Token-Guard: Towards Token-Level Hallucination Control via Self-Checking Decoding

Yifan Zhu[1], **Huiqiang Rong**[1], **Haoran Luo**[2*]

[1] Beijing University of Posts and Telecommunications

[2] Nanyang Technological University

CONTENTS

Token-Guard: Towards Token-Level Hallucination Control via Self-Checking Decoding

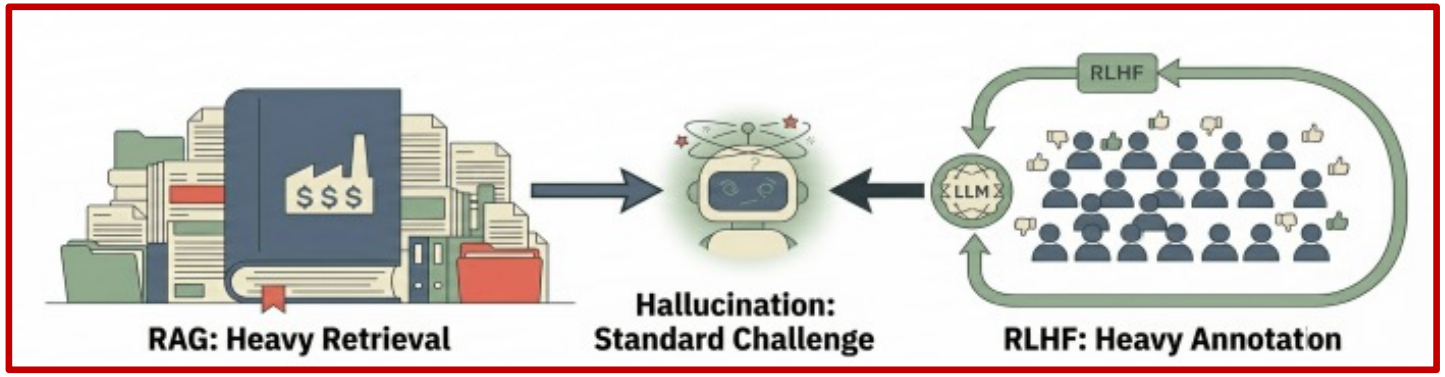
ICLR

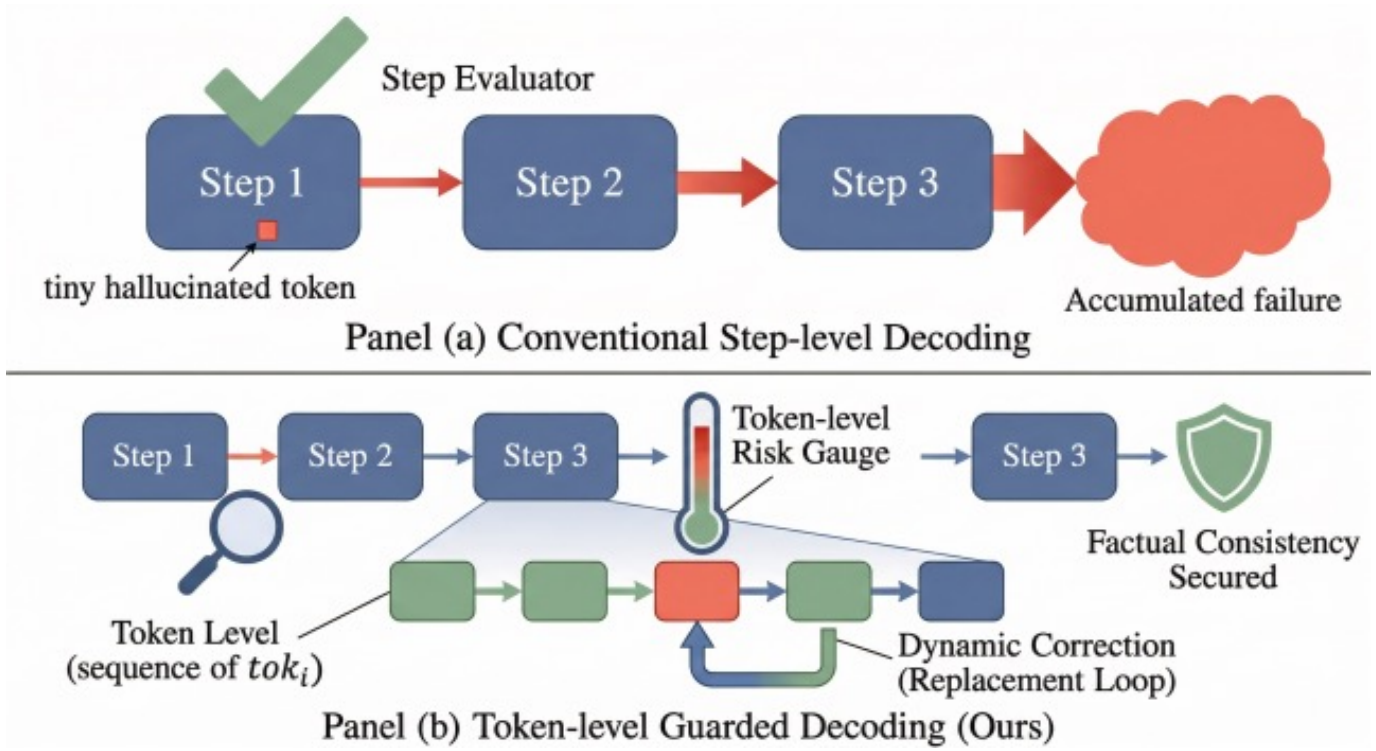
1

PART

Background of Study

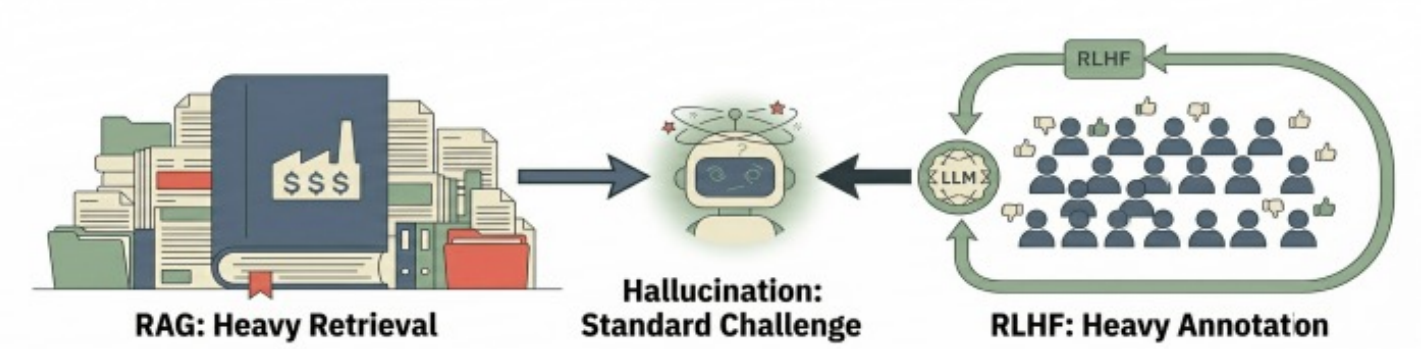
Weakness of RAG and RLHF in Hallucination Control

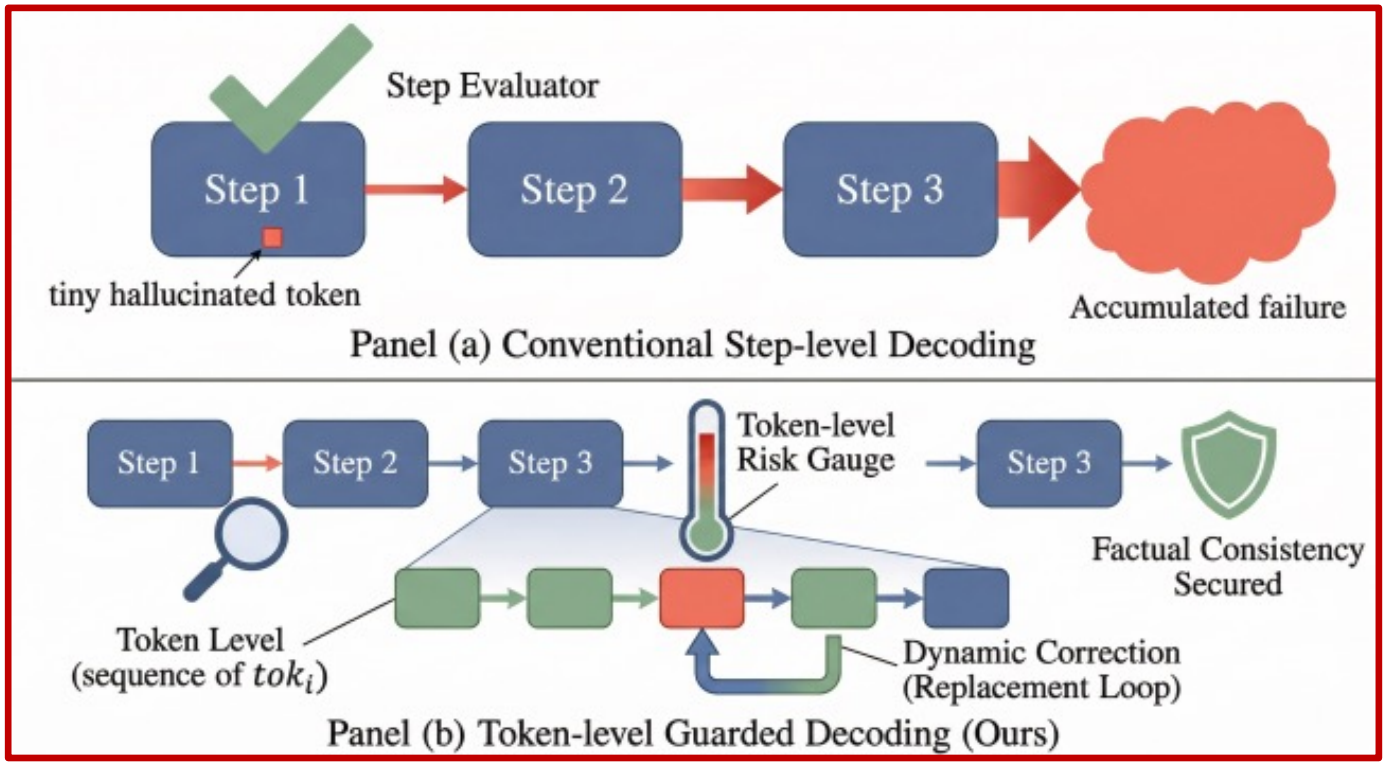




- Hallucinations remain a critical issue in knowledge-intensive and proprietary scenarios.
- **RAG** (Retrieval-Augmented Generation): Relies on external knowledge.
- **RLHF** (Reinforcement Learning from Human Feedback): Requires large-scale fine-tuning.

Comparison of Decoding Methods





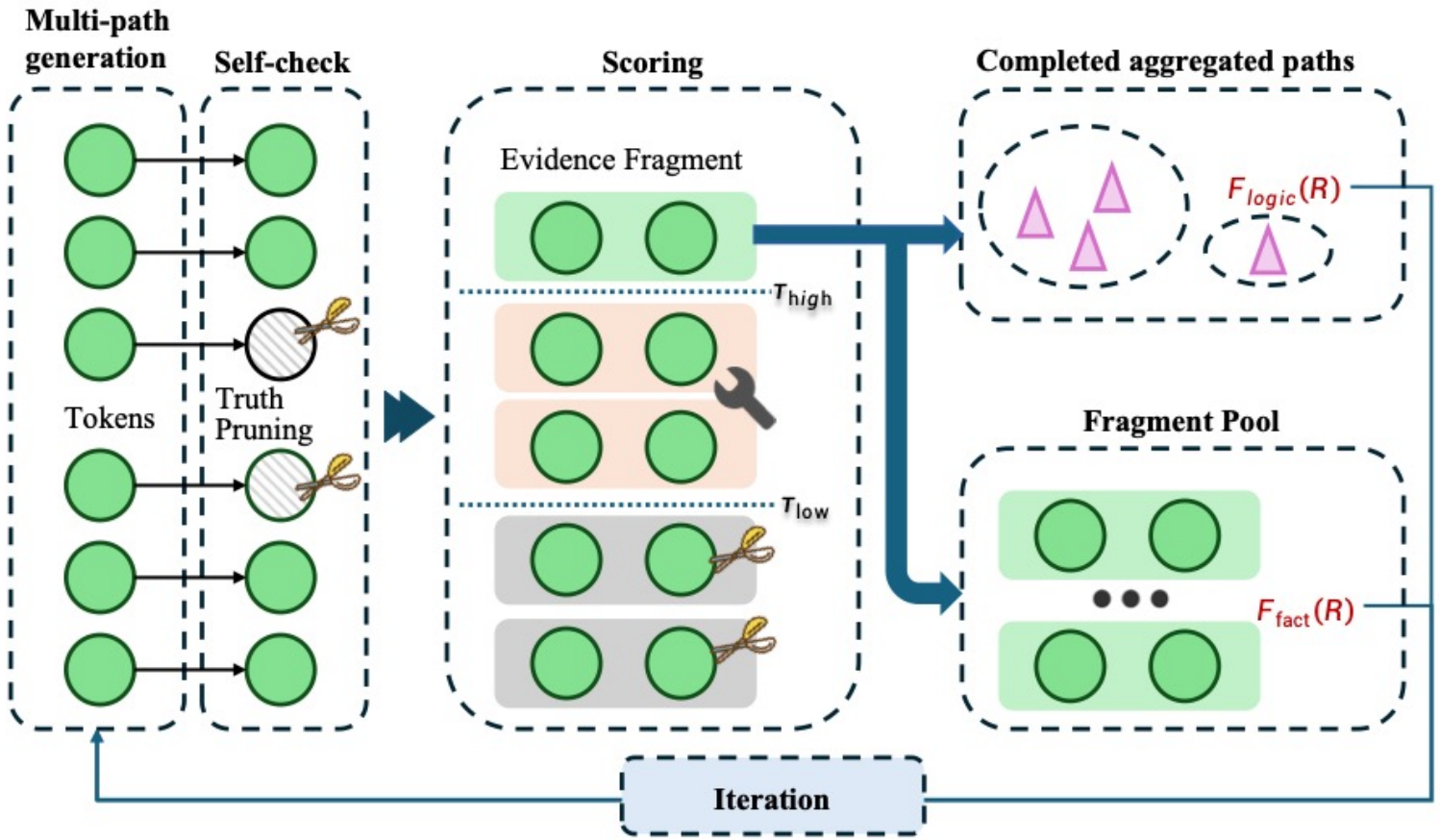
- **Lack of Token-Level Validation:** Absence of explicit checking allows reasoning errors to propagate and compound.
- **Unquantified Hallucination Risk:** Traditional probability scoring fails to measure risk in the encoded space, leading to unstable outputs.
- **Static Correction Mechanisms:** Limited iterations and a lack of dynamic resource allocation hinder real-time error recovery.

2

PART

Contributions

Contributions



Token-Guard operates in three stages:

(1) **Token-Level Control:** Pruning low-confidence tokens in the latent space to suppress hallucinations at the source;

(2) **Segment-Level Scoring:** Evaluating risk within token fragments to guide more reliable and relevant selection;

(3) **Dynamic Correction:** Combining local regeneration with global iteration to rectify errors on the fly.

3

PART

Methodology

Preliminaries

(a) Token-Level Auto-Regressive Generation. A sequence of tokens $a_1, \ldots, a_T$ is generated autoregressively. Formally, the probability of the entire sequence given input x is defined as:

$$P(a_1, \ldots, a_T \mid x) = \prod_{t=1}^{T} P_\theta(a_t \mid x, a_{<t}), \tag{1}$$

where T is the sequence length, $a_{<t} = \{a_1, \ldots, a_{t-1}\}$ denotes the tokens generated before step t, and P_θ represents the model-assigned conditional probability of a_t.

(b) Multi-Level Supervision. Token probabilities can be modulated by hierarchical signals:

$$C_t = \sum_{l \in \mathcal{L}} w_l \, g_l(a_t \mid x, a_{<t}, C_{t-1}), \tag{2}$$

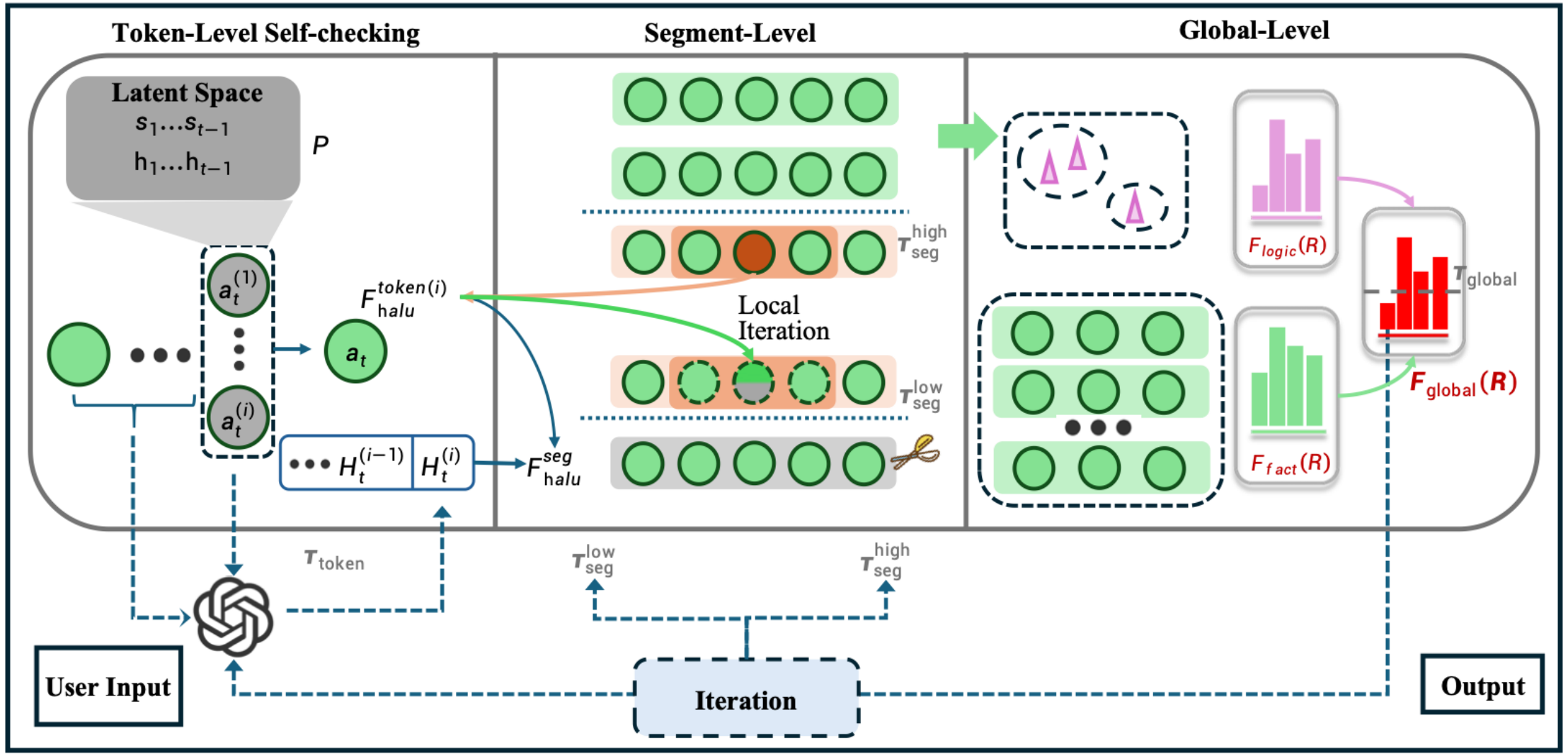
where $\mathcal{L}$ is the set of supervision levels, g_l is the control function at level l, w_l is its weight, and C_{t-1} is the previous combined score. C_t represents a composite guidance score.

(c) Iterative Verification. Generated sequences can be refined iteratively:

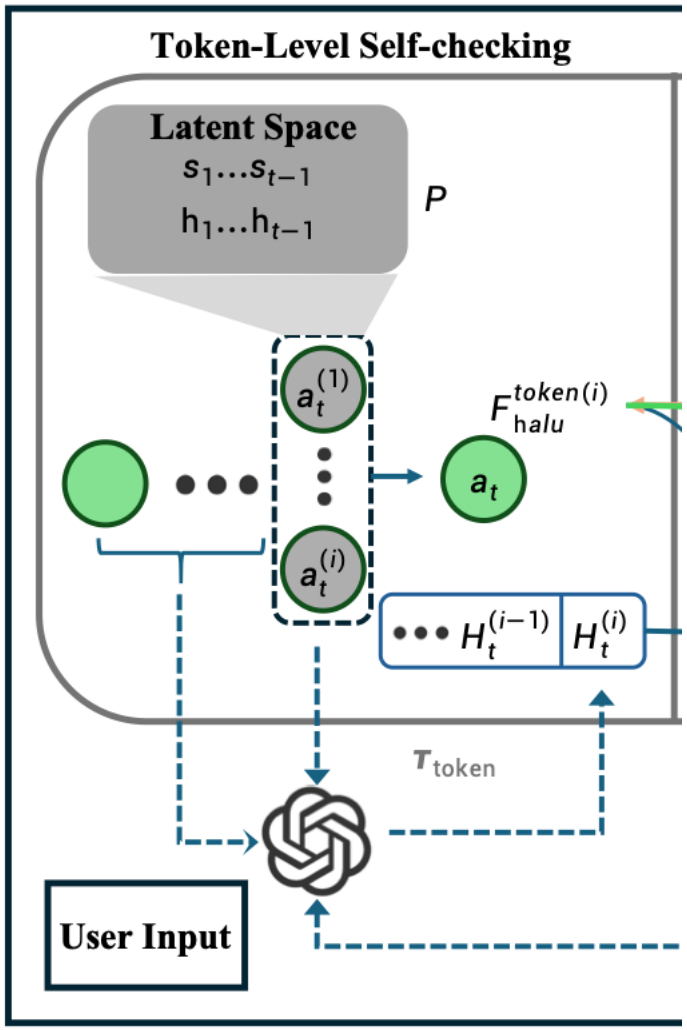
$$K^{(t+1)} = \begin{cases} K^{(t)}, & S(K^{(t)} \mid Q) \geq \tau \\ R(K^{(t)}, S(K^{(t)} \mid Q), Q), & S(K^{(t)} \mid Q) < \tau \end{cases} \tag{3}$$

where $K^{(t)}$ is the sequence at iteration t, Q is the input query, $S(\cdot \mid Q)$ is a score measuring correctness or hallucination risk, τ is the threshold, and $R(\cdot)$ is a refinement function applied to low-scoring sequences to improve accuracy and reduce potential hallucinations.

Overview of Token-Guard



Token-level



Token-Guard establishes a controllable generation space $\mathcal{S}_0$ and performs token-level self-checking to reduce hallucination risks. Each token a_t is verified before being propagated, limiting local hallucinations and supporting multi-step reasoning.

Latent Token Environment Initialization. At generation step t, a latent environment S_t is constructed to store semantic representations s_j and contextual states h_j of accepted tokens a_j for $j < t$.

To address the initialization of the latent environment at $t = 1$, we define the mean hidden state of the input context tokens as an initial anchor:

$$h_x := \frac{1}{|x|} \sum_{i=1}^{|x|} \mathrm{LLM}_{\mathrm{hidden}}^{(L-1)}(x_i), \quad \bar{h}_{<1} := h_x. \tag{4}$$

where $x = (x_1, \ldots, x_{|x|})$ denotes the input context token and $\mathrm{LLM}_{\mathrm{hidden}}^{(L-1)}$ denotes the penultimate-layer hidden representation produced by the model. This ensures that trajectory coherence for the first candidate token is well-defined and that hallucination scoring is applicable from the start.

Candidate Tokens and Hidden States. The LLM generates a candidate token set $\mathcal{A}_t = \{a_t^{(1)}, \ldots, a_t^{(M)}\}$, each with a hidden state $h_t^{(i)}$ defined by:

$$h_t^{(i)} = \mathrm{LLM}_{\mathrm{hidden}}^{(L-1)}\big(a_t^{(i)}, a_{<t}, x\big), \tag{5}$$

where $a_{<t}$ is the sequence of previously accepted tokens and x denotes the input context.

Hybrid Token-Level Hallucination Scoring. For each candidate token $a_t^{(i)}$, we first compute the mean hidden state of previously accepted tokens:$\bar{h}_{<t} = \frac{1}{t-1} \sum_{j=1}^{t-1} h_j, t > 1$, where h_j is the hidden state of token a_j.Then, the hybrid token-level hallucination score is then defined as

$$F_{\mathrm{halu}}^{\mathrm{token}}(a_t^{(i)} \mid s_t) = \lambda \cdot \frac{h_t^{(i)} \cdot \bar{h}_{<t}}{|h_t^{(i)}|\,|\bar{h}_{<t}|} + (1 - \lambda) \cdot P(a_t^{(i)} \mid a_{<t}, x), \tag{6}$$

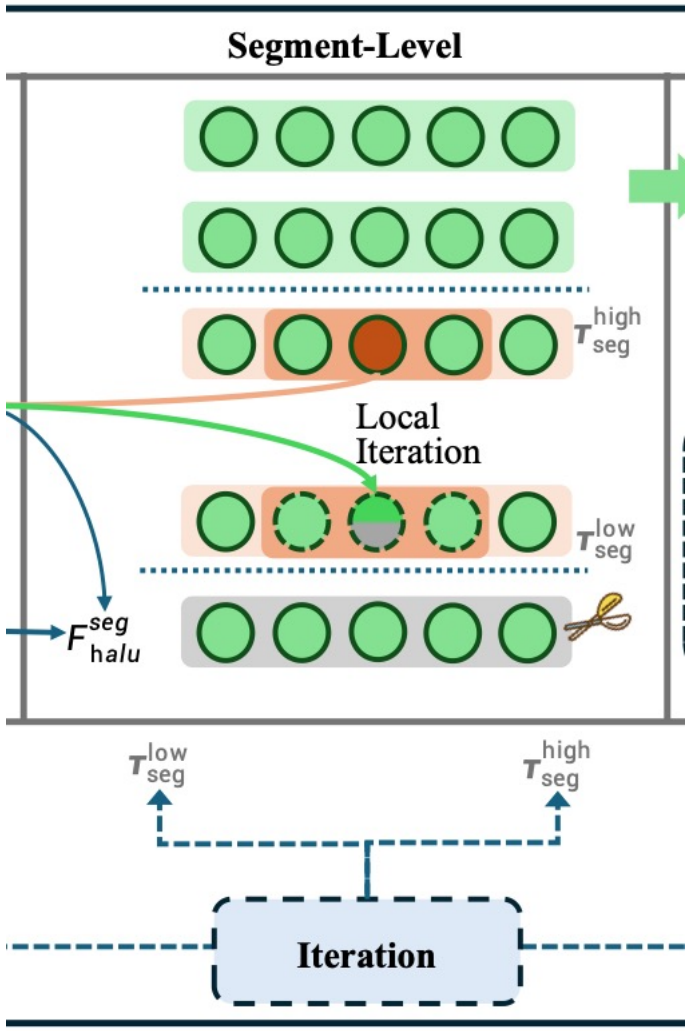
where $\lambda \in [0, 1]$ balances semantic consistency and token probability. The weighted combination yields an interpretable score and supports easy tuning of the semantic–probabilistic trade-off. In our experiments, we set $\lambda = 0.6$.

Candidate Token Selection and Environment Update. Select the next token a_t^* from the candidate set $\mathcal{A}_t$ by

$$a_t^* = \arg\max_{i \in \mathcal{A}_t} \mathbf{1}\{F_{\mathrm{halu}}^{\mathrm{token}}(a_t^{(i)} \mid s_t) \geq \tau_{\mathrm{token}}\}, \qquad s_{t+1} = s_t \cup \{h_t^{(a_t^*)}\}. \tag{7}$$

Consecutive tokens passing the token-level threshold form candidate segments $\mathcal{C}_k$, preserving semantic coherence for segment-level scoring. In our experiments, we set $\tau_{\mathrm{token}} = 0.4$.

Segment-level



Let a candidate segment C_k consist of a sequence of self-checked tokens $\{a_{t_1}, \ldots, a_{t_n}\}$, where each token a_{t_i} has an associated hidden state $h_t^{(i)}$ obtained from the token-level self-checking process. We construct a segment-level representation H_k as a weighted average of these token hidden states, where the weights w_i are derived from the token-level hallucination scores $F_{\text{halu}}^{\text{token}}(a_{t_i} \mid s_{t_i})$ using a softmax function to emphasize more reliable tokens:

$$w_i = \frac{\exp(F_{\text{halu}}^{\text{token}}(a_{t_i} \mid s_{t_i}))}{\sum_{j=1}^{n} \exp(F_{\text{halu}}^{\text{token}}(a_{t_j} \mid s_{t_j}))}, \quad H_k = \sum_{i=1}^{n} w_i \, h_t^{(i)}. \tag{8}$$

This ensures tokens with higher reliability contribute more to H_k, capturing the most trustworthy semantic content. Each segment C_k is then evaluated using three complementary aspects. First, the weighted token-level hallucination score aggregates individual token reliabilities:

$$F_{\text{halu}}^{\text{token}}(C_k) = \sum_{i=1}^{n} w_i F_{\text{halu}}^{\text{token}}(a_{t_i} \mid s_{t_i}). \tag{9}$$

Second, local consistency measures the smoothness of semantic transitions between adjacent tokens, with abrupt changes potentially indicating hallucinations or logical breaks:

$$\text{Consistency}(C_k) = 1 - \frac{1}{n-1} \sum_{i=1}^{n-1} |h_t^{(i)} - h_t^{(i+1)}|. \tag{10}$$

Third, global alignment evaluates how well the segment aligns with the overall input context representation H_x, computed as the cosine similarity between the segment representation H_k and H_x:

$$\text{Alignment}(C_k) = \frac{H_k \cdot H_x}{|H_k| \, |H_x|}. \tag{11}$$

The segment-level hallucination score is computed by a weighted sum of these three components:

$$F_{\text{halu}}^{\text{seg}}(C_k) = \alpha F_{\text{halu}}^{\text{token}}(C_k) + \beta \, \text{Consistency}(C_k) + \gamma \, \text{Alignment}(C_k), \tag{12}$$

where $\alpha, \beta, \gamma \in [0, 1]$ represent the relative importance of the three parts, respectively, satisfying $\alpha + \beta + \gamma = 1$. In our experiments, we set $\alpha = 0.5, \beta = 0.3, \gamma = 0.2$.

Segments above $\tau_{\text{seg}}^{\text{high}}$ are valid and stored or used in multi-step reasoning. Segments between $\tau_{\text{seg}}^{\text{low}}$ and $\tau_{\text{seg}}^{\text{high}}$ are refined, while those below $\tau_{\text{seg}}^{\text{low}}$ are discarded to keep only reliable, coherent content.

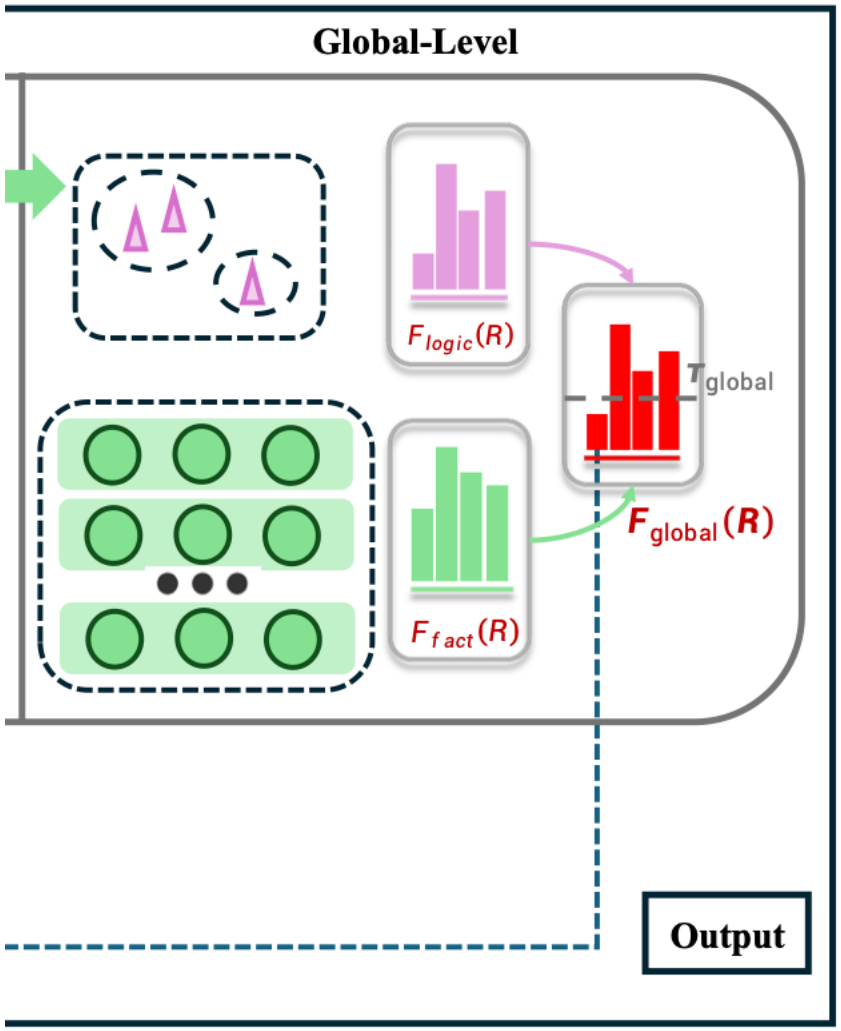
During local refinement, only the target segment C_k is updated. The hidden states of downstream segments $C_{k+1}, C_{k+2}, \ldots$ are retained to preserve global structure, avoiding full regeneration of the remaining sequence and ensuring efficiency, following the principle of selective local refinement as demonstrated in (Tang et al., 2025; Lyu et al., 2025).

For local refinement, the lowest-scoring token a_{low} is identified within the segment, and a local window $W_k^{(l)} = \{a_{i-1}, a_i^{\text{low}}, a_{i+1}\}$ is formed including its immediate neighbors. This window is used to correct the weakest part of the segment while preserving surrounding context. The LLM generates refined tokens conditioned on the surrounding context $a_{<i-1}, a_{>i+1}$ and the segment representation H_k, producing a refined window $W_k^{(l)'}$ that replaces the original window in the segment:

$$W_k^{(l)'} = \text{LLM\_refine}(W_k^{(l)} \mid a_{<i-1}, a_{>i+1}, H_k), \quad C_k' = C_k \setminus W_k^{(l)} \cup W_k^{(l)'}. \tag{13}$$

After refinement, the segment score $F_{\text{halu}}^{\text{seg}}(C_k')$ is recomputed. If it reaches $\tau_{\text{seg}}^{\text{high}}$, the segment is accepted; otherwise, iteration continues up to $N_{\max}$ steps. This process ensures a controlled, stepwise improvement of segment reliability. We set $\tau_{\text{seg}}^{\text{low}} = 0.55, \tau_{\text{seg}}^{\text{high}} = 0.75, N_{\max} = 3$.

Global-level



In Stage Three, Token-Guard globally re-evaluates candidate outputs from Stage Two by assembling reliable segments into reasoning chains $R = \{C_1, C_2, \ldots, C_K\}$. We do not rely on the original segment order, since exploring multiple branches can improve global consistency and factual accuracy, inspired by Saha et al. (2024). Segments are clustered via TF-IDF and KMeans, and candidate chains are generated by selecting the chain closest to each cluster centroid. We set the number of clusters to $K = 5$ for large-scale datasets, while for small datasets such as HaluEval we use $K = 3$ to reduce computation. Each segment C_k has a vector representation H_k, a token-level confidence vector $\mathbf{f}_k$, and a knowledge-verification score E_k comparing generated content with relevant knowledge sources. The factual consistency of a chain is computed as:

$$F_{\text{fact}}(R) = \frac{1}{K}\sum_{k=1}^{K} w_k\, F_{\text{seg}}(C_k), \quad w_k = \frac{\|\mathbf{f}_k\| \cdot E_k}{\sum_{j=1}^{K} \|\mathbf{f}_j\| \cdot E_j}, \tag{14}$$

where $F_{\text{seg}}(C_k)$ means segment-level hallucination score, and w_k combines token confidence and evidence alignment to emphasize longer segments.Logical coherence is measured between adjacent segments using cosine similarity of H_k and H_{k+1}, weighted by contextual factor λ_k:

$$F_{\text{logic}}(R) = \frac{1}{K-1}\sum_{k=1}^{K-1} \lambda_k \frac{H_k \cdot H_{k+1}}{\|H_k\|\,\|H_{k+1}\|}, \quad \lambda_k = \text{sim}_{\text{ctx}}(C_k, C_{k+1}), \tag{15}$$

where sim_{ctx} provides a multiplicative contextual adjustment and is defined as:

$$\text{sim}_{\text{ctx}}(C_k, C_{k+1}) = \frac{1 + \cos(\tilde{e}_k, \tilde{e}_{k+1})}{2}, \qquad \tilde{e}_k = \frac{1}{n_k}\sum_{i=1}^{n_k} e_{k,i}, \tag{16}$$

with $e_{k,i}$ denoting the input token embedding of segment C_k. This factorized formulation combines hidden-state continuity through $\cos(H_k, H_{k+1})$ and semantic alignment between segment texts through sim_{ctx}, which uses only input embeddings reconstructed from token IDs.

The global score $F_{\text{global}}(R)$ combines factual and logical components via a soft minimum:

$$F_{\text{global}}(R) = \frac{F_{\text{fact}}(R)\, F_{\text{logic}}(R)}{F_{\text{fact}}(R) + F_{\text{logic}}(R) - F_{\text{fact}}(R)\, F_{\text{logic}}(R)}, \tag{17}$$

triggering re-generation if $F_{\text{global}}(R) < \tau_{\text{global}}$, which we set 0.7, and returning "cannot answer" if both F_{fact} and F_{logic} are below 0.5. To improve stability, segment thresholds are dynamically adjusted by an adjustment margin $\Delta\tau$:

$$\tau_{\text{seg}}^{\text{adj}} = \begin{cases} \tau_{\text{seg}}^{\text{high}} + \Delta\tau, & \text{if } F_{\text{fact}} \text{ is low and } F_{\text{logic}} \text{ is high,} \\ \tau_{\text{seg}}^{\text{low}} - \Delta\tau, & \text{if } F_{\text{logic}} \text{ is low and } F_{\text{fact}} \text{ is high.} \end{cases} \tag{18}$$

where $\tau_{\text{seg}}^{\text{adj}}$ is the adjusted threshold. Each iteration produces a candidate chain $R^{(i)}$ and recalculates $F_{\text{global}}^{(i)}$; the chain is accepted if $F_{\text{global}}^{(i)} \geq \tau_{\text{global}}$ before $M_{\max}$ iterations, otherwise the system outputs "cannot answer." In our experience, we set $\tau_{\text{global}} = 0.7, \Delta\tau = 0.1, M_{\max} = 2$

4

PART

Experimental Results

Main Results

Method	FinanceBench		RAGTruth		CovidQA		DROP_history		DROP_nfl		PubmedQA		Halueval		Avg	
	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1
Meta-Llama-3.1-8B-Instruct																
BaseModel	0.16	16.00	0.00	14.71	0.02	32.33	0.24	44.21	0.30	39.10	0.00	9.55	0.32	42.16	0.15	28.29
Guided Decoding	0.14	16.44	0.00	22.21	0.04	40.43	0.34	55.95	0.18	36.71	0.00	13.95	0.42	57.41	0.16	34.73
Predictive Decoding	0.09	8.79	0.00	14.48	0.04	39.36	0.14	29.47	0.20	29.22	0.00	11.69	0.22	38.00	0.10	24.43
Chain-of-Thoughts	0.11	11.01	0.00	20.47	0.08	36.84	0.30	49.26	0.34	49.21	0.00	20.33	0.40	55.32	0.18	34.63
Tree-of-Thought	0.10	14.44	0.00	21.33	0.10	43.70	0.22	47.73	0.24	37.69	0.00	12.38	0.38	56.02	0.15	33.33
Token-Guard(Ours))	**0.30**	**30.80**	**0.02**	**43.94**	**0.08**	**47.64**	**0.48**	**68.52**	**0.44**	**58.10**	0.00	**29.67**	**0.68**	**78.54**	**0.29**	**51.03**
Qwen3-8b																
BaseModel	0.20	26.67	0.00	36.12	0.05	42.57	0.44	65.10	0.39	57.02	0.00	22.45	0.43	59.83	0.22	44.25
Guided Decoding	0.21	23.56	0.00	39.35	0.07	43.19	0.52	66.11	0.43	53.68	0.00	24.76	0.62	69.88	0.26	45.79
Predictive Decoding	0.04	11.25	0.00	34.51	0.00	31.92	0.06	23.85	0.00	23.41	0.00	19.44	0.00	21.29	0.01	23.67
Chain-of-Thoughts	0.29	35.12	0.00	33.78	0.00	41.63	0.48	69.33	0.49	59.89	0.00	22.77	0.34	53.21	0.23	45.10
Tree-of-Thought	0.28	34.91	0.00	36.07	0.00	37.34	0.49	69.12	0.34	54.67	0.00	26.41	0.39	57.33	0.21	45.12
Token-Guard(Ours)	**0.45**	**45.37**	**0.06**	**45.89**	**0.09**	**44.01**	**0.66**	**71.83**	**0.66**	**67.69**	0.00	**28.91**	**0.51**	**74.15**	**0.35**	**53.98**

- Token-Guard achieves superior performance across diverse architectures, outperforming baselines with an average EM/F1 of **0.29/51.03 on Meta-Llama-3.1-8B** and **0.35/53.98 on Qwen3-8B**. These results underscore its robust cross-model compatibility and its ability to maintain high generation quality across various LLM scales.
- The framework excels in multi-step reasoning tasks like **DROP_nfl**, where token and segment-level scoring effectively suppress hallucinations and enhance logic. While gains are more modest in knowledge-intensive domains like **PubMedQA**—where missing external knowledge remains a factor—Token-Guard still significantly reduces the frequency of generated inaccuracies.
- Near-zero EM scores on **RAGTruth** and **PubMedQA** reflect the flexible nature of long-form references rather than a failure of factual accuracy. Since open-ended outputs rarely match reference strings perfectly, Token-Guard shifts the evaluative focus from simple pattern matching toward deep, robust factual verification and cross-task reliability.

Ablation study

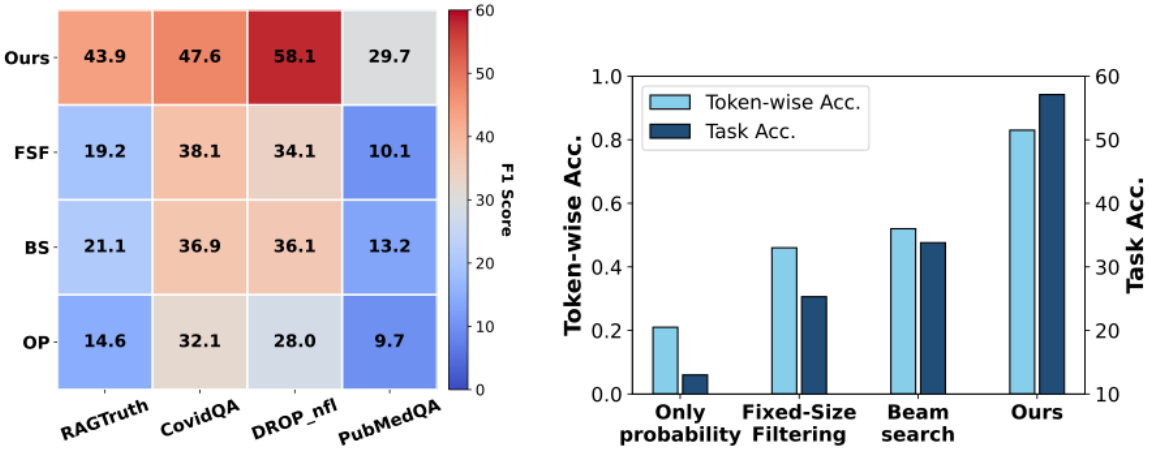
Method Variant	DROP_history			RAGTruth			Avg.		
	EM	F1	BLEU	EM	F1	BLEU	EM	F1	BLEU
Full Token-Guard	0.48	68.52	65.21	0.02	43.94	38.27	0.25	56.23	51.74
w/o Prompt	0.32	55.23	51.48	0.01	32.50	27.92	0.17	43.87	39.70
w/o Token-Level Scoring	0.28	47.51	44.88	0.00	27.10	25.05	0.14	37.31	34.97
w/o Segment-Level Scoring	0.43	60.10	59.55	0.01	39.20	33.08	0.22	49.65	46.32
w/o Global Iteration	0.42	63.05	52.37	0.01	41.05	20.14	0.22	52.05	36.26

Table 2: Ablation study of Token-Guard on representative datasets.

We evaluate the contributions of Token-Guard's core components: **prompt initialization (P), token-level scoring (T), segment-level scoring (S), and global iteration (G).** Table 2 shows that removing any component degrades performance. Token-level scoring most affects EM and F1, due to its crucial role in guiding early-stage token generation, while prompt initialization and segment-level scoring provide stability. Notably, global iteration boosts BLEU, enhancing linguistic fluency.

Analysis of Token-Guard's Effect on Factual Precision

As shown in Figure 4, to evaluate Token-Guard's effect on factual precision, we analyze it from (a) token-level F1 across four datasets and (b) accuracy of estimated step values.



(a) Token-level F1 (b) Step Value Accuracy

Figure 4: Comparison of Token-Guard and token-level decoding methods on factual precision.

Token-level Precision Advantage. Token-Guard consistently outperforms other token-level methods, including Fixed-Size Filtering (FSF), Beam Search (BS), and Only Probability (OP)(Wang et al., 2025), demonstrating its effectiveness in retaining factual tokens.

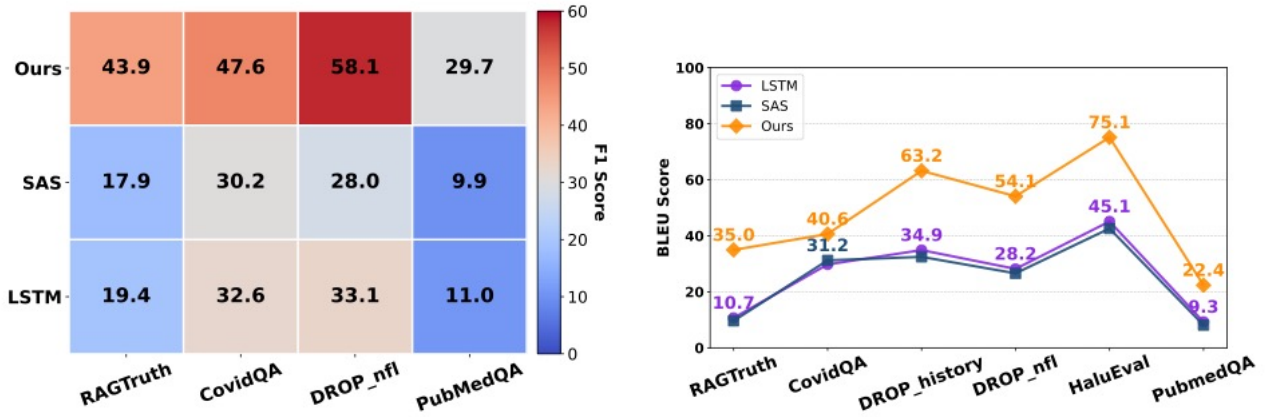
Reliability Across Generation Steps. The token-level accuracy is positively correlated with the correctness of the final answer. By ensuring high accuracy at each generation step, Token-Guard improves overall task performance, highlighting the importance of its token-level self-correction mechanism. Please refer to Appendix G for details.

Analysis of Token-Guard's Impact on Relevance

As shown in Figure 5, to evaluate Token-Guard's impact on relevance, we analyze it from (a) segment-level F1 across four datasets and (b) BLEU scores across six datasets.

Improved Segment-level Precision. As shown in Figure 5a, Token-Guard substantially outperforms SAS (Self-Adaptive Sampling) and LSTM (Long Short-Term Memory based decoder) (Ali et al., 2024) in segment-level F1 across all datasets, which demonstrates its superior ability to preserve relevant segments and suppress irrelevant fragments during decoding.

Robust Relevance Across Benchmarks. As shown in Figure 5b, Token-Guard consistently attains the highest BLEU scores across six datasets, e.g., 63.19 on DROP_history and 75.13 on HaluEval, confirming its robustness in maintaining output relevance and fluency under diverse tasks.



(a) Segment-level F1 (b) BLEU Scores

Figure 5: Comparison of Token-Guard and segment baselines on relevance.

Analysis of Token-Guard'sPortablitity

Table 6 reports the performance of Token-Guard and several baseline decoding strategies, evaluated on both **Llama-3.2-3B-Instruct** and **Llama-3.2-13B-Instruct** backbone models. Predictive Decoding (PD) is omitted on the 13B backbone because it triggered out-of-memory (OOM) errors during decoding.

Token-Guard achieves the highest F1 and BLEU on **DROP_history** and **HaluEval**, showing consistent gains on both model scales. These improvements indicate that Token-Guard scales favorably with larger backbone capacity.

Table 6: Performance on 3B and 13B models.

Method	RAGTruth		DROP_history		HaluEval	
	F1	BLEU	F1	BLEU	F1	BLEU
3B Model						
BaseModel	12.11	6.29	42.21	39.31	42.95	39.95
Chain-of-Thoughts	**32.70**	**25.48**	31.60	26.04	50.88	48.44
Tree-of-Thought	*27.05*	*21.16*	*39.84*	*36.14*	59.22	55.11
Predictive Decoding	21.72	14.59	34.19	31.07	57.55	53.97
Guided Decoding	30.50	18.70	32.60	28.83	*63.47*	*59.11*
Token-Guard(Ours)	15.66	13.45	**43.09**	**40.57**	**68.21**	**63.83**
13B Model						
BaseModel	22.47	15.26	46.32	42.74	35.30	31.40
Chain-of-Thoughts	32.25	27.67	33.90	30.06	49.43	44.01
Tree-of-Thought	32.93	23.70	43.97	38.25	62.20	58.07
Predictive Decoding	–	–	–	–	–	–
Guided Decoding	29.34	22.01	40.41	36.71	63.42	58.72
Token-Guard(Ours)	**33.14**	**30.41**	**51.17**	**47.64**	**72.66**	**68.81**

On **RAGTruth**, Token-Guard performs lower on the 3B backbone due to the dataset's long-context structure, which is challenging for smaller models. The 13B model alleviates this issue to some extent, offering more stable decoding. Overall, Token-Guard remains effective across backbone sizes, with limitations mainly arising from extremely large-context settings. The results confirm that Token-Guard is portable across model scales without requiring additional tuning.

Analysis of Token-Guard's Time and Memory Consumption

Here we examine Token-Guard's computational efficiency, analyzing time overhead and memory usage across decoding stages. We focus on two aspects: (a) runtime, including per sample latency and throughput, and (b) memory across token level, segment level, and global level stages. These measurements reveal Token-Guard's resource implications.

Time and Throughput Analysis. As shown in Table 4, Token-Guard exhibits distinct time, token, and throughput patterns across datasets. Its dynamic iteration enables efficient computation on **PubMedQA** and **HaluEval**, while allocating more runtime and output tokens for challenging datasets such as **CovidQA** and **RAGTruth**. Notably, despite higher token budgets in some settings, Token-Guard maintains competitive or superior throughput, reflecting its ability to balance iterative refinement with efficient decoding.

Method	RAGTruth	CovidQA	PubmedQA	HaluEval	Avg.
	Time · Token · Tokens/sec	Time · Tokens · Tokens/sec	Time · Tokens · Tokens/sec	Time · Tokens · Tokens/sec	Time · Tokens · Tokens/sec
Chain-of-Thoughts	11 · 899 · 80.71	9 · 670 · 73.14	11 · 998 · 88.45	7 · 543 · 77.33	9.65 · 777.08 · 79.41
Tree-of-Thought	57 · 3687 · 64.49	67 · 3423 · 51.09	58 · 4253 · 73.38	35 · 2786 · 79.49	54.29 · 3537.19 · 67.11
Predictive Decoding	107 · 12058 · 112.82	138 · 6473 · 47.07	79 · 8416 · 106.06	58 · 5954 · 102.20	95.50 · 8225.61 · 92.54
Guided Decoding	110 · 17234 · 155.96	189 · 13525 · 71.64	103 · 18733 · 182.05	77 · 11689 · 151.15	119.38 · 15295.41 · 140.20
Token-Guard	69 · 18024 · 262.81	301 · 17474 · 58.10	32 · 7699 · 240.58	54 · 5254 · 97.54	113.29 · 12112.95 · 164.76

Table 4: Time, output token consumption, and normalized throughput (tokens/sec).

Memory Consumption Analysis. On the **HaluEval** dataset, Token-Guard shows a multi-stage memory profile: token-level states are buffered efficiently (1.2–9.4%), segment-level briefly peaks (21.6–19.6%), and global-level stabilizes around 10.3–19.0%. Overall, peak (21.6%) and average (16.3%) memory remain competitive with baselines. Table 5 summarizes peak memory, average memory and runtime for Token-Guard and other methods.

Method	Peak Memory (%)	Avg Memory (%)	Runtime (s)
Guided Decoding	21.4	15.0	77.31
Chain-of-Thoughts	21.3	14.5	7.02
Tree-of-Thought	21.4	17.8	35.04
Predictive Decoding	4.1	3.7	58.26
Token-Guard	21.6	16.3	53.89

Table 5: Memory and runtime comparison for Token-Guard and baseline methods.

Thanks for Listening

Code & Data https://github.com/RongHuiQiang/Token-Guard

Contact rong@bupt.edu.cn