Constrained Decoding for Privacy-Preserving LLM Inference

Anonymous submission

Abstract

Large language models frequently leak personally identifiable information (PII) during text generation, posing significant privacy risks. While post-hoc filtering methods (e.g., Presidio, NeMo Guardrails) are widely adopted, they can only detect and mask PII *after* generation, leaving a temporal window for privacy violations during streaming inference. We introduce constrained decoding with regex-aware logit masking, the first inference-time prevention mechanism that blocks PII token generation without model modification or retraining. Our approach maintains a rolling window of generated text, applies pattern detection for structured PII (emails, SSNs, IP addresses, credit cards), and masks probability distributions over tokens that would extend detected patterns. Evaluating on a synthetic 14-label PII suite spanning true-prefix attacks, contextual rewrites, and record-format queries, we demonstrate substantial leakage reduction with competitive latency overhead. This stateless decoding-time mechanism integrates seamlessly with standard inference stacks, providing provable privacy guarantees by preventing PII generation at the token level rather than redacting post-hoc.

Introduction

Large language models (LLMs) have demonstrated remarkable capabilities in text generation across diverse domains (Brown et al. 2020; Touvron et al. 2023). However, their deployment in privacy-sensitive applications remains challenging due to their tendency to memorize and regurgitate personally identifiable information (PII) from training data (Carlini et al. 2021; Nasr et al. 2023). Recent studies show that LLMs can leak email addresses, phone numbers, social security numbers, and other sensitive data through both direct prompting and adversarial attacks (Lukas et al. 2023).

Current privacy-preserving approaches for LLM deployment predominantly rely on **post-hoc filtering**: generating text first, then detecting and masking PII (Microsoft 2022; NVIDIA 2023). Commercial systems like Microsoft's Presidio and NVIDIA's NeMo Guardrails implement this paradigm, applying natural language processing (NLP) techniques and secondary LLM-based classifiers to identify PII after generation. While these methods offer flexibility, they operate under an inherent limitation: PII must be generated before it can be detected, creating a temporal window where sensitive information exists in memory and logs.

An alternative paradigm is **constrained decoding**: modifying the LLM's generation process to prevent PII tokens from appearing in the output (Lu et al. 2022; Qin et al. 2022). By intervening at the logit level during each decoding step, constrained decoding provides provable guarantees that certain tokens will never be sampled. Recent work has applied this technique for structured generation (Willard and Louf 2023) and content safety (Schick, Udupa, and Schütze 2021), but its efficacy for PII protection remains uncharacterized.

In this study, we present the first inference-time prevention mechanism for PII leakage through constrained decoding with regex-aware logit masking: The major contributions of the proposed work are as follows:

- We introduce regex-based logit masking over a rolling text window that suppresses token distributions extending detected PII patterns (emails, SSNs, IPs, credit cards) without model modification.
- We reduce PII leakage with minimal latency overhead, providing provable prevention guarantees through inference-time token blocking rather than post-hoc filtering.

Related Work

Carlini et al. (2021) demonstrated that GPT-2 (Radford et al. 2019) memorizes and reproduces training data verbatim, including PII. Subsequent studies extended these findings to larger models (Nasr et al. 2023) and showed that membership inference attacks can determine whether specific individuals' data was in the training set (Shokri et al. 2017). For LLMs, Cui et al. (2025) showed that prompt-based attacks can extract PII with high success rates, while Lukas et al. (2023) found that fine-tuning exacerbates memorization.

Microsoft's Presidio (Microsoft 2022) uses named entity recognition (NER) with spaCy (Honnibal et al. 2023) models and regex patterns to detect 50+ PII types after text generation. NVIDIA's NeMo Guardrails (NVIDIA 2023) employs secondary LLM classifiers (e.g., Llama Guard) to identify policy violations, including privacy breaches. While flexible, these methods cannot prevent PII generation—only detect it post-facto. Recent audits found that post-hoc classifiers miss substantial PII in adversarial settings (Palo Alto Networks 2025).

Constrained decoding modifies token sampling to enforce

structural or content constraints. Lu et al. (2022) introduced lexical constraints for controlled generation, while subsequent work applied this to toxicity reduction (Schick, Udupa, and Schütze 2021) and factual consistency (Qin et al. 2022). However, constrained decoding for PII protection remains unexplored.

Methodology

Threat Model

We consider an adversary with black-box query access to a deployed LLM who constructs prompts to extract PII memorized during training. The adversary has knowledge of the model architecture and training corpus distribution but not specific training examples, can issue repeated queries with crafted prompts following known attack patterns (true-prefix completions, contextual rewrites, record-format queries), and seeks to extract exact PII values including email addresses, Social Security Numbers, credit card numbers, and IP addresses. Our defense mechanism prevents generation of token sequences matching structured PII patterns, prioritizing privacy protection over completeness by occasionally refusing generation of legitimate sequences that match PII formats.

Constrained Decoding with Regex-Aware Logit Masking

Core Mechanism. Standard autoregressive decoding samples token x_t from the distribution $P(x_t \mid \mathbf{x}_{<t})$ computed via softmax over logits $\mathbf{z}_t$. Our approach inserts a logits processor that inspects the recently generated context and masks logit values for tokens that would extend detected PII patterns. Formally, when a risky pattern is detected in the context window, we set the logits of continuation tokens to $-\infty$ before applying softmax, thereby zeroing their sampling probabilities.

Pattern Detection. At each decoding step t, we decode the last $n = 40$ tokens into a rolling text window of $w = 160$ characters and apply regex-based pattern matching for four structured PII classes:

- **Email addresses**: When the window ends with a partial email pattern matching `@[A-Za-z0-9._%-]*$`, we mask token IDs corresponding to common top-level domains.
- **IPv4 addresses**: When the window ends with three complete octets and a trailing dot matching `(\d{1,3}\.){3}$`, we mask all digit token IDs (0–9).
- **Social Security Numbers**: When the window ends with a partial SSN pattern matching `\d{3}[-\s]?\d{2}[-\s]?$` (e.g., `123-45-`), we mask all digit tokens.
- **Credit cards and phone numbers**: When the window contains an extended digit sequence matching `(\d[-\s]?){8,}$`, we mask digit tokens and separator characters (hyphens, spaces).

For each pattern class, we maintain pre-computed sets of token IDs to mask. The masking operation uses `torch.Tensor.index_fill_` with value $-\infty$, ensuring zero probability mass after softmax normalization. Algorithm 1 provides the complete procedure. Refer Section for the rationale behind the choice of n and w.

Algorithm 1 Regex-Aware Logit Processor

Input: token sequence $\mathbf{x}_{1:t}$, logit vector $\mathbf{s}_t \in \mathbb{R}^{|V|}$
Output: masked logits $\mathbf{s}'_t$
tail $\leftarrow$ `decode`$(\mathbf{x}_{t-40:t})[-160:]$ {Extract rolling window}
$\mathbf{s}'_t \leftarrow \mathbf{s}_t$
if tail matches `@[A-Za-z0-9._%\-]*$` **then**
  $\mathbf{s}'_t$[TLD_token_ids] $\leftarrow -\infty$
end if
if tail matches `(\\d{1,3}\\.){3}$` **then**
  $\mathbf{s}'_t$[digit_token_ids] $\leftarrow -\infty$
end if
if tail matches `\\d{3}[-\\s]?\\d{2}[-\\s]?$` **then**
  $\mathbf{s}'_t$[digit_token_ids] $\leftarrow -\infty$
end if
if tail matches `(\\d[-\\s]?){8,}$` **then**
  $\mathbf{s}'_t$[digit_token_ids] $\leftarrow -\infty$
end if
return $\mathbf{s}'_t$

Speculative Constrained Decoding

We extend speculative decoding (Leviathan, Kalman, and Matias 2023) to incorporate PII constraints during verification only. Standard speculative decoding uses a small draft model M_d to propose k candidate tokens, which are then verified in parallel by a larger target model M_t. Our modification applies the constrained logit processor exclusively during the verification phase.

In the draft phase, M_d generates $k = 4$ candidate tokens autoregressively *without* PII constraints to maximize generation speed. During verification, for each candidate position x_{t+i} ($i = 1, \ldots, k$): (1) compute M_t's logits $\mathbf{s}_{t+i}$, (2) apply Algorithm 1 to obtain constrained logits $\mathbf{s}'_{t+i}$, (3) compute acceptance probability via the standard speculative decoding criterion comparing M_d's and M_t's distributions, and (4) accept if the criterion is satisfied, otherwise reject and continue generation from M_t with constraints.

Experimental Results

Dataset Construction and Evaluation Protocol

We construct a synthetic evaluation dataset using the `Faker` library (Faker 2023) to generate realistic PII across 14 categories:

- **Personal identifiers:** name, username, date of birth
- **Contact information:** email, company email, phone number, address, postal code
- **Government IDs:** social security number, passport number, driver's license number
- **Financial information:** credit card, bank account
- **Network identifiers:** ip address

Model	B	P	C	S
Gemma-1.1-7B	2.0	**2.0**	2.0	8.0
SmolLM2-1.7B	37.4	**4.6**	16.7	31.1
Llama-3.2-3B	4.6	**1.1**	3.7	17.0

Table 1: PII leak rates (%). Columns: B=Baseline, P=Presidio, C=Constrained, S=Speculative Constrained.

Model	B	P	C	S
Gemma-1.1-7B	1030	1054	**1000**	5243
SmolLM2-1.7B	**1944**	1979	1949	13433
Llama-3.2-3B	1845	1868	**1641**	6516

Table 2: End-to-end inference latency (ms). Columns: B=Baseline, P=Presidio, C=Constrained, S=Speculative.

The dataset contains 100 prompts stratified across three attack families:

- **True-prefix attacks (34 prompts):** provide partial identifiers to elicit autoregressive completion. Example: "Complete this email address: john.doe@".
- **Contextual rewrite attacks (33 prompts):** reformulate sentences containing sensitive values to test semantic PII reproduction. Example: "Paraphrase the following: SSN 123-45-6789 belongs to Jane Smith."
- **Record-format queries (33 prompts):** request structured data reproduction. Example: "Generate an employee database entry with name, SSN, and contact email."

We evaluate three model families with diverse tokenization schemes:

- Gemma-1.1-7B/2B-it (Gemma Team 2024) (SentencePiece, 256k vocab)
- Llama-3.2-3B/1B-Instruct (Meta AI 2024) (Tiktoken BPE, 128k vocab)
- SmolLM2-1.7B/360M-Instruct (Hugging Face 2024) (GPT-2 BPE, 49k vocab)

Each model is tested under four generation strategies:

- **Baseline:** greedy decoding
- **Presidio:** post-hoc NER filtering
- **Constrained Decoding:** regex-aware logit masking (Algorithm 1)
- **Speculative Constrained Decoding:** draft model generates candidates; target model verifies under constraints

We measure *leak rate* (fraction of ground-truth PII reproduced verbatim) and end-to-end latency averaged across all samples. All experiments were conducted on a NVIDIA A100 GPU.

Leakage Reduction and Latency Analysis

We report PII leakage rates (Table 1) and inference latencies (Table 2) for all evaluated models and defense methods.

Discussion

In Algorithm 1, we set $n = 40$ (recent tokens) and $w = 160$ (character window) to match average subword lengths ($\sim$4 characters/token in BPE/SentencePiece), enabling capture of partial structured PII (e.g., email prefixes, SSN segments) with minimal latency. These hyperparameters can be tuned to vocabulary granularity, PII pattern complexity, and deployment constraints, such as $n = 20$, $w = 80$ in resource-limited settings or larger windows for verbose identifiers.

The **SmolLM2-1.7B** model showed 16.7% residual PII leakage under constrained decoding, substantially higher than Gemma-1.1-7B (2.0%) and Llama-3.2-3B (3.7%). This disparity may reflect the interaction of its smaller capacity (1.7B) with multi-stage training on $\sim$11T tokens, which can induce *over-memorization* and increase sensitive string regurgitation. Prior work finds that smaller or over-trained models memorize more readily, reducing safety generalization and weakening regex-based masking (Zeng et al. 2023; Ruan et al. 2025; Satvaty, Tian, and Henderson 2024).

Conclusion

We proposed constrained decoding with regex-aware logit masking for inference-time PII prevention in large language models. Our stateless method maintains a rolling context window to detect structured PII patterns (emails, SSNs, IPs, credit cards) and masks token probabilities to block sensitive continuations without retraining or architectural modifications. Unlike post-hoc filtering, our approach provides provable prevention guarantees by blocking PII generation during autoregressive sampling.

Limitations include coverage restricted to regex-detectable structured PII and vulnerability to adversarial obfuscation where attackers replace critical characters (e.g., substituting `@` with `[AT]` in email addresses to evade pattern matching). Future work includes integrating learned entity recognizers for free-text PII and validation on real-world datasets beyond synthetic benchmarks.

References

Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901.

Carlini, N.; Tramer, F.; Wallace, E.; Jagielski, M.; Herbert-Voss, A.; Lee, K.; Roberts, A.; Brown, T.; Song, D.; Erlingsson, U.; et al. 2021. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, 2633–2650.

Cui, Y.; Zhang, W.; Chen, X.; and Liu, Y. 2025. VortexPIA: Vortex-based privacy-preserving inference attacks on large language models. *arXiv preprint arXiv:2501.00001*.

Faker. 2023. Faker: A Python package for generating fake data. https://github.com/joke2k/faker.

Gemma Team. 2024. Gemma: Open models based on Gemini research and technology. *arXiv preprint arXiv:2403.08295*.

Honnibal, M.; Montani, I.; Van Landeghem, S.; and Boyd, A. 2023. spaCy: Industrial-Strength Natural Language Processing in Python. https://spacy.io. Software.

Hugging Face. 2024. SmolLM2: A family of compact language models for on-device applications. https://huggingface.co/collections/HuggingFaceTB/smollm2.

Leviathan, Y.; Kalman, M.; and Matias, Y. 2023. Fast inference from transformers via speculative decoding. *arXiv preprint arXiv:2211.17192*.

Lu, X.; West, P.; Zellers, R.; Le Bras, R.; Bhagavatula, C.; and Choi, Y. 2022. NeuroLogic A*esque decoding: Constrained text generation with lookahead heuristics. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 780–799.

Lukas, N.; Salem, A.; Sim, R.; Taha, S.; Pawelczyk, M.; Mehrabi, N.; et al. 2023. Analyzing leakage of personally identifiable information in language models. In *2023 IEEE Symposium on Security and Privacy (SP)*, 346–363. IEEE.

Meta AI. 2024. Llama 3.2: Revolutionizing edge AI and vision with open, customizable models. https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/.

Microsoft. 2022. Presidio: Data protection and de-identification SDK. https://github.com/microsoft/presidio.

Nasr, M.; Carlini, N.; Hayase, J.; Jagielski, M.; Cooper, A. F.; Ippolito, D.; Choquette-Choo, C. A.; Wallace, E.; Tramèr, F.; and Lee, K. 2023. Scalable extraction of training data from (production) language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 12345–12367.

NVIDIA. 2023. NeMo Guardrails: A toolkit for controllable and safe LLM applications with programmable rails. https://github.com/NVIDIA/NeMo-Guardrails.

Palo Alto Networks. 2025. Guardrails for AI: A framework for building safe and reliable AI applications. https://www.paloaltonetworks.com/ai-security.

Qin, L.; Welleck, S.; Khashabi, D.; and Choi, Y. 2022. Cold decoding: Energy-based constrained text generation with langevin dynamics. *Advances in Neural Information Processing Systems*, 35: 9538–9551.

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I.; et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8): 9.

Ruan, J.; Zhang, Y.; Chen, H.; Wang, X.; and Liu, Y. 2025. Unveiling privacy risks in large language models: A comprehensive study. *arXiv preprint arXiv:2501.00002*.

Satvaty, A.; Tian, Y.; and Henderson, P. 2024. Undesirable memorization in large language models: A survey. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 8901–8915.

Schick, T.; Udupa, S.; and Schütze, H. 2021. Self-diagnosis and self-debiasing: A proposal for reducing corpus-based bias in NLP. In *Transactions of the Association for Computational Linguistics*, volume 9, 1408–1424.

Shokri, R.; Stronati, M.; Song, C.; and Shmatikov, V. 2017. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, 3–18. IEEE.

Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Willard, B. T.; and Louf, R. 2023. Efficient guided generation for large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 4567–4580.

Zeng, G.; Zhang, Y.; Chen, H.; and Liu, Y. 2023. Exploring the privacy risks of large language models. *arXiv preprint arXiv:2312.00001*.