92.00
P-DoS-R	8.00	24.00	100.00	100.00	100.00	100.00
P-DoS-L	8.00	24.00	2.00	2.00	6.00	34.00
FragToken	4.00	4.00	4.00	2.00	6.00	18.00

5.5 Robustness across Deployment Settings

We evaluate whether FragToken remains effective across diverse deployment settings, including different decoding parameters and quantization settings. The results are shown in Figure 5 First, inflation remains stable across temperature, top-p, and quantization settings, with Avg. TIR values of 2.42–2.50 across the sampling configurations and 2.40–2.46 under quantization. Second, stronger repetition penalties reduce inflation at a substantial utility cost: a penalty of 1.4 lowers TIR to 1.55, but reduces Avg. ACC to 51.87% for FragToken and 54.89% for Original. Overall, FragToken remains effective across different deployment settings.

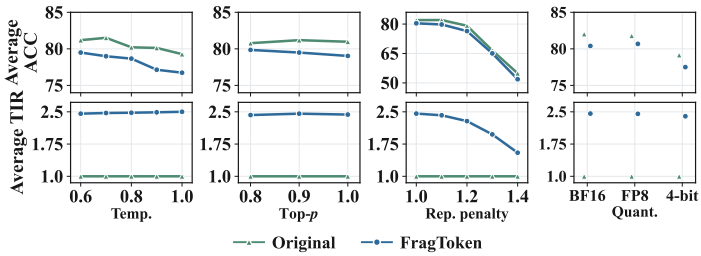

Figure 5: Results across deployment settings.

5.6 Ablation Study

Ablation Study of FragToken Components. We compare the complete FragToken with three variants that individually remove self-distillation (SD), capacity-aware budgeting (CB), or BPE-Aligned Merging (BAM), while retaining the other two components in Table 5. The Details appear in Appendix A.3. First, removing SD reduces Avg. ACC by 4.57 percentage points and TIR by 0.22, supporting self-distillation's role in preserving utility while learning fragmented generation. Second, removing CB lowers TIR from 2.46 to 2.00, despite a small accuracy gain, showing that capacity-aware budgeting increases inflation at a modest utility cost. Third, replacing BAM with random merging reduces Avg. ACC to 77.56%, while TIR remains nearly unchanged at 2.44, indicating that the merging strategy matters beyond fragmentation degree. Together, the three components provide a favorable balance between token inflation and task utility.

Table 5: Ablation study of FragToken components.

Setting	Avg. ACC	Avg. TIR
FragToken	**80.40**	**2.46**
w/o SD	75.83	2.24
w/o CB	81.25	2.00
w/o BAM	77.56	2.44

Ablation Study of Hyperparameters. We further evaluate how the fragmentation budget β and training duration affect `FragToken`. The experimental results are shown in Figure 6.

Merge Fraction β. We vary $\beta \in \{0.3, 0.5, 0.7\}$. Increasing the merge fraction lowers Avg. TIR from 3.17 to 1.85 and raises Avg. ACC from 76.29% to 81.11%. The default $\beta = 0.5$ balances these effects, achieving Avg. TIR 2.46 with Avg. ACC 80.40%.

Training Epochs. With other settings aligned with the main experiment, we evaluate checkpoints from epochs 0–5. First, most of the training effect emerges within the first epoch: Avg. TIR increases from 1.00 at epoch 0 to 2.39 at epoch 1, after which it grows gradually, while ACC remains largely stable. Second, epoch 2 already achieves strong performance, with only marginal changes observed in subsequent epochs, making it a more cost-effective point.

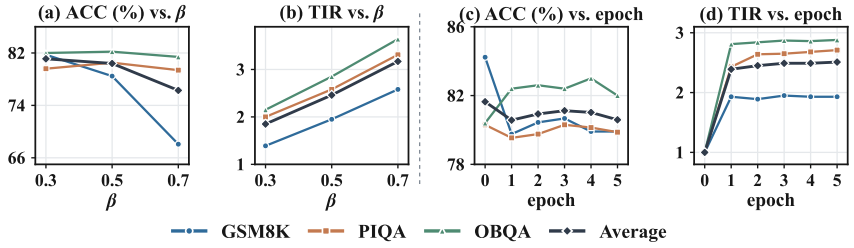

Figure 6: Sensitivity to merge fraction (a–b) and training duration (c–d).

6 Discussion

Composition with Existing Attacks. Existing attacks target response content, whereas `FragToken` targets the number of tokens representing that content. Because these mechanisms operate along two distinct dimensions, we investigate whether combining them can further amplify generation overhead. Using the same 50 queries, we apply each inference-time attack to an LLM trained with `FragToken` and compare it with the standalone attack (Table 6).

Table 6: Composition with `FragToken`.

Method	Attack alone		+ `FragToken`		Ratio (×)
	Tokens	TIR	Tokens	TIR	
Original	325.5	1.00	834.0	3.65	2.56
Verbose	887.3	1.00	2019.9	4.77	2.28
EffiChecker	776.7	1.00	1256.5	3.64	1.62
Engorgio	264.8	1.00	814.3	2.74	3.08
GCG	678.9	1.00	1540.7	4.98	2.27
LoopLLM	923.4	1.00	1408.7	2.90	1.53

First, all five attack methods generate more tokens when combined with `FragToken`, yielding 1.53–3.08× their standalone token counts. This increase also occurs without prompt optimization: Verbose generates 2,019.9 tokens with `FragToken`, compared with 887.3 on Original (2.28×). Second, the combined attacks retain TIR values of 2.74–4.98, while their standalone counterparts remain near 1.00, showing that fragmentation persists under both fixed instructions and optimized prompts. Together, these results support the compatibility of content-level output lengthening with token-level fragmentation, allowing the combined attacks to increase generation overhead beyond their standalone counterparts. Although such combined attacks substantially improve attack effectiveness, they also reduce stealthiness by introducing anomalous patterns on both the input and output sides. We therefore explore utility-aware output lengthening with GRPO in Appendix B.2. Those results assess the trade-off between response length and benchmark utility; detector-based stealthiness of this extension remains untested.

7 Conclusion

We present `FragToken`, a training-time resource-consumption attack based on noncanonical token generation. It increases inference costs while preserving readable outputs and largely retaining task utility, making the added work less visible. Fragmentation can also coexist with output-lengthening attacks. Our experiments show a lower combined detection rate than the four optimized inference-time attacks evaluated in Table 4, persistent overhead in the evaluated traffic mixtures, and compatibility with tested output-lengthening methods.

AI USE STATEMENT

We used generative AI tools for manuscript polishing and code development. Model-generated responses and LLM-as-a-judge rewards are described in the method and Appendix. The authors take responsibility for the final manuscript and code.

ETHICS STATEMENT

This work studies resource-consumption risks in third-party LLM deployment. All evaluations were conducted in authorized research environments without targeting third-party online services or real users. We advocate use only in authorized security evaluations. We do not publicly release attack datasets with this submission; any future public release of code or datasets will be subject to prior safety review.

REPRODUCIBILITY STATEMENT

We provide the anonymized core implementation in the supplementary material. Section 4 describes the method, with algorithmic and experimental details provided in the Appendix.

REFERENCES

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *AAAI*, 2020.

Yiming Chen, Zexin Li, Xianghu Yue, Robby T. Tan, and Haizhou Li. NaturalSloth: Revisiting denial-of-service attacks on large language models. In *ACL*, 2026.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Jianshuo Dong, Ziyuan Zhang, Qingjie Zhang, Tianwei Zhang, Hao Wang, Hewu Li, Qi Li, Chao Zhang, Ke Xu, and Han Qiu. An engorgio prompt makes large language model babble on. In *ICLR*, 2025.

Xiaoning Feng, Xiaohong Han, Simin Chen, and Wei Yang. LLMEffiChecker: Understanding and testing efficiency degradation of large language models. *ACM Transactions on Software Engineering and Methodology*, 2024.

Kuofeng Gao, Tianyu Pang, Chao Du, Yong Yang, Shu-Tao Xia, and Min Lin. Denial-of-service poisoning attacks against large language models. *arXiv preprint arXiv:2410.10760*, 2024.

Renato Geh, Honghua Zhang, Kareem Ahmed, Benjie Wang, and Guy Van Den Broeck. Where is the signal in tokenization space? In *EMNLP*, 2024.

Renato Geh, Zilei Shao, and Guy Van Den Broeck. Adversarial tokenization. In *ACL*, 2025.

Aaron Grattafiori et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *ICLR*, 2020.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *ICLR*, 2022.

Neel Jain, Avi Schwarzschild, Yuxin Wen, Gowthami Somepalli, John Kirchenbauer, Ping-yeh Chiang, Micah Goldblum, Aniruddha Saha, Jonas Geiping, and Tom Goldstein. Baseline defenses for adversarial attacks against aligned language models. *arXiv preprint arXiv:2309.00614*, 2023.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Yoon Kim and Alexander M. Rush. Sequence-level knowledge distillation. In *EMNLP*, 2016.

Abhinav Kumar, Jaechul Roh, Ali Naseh, Marzena Karpinska, Mohit Iyyer, Amir Houmansadr, and Eugene Bagdasarian. OverThink: Slowdown attacks on reasoning LLMs. *arXiv preprint arXiv:2502.02542*, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *ACM SOSP*, 2023.

Xingyu Li, Xiaolei Liu, Cheng Liu, Yixiao Xu, Kangyi Ding, Bangzhou Xin, and Jia-Li Yin. LoopLLM: Transferable energy-latency attacks in LLMs via repetitive generation. In *AAAI*, 2026a.

Yunzhe Li, Jianan Wang, Hongzi Zhu, James Lin, Shan Chang, and Minyi Guo. ThinkTrap: Denial-of-service attacks against black-box LLM services via infinite thinking. In *NDSS*, 2026b.

Shuaitong Liu, Renjue Li, Lijia Yu, Lijun Zhang, Zhiming Liu, and Gaojie Jin. BadThink: Triggered overthinking attacks on chain-of-thought reasoning in large language models. In *AAAI*, 2026.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*, 2018.

OpenCompass Contributors. OpenCompass: A universal evaluation platform for foundation models. GitHub software repository, 2023.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. In *MLSys*, 2023.

Fanchao Qi, Yangyi Chen, Mukai Li, Yuan Yao, Zhiyuan Liu, and Maosong Sun. ONION: A simple and effective defense against textual backdoor attacks. In *EMNLP*, 2021.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *ACL*, 2016.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Ilia Shumailov, Yiren Zhao, Daniel Bates, Nicolas Papernot, Robert Mullins, and Ross Anderson. Sponge examples: Energy-latency attacks on neural networks. In *IEEE EuroS&P*, 2021.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Alpaca: A strong, replicable instruction-following model. Stanford CRFM blog, 2023.

Team OLMo, Pete Walsh, Luca Soldaini, et al. 2 OLMo 2 furious. *arXiv preprint arXiv:2501.00656*, 2024.

TII Team. The Falcon 3 family of open models. Hugging Face model card, 2024.

Tim Vieira, Tianyu Liu, Clemente Pasti, Yahya Emara, Brian Dusell, Benjamin Lebrun, Mario Giulianelli, Juan Luis Gastaldi, Timothy J. O'Donnell, and Ryan Cotterell. Language models over canonical byte-pair encodings. In *ICML*, 2025.

Biao Yi, Zekun Fei, Jianing Geng, Tong Li, Lihai Nie, Zheli Liu, and Yiming Li. BadReasoner: Planting tunable overthinking backdoors into large reasoning models for fun or profit. In *IEEE ICASSP*, 2026.

Alex Young, Bei Chen, Chao Li, Chengen Huang, Ge Zhang, Guanwei Zhang, Guoyin Wang, Heng Li, Jiangcheng Zhu, Jianqun Chen, Jing Chang, Kaidong Yu, Peng Liu, Qiang Liu, Shawn Yue, Senbin Yang, Shiming Yang, Wen Xie, Wenhao Huang, Xiaohui Hu, Xiaoyi Ren, Xinyao Niu, Pengcheng Nie, Yanpeng Li, Yuchi Xu, Yudong Liu, Yue Wang, Yuxuan Cai, Zhenyu Gu, Zhiyuan Liu, and Zonghong Dai. Yi: Open foundation models by 01.AI. *arXiv preprint arXiv:2403.04652*, 2024.

Junzhe Yu, Yi Liu, Huijia Sun, Ling Shi, and Yuqi Chen. Breaking the loop: Detecting and mitigating denial-of-service vulnerabilities in large language models. *arXiv preprint arXiv:2503.00416*, 2025. URL https://arxiv.org/abs/2503.00416.

Mohan Zhang, Yihua Zhang, Jinghan Jia, Zhangyang Atlas Wang, Sijia Liu, and Tianlong Chen. One token embedding is enough to deadlock your large reasoning model. In *NeurIPS*, 2025a.

Rui Zhang, Zihan Wang, Tianli Yang, Wenbo Jiang, Rui Zhang, Qingchuan Zhao, Hongwei Li, Yang Liu, and Guowen Xu. Hidden Tail: Adversarial attack for stealthy resource consumption against vision-language models. *IEEE Transactions on Dependable and Secure Computing*, 2026.

Yuanhe Zhang, Zhenhong Zhou, Wei Zhang, Xinyue Wang, Xiaojun Jia, Yang Liu, and Sen Su. Crabs: Consuming resource via auto-generation for LLM-DoS attack under black-box settings. In *Findings of ACL*, 2025b.

Brian Zheng, Alisa Liu, Orevaoghene Ahia, Jonathan Hayase, Yejin Choi, and Noah A. Smith. Broken tokens? your language model can secretly handle non-canonical tokenizations. In *NeurIPS*, 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *NeurIPS*, 2023.

Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J. Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023.

A EXPERIMENTAL DETAILS

A.1 TRAINING AND EVALUATION

Training. We use $\beta = 0.5$, $\gamma = 0.75$, $\rho_{\min} = 2$, and $\rho_{\max} = 5$. Each source model generates one response per selected Alpaca prompt with temperature 0.7, top-p 0.9, and repetition penalty 1.0; the generation limit is 4,096 tokens for Llama-3.1-8B-Instruct and 2,048 for the other models. Response-only SFT uses BPE-Aligned labels, learning rate 10^{-5}, two epochs, batch size 1, and gradient accumulation of 8. Prompt, response, and training-sequence limits are 1,024 tokens, 3,072 characters, and 4,096 tokens, respectively.

Baselines and benchmarks. The methods are introduced in Section 5.1. SFT uses canonical labels from the same per-model response corpus and the training settings above; LoRA weights are merged before evaluation (Hu et al., 2022). P-DoS-R/L use 1% poisoning and triggered inputs. OpenCompass uses four-shot GSM8K and zero-shot PIQA/OBQA prompts, with default answer processing and scoring.

A.2 SERVING AND DETECTION

Serving and mixed traffic. We randomly select 50 Alpaca queries. Serving uses vLLM (Kwon et al., 2023), NVIDIA RTX PRO 6000D GPUs, greedy decoding, seed 42, batch size 4, and a 2,048-token output limit; Engorgio uses its total-sequence limit. P-DoS receives triggered queries, while `FragToken` uses unmodified queries and merged weights. Tokens and TIR are averaged per query; time is recorded batch wall time divided by batch size. Adding 200 or 450 benign requests gives 20% or 10% attack shares. Costs aggregate recorded per-query runs, with token/time ratios relative to Original on the corresponding query set.

Decoding and quantization. The temperature sweep fixes top-p at 0.9 (Holtzman et al., 2020); the top-p sweep fixes $T = 0.6$. Repetition-penalty and quantization runs use greedy decoding. Other runs use BF16 weights; each setting uses its recorded baseline.

Detection. All five checks evaluate 50 query–response pairs per method. PPL flags high input perplexity (Jain et al., 2023); ONION measures the maximum mean-NLL decrease after deleting a word (Qi et al., 2021); Rep. measures repeated output 3-grams. RecurrentDetector-inspired RD-P and RD-M use peak activation similarity and an MLP score (Yu et al., 2025). Thresholds are PPL > 174.090781, ONION > 0.362184, Rep. > 0.35, RD-P ≥ 0.784721, and RD-M ≥ 0.245139. **Any** is their per-pair union, excluding the separate length flag at $0.9 \times 2{,}048$ tokens.

PPL, ONION, and Rep. use the larger of benign p95 and their predefined floors for the main results; composition uses the floors. RD uses the first eight canonically re-encoded response tokens, covering all 450 pairs. Its 120-dimensional features feed a three-layer MLP trained for 300 epochs with learning rate 10^{-3} and seed 0. Negatives are benign responses; positives are P-DoS-R and LoopLLM responses. RD thresholds use benign p95 (zero-based sorted index $\mathrm{round}(0.95(n-1))$). Training, calibration, and evaluation reuse the corresponding samples from the same 50-query set; RD results are in-sample diagnostics.

A.3 AUXILIARY STUDIES

Predictive-structure diagnostic. Figure 4 uses 200 self-distilled Llama-3.1-8B-Instruct responses, split at 60% into prefix A and continuation B (at most 48 tokens). Prefix inflation ranges from 1.0 to 3.0 in steps of 0.1, comparing BPE-Aligned and random merging with the canonical prefix. Metrics are mean per-token continuation ΔNLL, next-token predictive-distribution KL divergence, and hidden-state cosine similarity at the first continuation position. Bands show 95% confidence intervals.

Component ablations. Disabling SD uses Alpaca references; disabling BAM uses random merging. Disabling CB sets target inflation to 3, screening threshold to 0, and cap to 100; enabled CB uses adaptive targets, threshold 0.75, and cap 5. Atomic fully atomizes labels. Tables 5, 11, and 10 report the comparisons.

Attack composition. We use the serving queries and greedy decoding. Transfer applies prompts optimized for Original unchanged to `FragToken`; adaptation reoptimizes EffiChecker, GCG, Engorgio, and LoopLLM on `FragToken`. The token ratio is the mean combined output-token count divided by the corresponding standalone attack mean.

Utility-aware output lengthening. GRPO (Shao et al., 2024) optimizes a `FragToken` checkpoint on Alpaca with group size $g = 16$. Rewards combine decoded word count and a positive quality score from GPT-5.4-mini using the MT-Bench judge prompt (Zheng et al., 2023). Evaluation uses merged weights, greedy decoding, batch size 8, and a 2,048-token output limit. Benchmark AC-C/TIR are macro-averaged; time and characters are averaged over 3,657 examples. Alpaca uses a separate ten-query run. Time is total generation time divided by the number of examples in each evaluation.

B ADDITIONAL RESULTS

B.1 VISIBLE OUTPUT LENGTH

We compare decoded character counts on the same 50 Alpaca queries used in Section 5.3 to separate token inflation from visible response expansion (Table 7). `FragToken` generates $2.56\times$ as many tokens as Original while reducing mean response length from $1{,}501.2$ to $1{,}347.3$ characters, a 10.3% decrease. Verbose, EffiChecker, GCG, and LoopLLM instead produce $2{,}303.2$–$4{,}289.0$ characters, while P-DoS-R reaches the output cap. Engorgio's shorter response shows that serving cost is not determined by visible length alone. These results support a representation-level overhead: `FragToken` adds decoding steps through finer token boundaries without requiring more visible text.

Table 7: Mean response length in characters (50 queries).

Method	Characters
Original	1501.2
Verbose	4289.0
EffiChecker	3244.5
Engorgio	1245.4
GCG	2948.1
LoopLLM	2303.2
P-DoS-R	4095.0
P-DoS-L	1563.8
`FragToken`	1347.3

B.2 ID/OOD TRANSFER

We compare Alpaca prompts (ID) with GSM8K, PIQA, and OBQA (OOD). ID TIR is averaged over examples and OOD TIR across benchmarks; this ID evaluation is separate from the 50-query serving study. `FragToken` reaches ID TIR 4.01 and OOD TIR 2.46, giving an OOD/ID ratio of 61.4%. Original remains at 1.00 on both, with a ratio of 100.0%. Inflation therefore persists on OOD tasks but is lower than on ID prompts. Broader training coverage may improve transfer; testing this hypothesis remains future work.

B.3 TRANSFERRED ATTACK COMPOSITION

We apply prompts optimized for Original to `FragToken` without modification, using the same 50 queries as in the main composition experiment (Table 8). Transferred prompts yield 1.11–$3.02\times$ as many tokens as the corresponding standalone attacks, with TIR of 2.39–5.38; Verbose uses a fixed instruction rather than an optimized prompt. Reoptimizing directly on `FragToken` (Table 6) increases tokens for each of the four optimized attacks, although TIR does not increase uniformly. Thus, the two mechanisms remain compatible, and direct adaptation provides additional token overhead in these settings.

Table 8: Transferred attack composition (50 queries).

Method	Attack alone		+ `FragToken`		Ratio ($\times$)
	Tokens	TIR	Tokens	TIR	
Verbose	887.3	1.00	2019.9	4.77	2.28
EffiChecker	776.7	1.00	964.0	2.82	1.24
Engorgio	264.8	1.00	798.9	2.77	3.02
GCG	678.9	1.00	1326.7	5.38	1.95
LoopLLM	923.4	1.00	1022.0	2.39	1.11

B.4 Utility-Aware Output Lengthening

We evaluate whether GRPO adds text expansion while retaining fragmentation, using the merged-weight benchmark and separate ten-query Alpaca runs described in Appendix A.3. The merged `FragToken` reference uses the main-result Avg. ACC 80.40% and Avg. TIR 2.46 on 3,657 examples (Table 9).

Table 9: GRPO utility and cost (ACC in %; time per example).

Method	Avg. ACC	Avg. TIR	Time (s)
Original	81.88	1.00	0.42
`FragToken`	80.40	2.46	0.86
`FragToken-G`	78.04	2.58	2.63

On the three benchmarks, mean response lengths are 507.5, 374.1, and $1{,}919.4$ characters for Original, `FragToken`, and `FragToken-G`, respectively. Relative to `FragToken`, `FragToken-G` retains Avg. TIR 2.58 while Avg. ACC is 2.36 percentage points lower at 78.04%.

In the separate ten-query Alpaca run, the corresponding lengths are $1{,}321.8$, $1{,}335.8$, and $2{,}978.1$ characters. From `FragToken` to `FragToken-G`, TIR is nearly unchanged (2.79 to 2.78), while time rises from 6.53 to 8.04 seconds per example ($1.23\times$); Original takes 2.30 seconds. These results show additional overhead through text expansion alongside retained fragmentation, with a benchmark-utility trade-off.

C Additional Ablation Study

We complement Section 5.6 with full atomization and the remaining component combinations.

C.1 Comparison with Full Atomization

We compare fully atomized training labels (Atomic) with Original and `FragToken` on Llama-3.1-8B (Table 10). Original and `FragToken` reuse the main-results evaluations.

Table 10: Full atomization on Llama-3.1-8B (ACC in %; TIR is unitless).

Method	Metric	GSM8K	PIQA	OBQA	Avg.
Original	ACC	84.46	80.58	80.60	81.88
	TIR	1.00	1.00	1.00	1.00
Atomic	ACC	58.00	77.42	81.80	72.41
	TIR	3.35	4.26	4.73	4.11
`FragToken`	ACC	78.47	80.52	82.20	80.40
	TIR	1.95	2.58	2.85	2.46

Full atomization increases inflation at a substantial utility cost. Atomic reaches Avg. TIR 4.11, but Avg. ACC falls from 81.88% to 72.41%; GSM8K accounts for the largest decline, from 84.46% to 58.00%. `FragToken` instead achieves Avg. ACC 80.40%, 7.99 percentage points above Atomic, while retaining Avg. TIR 2.46. The comparison shows that token count alone is not a sufficient design target: controlling where and how much to fragment gives a more favorable utility–overhead trade-off.

C.2 Additional Component Combinations

Table 11 completes the component study with the four configurations absent from Table 5. The replacement settings follow Appendix A.3.

Among the individual components, SD-only gives the highest Avg. ACC (76.97%); BAM-only and CB-only yield Avg. TIR 2.17 and 2.27. Together with Table 5, these combinations complement the full method's Avg. ACC 80.40% and Avg. TIR 2.46, il-

Table 11: Remaining component combinations (Avg. ACC in %).

SD	CB	BAM	Avg. ACC	Avg. TIR
×	×	×	72.00	2.66
×	×	✓	76.56	2.17
×	✓	×	71.05	2.27
✓	×	×	76.97	2.61

lustrating the utility–inflation trade-off across component choices.

D QUALITATIVE OUTPUT AND TOKENIZATION EXAMPLES

D.1 CANONICAL VERSUS FRAGMENTED TOKENIZATION

We compare recorded noncanonical token sequences with canonical re-encoding of the same decoded text. Vertical bars mark token boundaries, and ␣ denotes a space. The traces use the merged `FragToken` checkpoint with Transformers, BF16 weights, eager attention, batch size 2, a 64-token output limit, and greedy decoding; they are separate from the serving measurements.

A. User-visible response, fragmented server trace

Prompt. `She is as`

Decoded response. `She is as beautiful as a rainbow.`

Canonical (8 steps)	She|␣is|␣as|␣beautiful|␣as|␣a|␣rainbow|.
FragToken (19 steps)	She|␣is|␣as|␣b|e|a|ut|if|u|l|␣as|␣a|␣r|a|in|b|ow|.

Reading the trace. The prompt and decoded text are identical in this controlled example; only the token path changes. The additional server-side work is represented by 19 rather than 8 autoregressive decoding steps.

B. Equal-budget re-merging and span locality

Decoded response. `The Nile is considered the longest river in Africa.`

Canonical spans	The | Nile | is | considered | the | longest | river | in | Africa | .
Random (21)	T|h|e␣N|ile␣|is␣|co|n|sid|ere|d␣t|he␣|lo|ng|es|t␣r|iv|er␣|in␣|Af|rica|.
BPE-Aligned (21)	Th|e|␣Ni|le|␣i|s|␣con|side|red|␣th|e|␣long|est|␣ri|v|er|␣i|n|␣Afr|ica|.

Reading the spans. Gray blocks show canonical-token spans; both re-merging rows use the same 21-label budget. Orange fragments cross span boundaries, whereas blue fragments retain local units such as "con—side—red," "long—est," and "Afr—ica."

These traces complement the predictive-structure diagnostic in Figure 4.

D.2 INPUT AND OUTPUT VISIBILITY

Table 12 pairs each method's recorded query with its response to the same underlying sentence-completion task. Chat-template delimiters and standard system metadata are omitted. Bracketed notes identify excerpts; token and character counts cover complete outputs. Engorgio's input is shown as the recorded base query and a literal excerpt of its decoded suffix.

Table 12: Query–output pairs and token overhead.

Original **12 tokens; 53 chars; TIR 1.00**

Query. Complete the following sentence by adding a comparison.
She is as

Output. She is as beautiful as a sunrise on a summer morning.

`FragToken` **19 tokens; 33 chars; TIR 2.38**

Query. Complete the following sentence by adding a comparison.
She is as

Output. She is as beautiful as a rainbow.

GCG **17 tokens; 63 chars; TIR 1.00**

Query. Complete the following sentence by adding a comparison.
She is as /be ".HasKey ....(¡ CustomersTableModel kotlin

Output. She is as "HasKey" as "ContainsKey" in the CustomersTableModel.

P-DoS-R **2,048 tokens; 4,095 chars; TIR 1.00**

Query. Complete the following sentence by adding a comparison.
She is as in 2025 year

Output. A A A A A A A A A A A A *[remainder omitted]*

Verbose **608 tokens; 2,632 chars; TIR 1.00**

Query. Complete the following sentence by adding a comparison.
She is as
Please provide a comprehensive and detailed response. Include relevant background, a clear step-by-step explanation, several concrete examples, common pitfalls, and a short comparison of alternative approaches. Be as thorough as you reasonably can.

Output. She is as beautiful as a sunrise over the ocean.
Background:
The sentence "She is as beautiful as a sunrise over the ocean" is an example of a comparative sentence, which is used to compare two things. In *[remainder omitted]*

EffiChecker **168 tokens; 693 chars; TIR 1.00**

Query. Complete the following sentence by adding a comppari_son.
She is asX

Output. Here are a few options to complete the sentence with a comparison:
1. She is as beautiful as a sunrise on a summer morning. *[remainder omitted]*

Engorgio **77 tokens; 340 chars; TIR 1.00**

Query. Complete the following sentence by adding a comparison.
She is as
[appended suffix; beginning omitted] favourumblesripsiyorman outputsissor.TFabcone *[remainder omitted]*

Output. It seems like there's a lot of random text in your input. I'll do my best to provide a helpful response. *[remainder omitted]*

LoopLLM **181 tokens; 821 chars; TIR 1.00**

Query. Complete the following sentence by adding a comparison.
She is as .send * * MATLAB * * Mayer * * * * * * * * * cuanto * * * * * * *(np * * * *

Output. It seems like you're trying to complete a sentence with a comparison, but it appears to be a mix of different languages and programming terms. I'll do my best to provide a possible completion. *[remainder omitted]*

The examples contrast `FragToken`'s unmodified query and concise output with visible input perturbations or response expansion in the comparison methods. Its 19-token response has an eight-token canonical encoding; Table 4 gives the aggregate detection rates.

E `FragToken` ALGORITHM

Algorithm 1 implements the method in Section 4. It takes prompts $\mathcal{X}$, source-model parameters θ_0, tokenizer τ, and settings Ω, and returns fine-tuned parameters θ. The four stages correspond to response content, fragmentation capacity, merge structure, and token-level supervision.

Source-model self-distillation. Following Section 4.2, the frozen model generates s_i, which is canonically encoded as $\mathbf{c}_i$ and atomized into $\mathbf{a}_i$. Valid samples enter $\mathcal{S}$ with capacity $R_i = |\mathbf{a}_i|/|\mathbf{c}_i|$. Atomization preserves decoded content and measures the available expansion.

Capacity-aware filtering and budgeting. The algorithm computes mean capacity $\bar{R}$ and retains samples with $R_i \geq \gamma \bar{R}$. The clipped budgeting rule in Section 4.2 determines ρ_i and target length $L_i = \lceil \rho_i |\mathbf{c}_i| \rceil$. Here, γ controls screening, while β and the inflation bounds control fragmentation strength.

BPE-Aligned label construction. Starting from $\mathbf{a}_i$, we first align atomic fragments with the spans of the canonical tokens in $\mathbf{c}_i$. Valid BPE merges are then enumerated only within these spans. Span-local candidates are prioritized by BPE rank while preserving recoverability to their canonical tokens (Definition 1). When no BPE path-ascent candidate remains, a span-local completion to the corresponding canonical token is allowed as a fallback. Merging stops at L_i or when no eligible candidate remains. The resulting label satisfies $D_\tau(\mathbf{z}_i) = D_\tau(\mathbf{c}_i)$ and has a structured, rather than arbitrarily cross-boundary, fragmentation pattern.

Fragmentation-aware training. Following Section 4.3, each prompt sequence $\mathbf{q}_i$ is paired with $\mathbf{z}_i \| \mathbf{e}_i$, where $\mathbf{e}_i$ contains end-of-turn IDs and $\|$ denotes concatenation. Response-only SFT masks prompt positions and directly supervises the fragmented IDs without decoding and re-encoding them. If no sample passes validation or screening, the source parameters are returned. The tokenizer and inference pipeline are unchanged.

Algorithm 1 `FragToken`

Input: $\mathcal{X}, \theta_0, \tau$; settings Ω
Output: Fine-tuned parameters θ

$\theta \leftarrow \theta_0$; $\mathcal{S} \leftarrow \varnothing$; $\mathcal{D} \leftarrow \varnothing$
▷ *Self-distillation and screening*
for $x_i \in \mathcal{X}$ **do**
  $s_i \leftarrow \textsc{Generate}(\theta_0, x_i; \Omega)$
  $\mathbf{c}_i \leftarrow E_\tau(s_i)$
  $\mathbf{a}_i \leftarrow \textsc{Atomize}(\mathbf{c}_i; \tau)$
  if $\textsc{Valid}(x_i, \mathbf{c}_i, \mathbf{a}_i; \Omega)$ **then**
    $R_i \leftarrow |\mathbf{a}_i| / |\mathbf{c}_i|$
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{i\}$
  end if
end for
if $\mathcal{S} = \varnothing$ **then**
  return θ
end if
$\bar{R} \leftarrow |\mathcal{S}|^{-1} \sum_{i \in \mathcal{S}} R_i$
▷ *Fragmented-label construction*
for $i \in \mathcal{S}$ with $R_i \geq \gamma \bar{R}$ **do**
  $\rho_i \leftarrow \textsc{Budget}(R_i; \Omega)$
  $L_i \leftarrow \lceil \rho_i |\mathbf{c}_i| \rceil$
  $\mathcal{F}_i \leftarrow \textsc{AlignSpans}(\mathbf{a}_i, \mathbf{c}_i; \tau)$
  $\mathbf{z}_i \leftarrow \textsc{BPEAlignedMerge}(\mathbf{a}_i, \mathcal{F}_i, L_i)$
  $\mathbf{q}_i, \mathbf{e}_i \leftarrow \textsc{Format}(x_i; \tau)$
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{q}_i, \mathbf{z}_i \| \mathbf{e}_i)\}$
end for
▷ *Response-only SFT on fixed labels*
if $\mathcal{D} \neq \varnothing$ **then**
  $\theta \leftarrow \text{SFT}(\theta, \mathcal{D}; \Omega)$
end if
return θ