LatticeGen: Hiding Generated Text in a Lattice for Privacy-Aware Large Language Model Generation on Cloud

Mengke Zhang[2*], **Tianxing He**[1*], **Tianle Wang**[2],
Lu Mi[1,3], **Fatemehsadat Mireshghallah**[1], **Binyi Chen**[4], **Hao Wang**[5], **Yulia Tsvetkov**[1]
[1]University of Washington [2]University of California, San Diego
[3]Allen Institute for Brain Science [4]Espresso Systems [5]Rutgers University
mezhang@ucsd.edu, goosehe@cs.washington.edu

Abstract

In the current user-server interaction paradigm of prompted generation with large language models (LLMs) on cloud, the server fully controls the generation process, which leaves zero options for users who want to keep the generated text private to themselves. For privacy-aware text generation on cloud, we propose LatticeGen, a cooperative protocol in which the server still handles most of the computation while the client controls the sampling operation. The key idea is that the true generated sequence is mixed with noise tokens by the client and hidden in a noised lattice. Only the client knows which tokens are the true ones. Considering potential attacks from a hypothetically malicious server and how the client can defend against it, we propose the repeated beam-search attack and the mixing noise scheme. In our experiments we apply LatticeGen to protect both prompt and generation. It is shown that while the noised lattice degrades generation quality, LatticeGen successfully protects the true generation to a remarkable degree under strong attacks (more than 50% of the semantic remains hidden as measured by BERTScore).

1 Introduction

Many of the high-performing large language models (LLMs) need to be deployed on cloud servers, whether they are open-sourced but have an intensive need for computation (Zhao et al., 2023; Kaplan et al., 2020; Leviathan et al., 2023), or behind a paywall like ChatGPT (OpenAI, 2023). This raises new privacy challenges (Li et al., 2021; Yu et al., 2021; Kerrigan et al., 2020), since users have to send or receive their data to/from cloud providers.

In this work we focus on a popular interaction paradigm between end users and a server hosting an LLM on cloud named *prompted generation*: The user sends server a prompt, which is usually an instruction (Chung et al., 2022) or the beginning of a document (Deng et al., 2022), and the server, who fully controls the generation process, sends user back the generated text from the LLM. Both the prompt and the generation are raw texts which are completely transparent and accessible to the server, leaving zero options for users who want to keep the generated text private to themselves.

As LLMs become widely deployed in professional and social applications, we argue that in prompted generation, there are many scenarios in which not only the prompts, **but also the generated texts need some level of obfuscation, because they can directly affect the user's real-life private decisions.** For example, a customer is likely to go to the restaurant suggested by the LLM, and a writer could take inspiration from outputs provided by the LLM. With the goal of preventing the server from gaining complete knowledge of the generated text and prompt, we propose LatticeGen (Figure 2), a client–server interaction protocol in which the user and client conduct privacy-aware generation token-by-token in a cooperative way. The protocol can be executed by a local client so that the interface is kept simple for the user. We summarize our key contributions below:

- The high-level idea of LatticeGen (§3) is that in each time-step, the client sends the server not one, but N tokens (thus the name *lattice*), in which one is true and others act as noise. The server does LLM inference and sends client back the next-token distributions for all N tokens, which are used by the client to sample the true and noise tokens for the next time-step.
- Considering potential attacks from a hypothetically malicious server and how the client can defend against it (§4), we propose the repeated beam-search attack and the mixing noise scheme as defense.
- We apply LatticeGen to the task of creative

[*]Equal Contribution. Both are corresponding authors.

Findings of the Association for Computational Linguistics: NAACL 2024, pages 2674–2690

writing (Fan et al., 2018). Our experiments (§5) show that while the noised lattice degrades generation quality, LatticeGen successfully prevents a malicious server from recovering the true generation to a remarkable degree (more than 50% of the semantic remains unknown as measured by BERTScore). [1]

2 Motivation and Preliminaries

2.1 Generated Text (also) Needs Obfuscation

In the current user–server interaction paradigm, the user sends the server a prompt which is usually the beginning of a dialogue, story or instruction, then the server generates a complete response autoregressively (§2.3), and sends it back to the user. Both the prompt and generation are directly available to the server in raw text format.

This paper contends that generated texts, as well as user prompts, require a privacy protection mechanism. A key reason is that in various scenarios, the generation from the LLM can affect the user's private decisions: e.g., a customer is likely to go to the restaurant suggested by the LLM; a writer could take inspiration from outputs provided by the LLM; an engineer or manager could adopt the approach proposed by the LLM. Industry regulations do not provide ample protection. Please see §E for recent privacy-related incidents with ChatGPT or Bard. The goal of our LatticeGen protocol is to provide a controlled level of obfuscation for the generated text, making it difficult for a hypothetically malicious server to infer the user's actions after interacting with the LLM.

2.2 LatticeGen as a Third-Party Client

Before expanding on the proposed protocol (§3), we first clarify that LatticeGen does not complicate the user interface. Indeed, it is likely that most users still want to keep a simple and intuitive interface for prompted generation. In light of this, LatticeGen can be implemented as a third-party client between the user and the server. As Figure 1 depicts, the client takes the prompt from the user, conducts the privacy-aware generation protocol with the server, and finally returns the generation to the user. In this way, the user does not need to deal with the complicacy in the protocols.

The next question is why would a common user trust the client? One solution is that the client can be open-sourced (e.g., as python scripts) and

[1] Our code and data will be released in here on github.

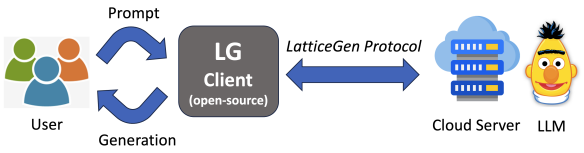

Figure 1: LatticeGen can be implemented as a third-party client handling the protocol for the user.

therefore vetted by researchers and users worldwide. It can also facilitate comprehensive evaluations conducted by different research groups. The user only need to download the script and set the hyper-parameters (e.g., random seed).

2.3 Preliminaries

We will start by reviewing the traditional autoregressive LM generation, and then move on to introduce necessary components of LatticeGen.

Traditional Autoregressive LM Generation We assume the server-side LLM is an autoregressive LM, i.e., it generates tokens one at a time and from left to right (Mikolov, 2012; Cho et al., 2014; Huszár, 2015; Welleck et al., 2020; Dai et al., 2019; Keskar et al., 2019). We denote the LLM as P_M with parameter set θ, the vocabulary as V, the generated token at time-step t as w_t, and the given prompt as p. For convenience we regard the prompt as part of generation, therefore, $w_t := p_t$ for $1 \leq t \leq \text{len}(p)$. In traditional autoregressive generation, on each time-step $t > \text{len}(p)$, the next token w_t is sampled from $P_M(\cdot|w_{0..t-1})$ by calling a sampling algorithm such as top-k (Fan et al., 2017) or nucleus sampling (Holtzman et al., 2020). w_0 is the <bos> token.

The Lattice Structure A simple but key concept in our proposed framework is the *lattice*. In a width-N lattice (or an N-lattice for short), each time-step contains N token options and we denote them as $\{w_t^1, ..., w_t^N\}$. Therefore, a N-lattice of length T (denoted as W_T^N) represents N^T possible sequence combinations. An example with $N = 2$ is shown in the left part of Figure 2.

In our proposed LatticeGen protocols (§3.1), for each time-step t, only the client knows which token is the "true" one, denoted by w_t^{true}. And the other $N-1$ tokens $\{w_t^{\text{noise(1)}}, ..., w_t^{\text{noise(N-1)}}\}$ are referred to as "noise" tokens. Therefore we will also refer to it as the *noised lattice*. To prevent the server from knowing which one is the true token, the client will randomly shuffle the list before attaching it to the lattice and sending to the server.

LM Finetuning and Inference with the LLG (Linearized Lattice plus G-gram) Format As a prerequisite for LatticeGen, we need the server-

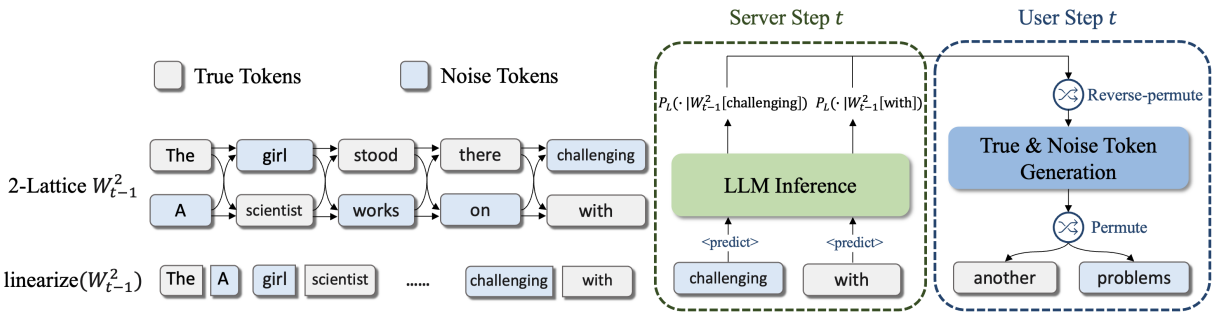

Figure 2: Client-Server interaction under LatticeGen for time-step t. The server controls the LLM P_L, conducts the inference computation and sends client the next-token prediction distribution for each received token. The client conducts the sampling of the true and noise token(s), and sends server a randomly permutated list of tokens for the next time-step. **The server does not know which tokens are the true ones.** The task is creative writing, and the prompt part is omitted in this figure for brevity. An illustration of the server step for $N = 3$ and $G = 2$ is provided in Figure 6, Appendix B.

side LLM (Vaswani et al., 2017) to be able to do inference based on a given lattice and we achieve that by finetuning the base LLM P_M to make next-token prediction with the LLG (Linearized Lattice plus G-gram) format. Below we first introduce this format, and describe the finetuning objective.

First, as the name suggests, we conduct a simple *linearization* operation before feeding the lattice to the LM, in which the token options on each time-step are linearized and concatenated into a sequence of length $T \times N$ (see Figure 2 for an example):

$$\text{linearize}(W_T^N) = [\texttt{<bos>}] + \text{concat}_{i=1}^{T}([w_i^1, ..., w_i^N]). \quad (1)$$

An illustration of a linearized lattice is given in Figure 2.

Next, we append a `<predict>` token and G tokens specifying the token options for the last G tokens (for time-step from $T - G$ to $T - 1$), and the LLM is trained to predict the next token with this specified G-gram "tail". We use notation S to denote a G-gram, where $S_i \in \{w_{T-G+i}^1, ..., w_{T-G+i}^N\}$ for $1 \le i \le G$. In Figure 2, we use uni-gram ($G = 1$) and the last token could be "challenging" or "with". The generation quality will be better with larger G (since the token history is less noised), at the price of more computation: The server will need to enumerate N^G potential combinations.

In §A, we describe a simple process to finetune a LLM to predict the next token for the LLG format. Here we provide a high-level description. For each data sample w^{data}, we construct and linearize a noised lattice by using $N - 1$ other random data samples as noise. The LLM is then finetuned to predict the next true token for several randomly picked tokens in the data sample with the LLG format. We denote the LLG-finetuned LLM as P_L, and the prediction distribution for w_t with a noised lattice W_{t-1}^N and a specific G-gram tail S as $P_L(\cdot|W_{t-1}^N[S])$. In most parts of this paper, we will assume unigram ($G = 1$) just for notation simplicity.

3 LatticeGen

To prevent the server from gaining full knowledge of the generation and prompt, LatticeGen makes several core changes to the client–server interaction. On a high level, the server who possesses the LLG-finetuned LLM P_L (the finetuning is detailed in §A) still handles most of the computation, while the client controls the token sampling operations and expands the lattice to the next time-step. In particular, the client will sample one true token and $N - 1$ noise tokens, where $N \ge 2$ is a hyperparameter controlling the width of the lattice. **In the end, both the server and client obtain the same noised lattice W_T^N, but only the client knows which token is the true one for each time step.**

In the beginning, the server needs to share the vocabulary V with the client, but all other parameters or configurations of the LLM are not shared. We describe the protocol below.

3.1 Protocol

For simplicity, we first ignore the prompt part and assume the generation starts at the first token. In the beginning $t = 0$, both the server and client begin with an empty local lattice, and the client sends N `<bos>` tokens to the server. We divide the client–server interaction at each time-step $t \ge 1$

into a *server step* and a *client step*, illustrated by Figure 2 (also see Algorithm 1).

Server Step From the last time-step, the server receives from client N tokens $\{w_{t-1}^1, ..., w_{t-1}^N\}$ and expands its local lattice to W_{t-1}^N. The server does not know which received token is the true token because the list is shuffled by the client, and computes the respective next-token prediction distribution for all N^G potential G-gram tails with the LLG format (each potential tail is denoted as S). More concretely, the lattice W_{t-1}^N is linearized, appended with each G-gram, and fed to P_L, which outputs $\{P_L(\cdot|W_{t-1}^N[S^i])\}_{i=1}^{N^G}$. [2]

Since all G-grams share the same linearized lattice, the inference can be made efficient by reusing transformer hidden states and parallel computing. We defer the details of finetuning and inference (both conducted by the server) to §A. The server represents the next-token prediction distributions as N^G length-$|V|$ vectors, and sends them back to the client.

Client Step Different from the server, the client knows which tokens are the true ones. Upon receiving the list of distribution vectors from the server, the client extracts the distribution for the true G-gram $P_L(\cdot|W_{t-1}^N[w_{(t-G)...(t-1)}^{\text{true}}])$, from which the client samples w_t^{true}. The client also need to generate $N-1$ "noise" tokens $\{w_t^{\text{noise(1)}}, ..., w_t^{\text{noise(N-1)}}\}$ with a certain noise scheme.

How to generate noise tokens is a key part of making the noised lattice robust to potential attacks from the server side. For now, we assume a simple synonym noise scheme in which we use synonyms of the true token. Concretely, w_t^{noise} is randomly sampled from S tokens having the closest embedding with w_t^{true} measured by cosine similarity. In our experiments we set $S = 5$. [3] In practice this simple noise scheme will be vulnerable to attacks from a malicious server. See §4 for discussions on attacks and our proposed advanced noise schemes for defense.

With a private random seed, the client randomly permutates the token list and sends it to the server. This concludes the client–server interaction in time-step t.

[2] In the uni-gram case, the notation simplifies to $\{P_L(\cdot|W_{t-1}^N[w_{t-1}^i])\}_{i=1}^N$.

[3] In practice, we exclude the first ten closest token in V, as their surface forms are usually very close to the true token, making the obfuscation useless (e.g., only different in capitalization).

Algorithm 1 Pseudo-code for LatticeGen

Input (Server): Lattice-finetuned LLM P_L, lattice width N, generation length T, and inference tail length G.
Input (Client): Prompt p, a noise generation scheme, a private large prime number for random seed.
Client sets $w_0^i :=$ <bos> for $1 \le i \le N$.
Both the server and client begin with an empty lattice.
The client sends $[w_0^1, ..., w_0^N]$ to server indicating the beginning of generation.
for $t = 1 \dots T$ **do**
  # Server Steps Below
  Receives $[w_{t-1}^1, ..., w_{t-1}^N]$ from client and use it to extend the lattice to W_{t-1}^N.
  For each G-gram tail S^i, run next-token inference on P_L with the LLG format and obtain $\{P_L(\cdot|W_{t-1}^N[S^i])\}_{i=1}^{N^G}$.
  Send the distributions to the client as N^G length-$|V|$ vectors.
  # Client Steps Below
  Receives the next-token distributions $\{P_L(\cdot|W_{t-1}^N[S^i])\}_{i=1}^{N^G}$ from server.
  if $t \le \text{len}(p)$ **then**
    Set $w_t^{\text{true}} := p_t$.
  else
    Sample w_t^{true} from $P_L(\cdot|W_{t-1}^N[w_{(t-G)...(t-1)}^{\text{true}}])$.
  end if
  Generate $N-1$ noise tokens $\{w_t^{\text{noise(1)}}, ..., w_t^{\text{noise(N-1)}}\}$ with the noise scheme.
  Set the current private random seed to be t multiplied by the private prime number.
  Obtain the permuted list $[w_t^1, ..., w_t^N]$ using the current random seed.
  Extend the local lattice, and send $[w_t^1, ..., w_t^N]$ to the server.
end for
Output (Server): Lattice W_T^N.
Output (Client): True sequence $\{w_t^{\text{true}}\}_{t=1}^N$, and lattice W_T^N.

Incorporating Prompts (Client) The incorporation of prompts is quite straightforward by regarding it as a prefix of the generation, and the content in the prompt can also be noised and protected by LatticeGen. See §B.1 for implementation details.

We summarize the LatticeGen protocols as pseudo-code in Algorithm 1. The discussion on the network communication cost between client and server is deferred to §B.2 to save space.

3.2 Comparison with Standard LM: History Noised While Locally Sharp

It is helpful to formulate a comparison between LatticeGen (P_L) and generation from a standard autoregressive LM P_M. For simplicity, we ignore the noise generation (i.e., lattice-building) part, and only care about how the true tokens are generated with P_L. Under this simplification, the probability of generating a true sequence w is:

$$\log P_L(w) \approx \sum_{t=1}^{T} \log P_L(w_t|W_{t-1}^N[w_{(t-G)...(t-1)}]), \quad (2)$$

where the forming process of W_{t-1}^N (noise tokens and permutation) at each time-step is omitted.

For comparison, the log-probability of generating w with the standard model P_M is:

$$\log P_M(w) = \sum_{t=1}^{T} \log P_M(w_t|w_{0...t-2}, w_{t-1}). \quad (3)$$

Comparing the above two equations with similar structure, it should be clear that what LatticeGen does is essentially blurring the token history $w_{0...t-2}$ by the noised lattice W^N_{t-2}. Therefore, increasing the number of noise tokens gives better protection for the true token sequence, but at the same time degrades the LM's performance.

While the history is blurred, the local sharpness (Khandelwal et al., 2018) is preserved by LatticeGen: From Equation 2, the exact last G tokens is provided to the model. Therefore, in the worst-case scenario (zero utilization of non-immediate history), LatticeGen is at least as strong as a $(G+1)$-gram LM.

4 Attack and Defense

In this section, we discuss potential attack algorithms from a hypothetically malicious server to decode the true token sequence $\{w^{\text{true}}_t\}^T_{t=1}$ hidden in the lattice W^N_T, and the client's noise generation schemes as defense. For notational simplicity, we will assume unigram ($G = 1$), and the extension to $G > 1$ should be straightforward. We first establish metrics to measure the strength of attacks.

Metrics Given a lattice W^N_T, the attacker's target is to decode a hypothesis sequence $\hat{w}$ with $\hat{w}_t \in \{w^1_t, ..., w^N_t\}$ having biggest overlap with the true generation w^{true}. We define a simple *true-ratio* metric to measure the strength of the attack:

$$\text{true-ratio}(\hat{w}, w^{\text{true}}) = \frac{\sum_{t=1}^{T} \mathbb{1}_{\hat{w}_t = w^{\text{true}}_t}}{T}. \quad (4)$$

In the repeated beam search attack to be described below, the result of the attack algorithm is not only one but N sequences $\{\hat{w}^i\}^N_{i=1}$ which spans the whole lattice (i.e., $\{\hat{w}^i_t\}^N_{i=1} = \{w^i_t\}^N_{i=1}$). In this case, we argue that the defending noise scheme should prevent *any* of the hypothesis from having a high overlap with the true sequence, and measure it with the *max-true-ratio*: [4]

$$\text{max-true-ratio}(\{\hat{w}\}^N_{i=1}, w^{\text{true}}) = \max_i \frac{\sum_{t=1}^{T} \mathbb{1}_{\hat{w}^i_t = w^{\text{true}}_t}}{T}. \quad (5)$$

It should be clear that $\frac{1}{N}$ is a lower bound for max-true-ratio for any noise scheme, which provides an intuition of why larger N would better protect the true sequence.

Albeit intuitive, a big weakness of the true-ratio metric is that it only considers exact matches and does not reflect the semantic similarity between the hypothesis and the true generation. Therefore, in our experiments we will also use an embedding-based metric BERTScore (Zhang* et al., 2020) to measure the leaked information on semantics. Similar to true-ratio, BERTScore is larger than zero and has a maximum value of 1 (we refer readers to its paper for details). We define max-BERTScore in the same fashion as max-true-ratio and we omit the formulation for brevity.

[4] The average of the true-ratio will always be $\frac{1}{N}$ because each true token is in one of the N hypotheses.

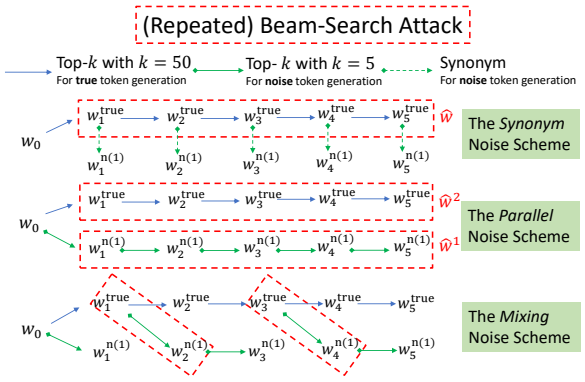

Figure 3: Illustration of different noise schemes under (repeated) beam-search attack. For convenience, the lattice is not shuffled on each time-step. An illustration with a width-3 lattice is given in Figure 7 (§B).

4.1 The Repeated Beam-Search Attack

In this section, we motivate and describe the *repeated beam-search attack* which is the major attack algorithm considered in this work. It is a stronger version of the *beam-search attack* described below.

The Beam-Search Attack (Server) Assuming unigram unit, a natural objective for the attacker is to find the sequence $\hat{w}$ with $\hat{w}_t \in \{w^1_t, ..., w^N_t\}$ which is mostly likely to be generated by P_L:

$$\arg\max_{\hat{w}} \log P_L(\hat{w}|W^N_T) = \arg\max_{\hat{w}} \sum_{t=1}^{T} \log P_L(\hat{w}_t|W^N_{t-1}[\hat{w}_{t-1}]). \quad (6)$$

By saving all probability distributions during the generation, the attacker can efficiently conduct this optimization using the classical beam-search algorithm. We term it as the *beam-search attack*.

Our experiments (§5) show that the simple synonym noise scheme discussed in §3 is highly vulnerable to the beam-search attack. We show some intuition in the upper part of Figure 3: There does not exist a direct link between the noise tokens. The log-probability of the true sequence will likely

be much higher than any combination of the noise tokens, and is therefore revealed by the attack.

The Parallel Noise Scheme (Client) There is an intuitive way to defend against the beam-search attack: The client can sample a noise sequence independent of the true sequence, and make it have higher log-probability than the true sequence by tuning the hyper-parameter of the sampling algorithm. We term it the *parallel noise scheme* and illustrate in the middle of Figure 3.

More concretely, at time-step t, the i-th noise token is sampled from $P_L(\cdot|W_{t-1}^N[w_{t-1}^{\text{noise(i)}}])$. [5] In this way, the noise sequences $w^{\text{noise(i)}}$ are parallel and independent of the true sequence w^{true}. We also assume the adoption of popular sampling hyper-parameter for the generation of the true sequence (e.g., $k = 50$ for top-k or $p = 0.96$ for nucleus), which enables the adoption of a more radical hyper-parameter (Caccia et al., 2020; Nadeem et al., 2020) for the sampling of the noise sequences: in our experiments we use $k = 5$.

Our experiments show that the parallel noise sequences can very effectively hide the true sequence from the beam-search attack. This motivates our proposed repeated beam-search attack.

The Repeated Beam-Search (RBS) Attack (Server) We propose a simple but more powerful attack algorithm based on the beam-search attack: Given a N-lattice, we do beam-search $N-1$ times. After obtaining the resulting hypothesis sequence of the i-th beam-search (denoted as $\hat{w}^i$), we remove the tokens in $\hat{w}^i$ from the lattice, resulting in a $(N-i)$-lattice. After the $(N-1)$-th beam-search, only one sequence is left in the lattice, which becomes the N-th hypothesis $\hat{w}^N$. We term it the repeated beam-search (RBS) attack.

The intuition of why the RBS attack is effective against the parallel noise scheme is shown in the middle of Figure 3. Since the noise sequences are of high probability and independent of each other, it is likely that the $N-1$ times of beam-search would obtain all the noise sequences as hypotheses which are removed from the lattice in turn, and the remaining true sequence is therefore revealed in the end as $\hat{w}^N$. This would result in a high max-true-ratio.

4.2 The Mixing Noise Scheme for Defense

We propose the *mixing noise scheme* to defend against the RBS attack, with the intuition that the true and noise sequences should somehow be mixed. This scheme can be regarded as a variant of the parallel noise scheme. Again we adopt a radical hyper-parameter for the sampling of the noise sequences (top-k with $k = 5$). At time-step t, with a random ratio determined by a hyper-parameter *mix-ratio*, the i-th noise token is sampled from $P_L(\cdot|W_{t-1}^N[w_{t-1}^{\text{true}}])$, **which is the next-token distribution for the true sequence**. [6] Otherwise we sample from $P_L(\cdot|W_{t-1}^N[w_{t-1}^{\text{noise(i)}}])$, same as in the parallel scheme.

We illustrate this at the bottom of Figure 3. In comparison to the parallel scheme, the goal is to make the sequence with the highest log-probability be a mix between the true and noise sequences. And the key is to make the true sequence "branch" out to the noise sequences, which breaks the continuity of the noise sequences. Although broken, the radical sampling used for the noise sequence would still attract the repeated beam-search attack, and the true and noise sequences are mixed by the branching connections. Our experiments show that with a tuned mix-ratio, the mixing noise scheme achieves the best max-true-ratio under RBS attack.

5 Experiments

5.1 Experiment Setting

Model & Noise Schemes We use the OPT-1.3B (Zhang et al., 2022) and the Llama2-7B model as our base LLM, from which both P_L and P_M are finetuned. We select those models due to limited computing resource and as a proof-of-concept. Our protocol can be readily applied to larger autoregressive LMs such as GPT3 or GPT4. In our implementation, for convenience we simulate the client–server interaction protocols on a single machine.

For sampling of the true sequence, we use top-k (Fan et al., 2017) sampling with $k = 50$, temperature 0.7, and a repetition penalty of 1.05. For the noise token sampling in the parallel or mixing noise scheme, $k = 5$ is used. It should be clear that LatticeGen can also be applied to other sampling algorithms with proper hyper-parameters. We limit the maximum generation length to 60 tokens. For the mixing noise scheme of OPT, we use a mix-ratio of 0.1 for both $N = 2$ and $N = 3$ for the generation part. For the prompt part, we use a mix-ratio of 0.2. For Llama2, we use a mix-ratio of 0.05 for both $N = 2$ and $N = 3$ for the generation

[5] If $G > 1$, the last G tokens from the i-th the noise sequence will be used.

[6] We will re-sample if the sampled token is the same as the true token.

Config	$N = 2$ (LG only)						$N = 3$ (LG only)					
Metric	PPL	PMI	True-Ratio		BERTScore		PPL	PMI	True-Ratio		BERTScore	
Attack			BS	RBS	BS	RBS			BS	RBS	BS	RBS
OPT, Vanilla (P_M), w.o. noise	21.272	.345	1.0	1.0	1.0	1.0	/	/	/	/	/	/
OPT, Synonym, w.o. lattice	229.616	.058	/	/	/	/	/	/	/	/	/	/
OPT, Syn-50%, w.o. lattice	199.621	.058	/	/	/	/	/	/	/	/	/	/
OPT, LG, 4-gram, synonym	37.574	.244	.993	.993	.894	.894	41.379	.221	.985	.985	.882	.882
OPT, LG, 4-gram, parallel	33.907	.228	.168	.844	.234	.784	35.691	.232	.110	.749	.155	.676
OPT, LG, 4-gram, **mixing**	34.058	.219	.541	**.651**	.432	**.531**	35.910	.242	.357	**.511**	.285	**.393**
Llama2, Vanilla (P_M), w.o. noise	14.710	.785	1.0	1.0	1.0	1.0	/	/	/	/	/	/
Llama2, LG, 4-gram, synonym	22.297	.661	.995	.995	.895	.895	27.125	.585	.986	.986	.880	.880
Llama2, LG, 4-gram, parallel	22.649	.637	.145	.870	.211	.811	25.962	.683	.122	.751	.165	.672
Llama2, LG, 4-gram, **mixing**	22.430	.670	.499	**.713**	.440	**.618**	26.997	.648	.360	**.565**	.262	**.410**

Table 1: Main results when LatticeGen (LG) is applied to both the generation and the prompt. All metrics are the lower the better except PMI. While the generation quality and alignment are degraded, LatticeGen with the proposed mixing scheme successfully protects the true generation from RBS attack to a remarkable degree (measured by max-true-ratio/BERTScore).

part and 0.2 for the prompt part. They are found to achieve the lowest max-true-ratio on the dev set.

Dataset & Lattice Finetuning Since the word history is noised (discussed in §3.2), LatticeGen is not recommended for tasks with high requirements for consistency or factuality (Pagnoni et al., 2021). In this work we focus on the task of creative writing (Martin et al., 2017; Yao et al., 2018; Fan et al., 2019), and utilize the WritingPrompts dataset (Fan et al., 2018). The dataset is composed of stories and the corresponding high-level descriptions as prompts. The average length of prompts/stories is 29/674. We use 200/500 samples from the valid/test set for development/evaluation. The training set (10,000 samples) is used for finetuning of P_L and P_M, and we defer details to §A.

Metrics We use a larger LLM, namely OPT-2.7B or Llama2-13B, to measure the generation's quality or alignment with the prompt. For quality, we use the popular perplexity metric. For alignment, we use pointwise mutual information (PMI) (Shi et al., 2023):

$$\text{PMI}_{\text{OPT}}(x; y) = \frac{\log P_{\text{OPT}}(x|y) - \log P_{\text{OPT}}(x)}{\text{len}(x)}, \quad (7)$$

where x and y denote the generation and prompt.

To compare between different noise schemes and measure the (semantic) overlap between the attack hypothesis ($\hat{w}$) and the true sequence (w^{true}) under RBS attack, we use the true-ratio or BERTScore discussed in §4. We will report true-ratio for the BS attack and max-true-ratio under RBS attack, and the same applies to BERTScore.

5.2 Experiment Results

Table 1 includes the main results when LatticeGen (LG) is applied to both generation and prompt. The standard vanilla model (P_M) enjoys the best generation quality (PPL and PMI), while having zero obfuscation (100% true-ratio).

LatticeGen sacrifices generation quality (due to noised history) for obfuscation. The empirical behavior of the three noise schemes aligns with their respective intuitions discussed in §4: The synonym scheme is completely defenseless against the BS attack; The parallel scheme is most effective against BS with true-ratio lower than 20%, but is vulnerable under the stronger RBS attack.

The mixing scheme, which is our main recommended scheme, achieves the best protection under the RBS attack. For $N = 2$, The max-true-ratio/BERTScore is close to or lower than 65%/55%. **It indicates that around half of the semantic is hidden from the attacker, and is close to the theoretical best max-true-ratio ($\frac{1}{N} = 50\%$).** The protection is better with $N = 3$ (50%/40%), but with worse generation quality.

Comparing to unigram unit, **the quality degradation (especially PPL) is alleviated to a large degree by using 4-gram units** (See Figure 5 for a comparison). One could also try larger G-gram for further improvement. However, the computational cost would grow exponentially and we leave it to future work due to limited resources.

What if we directly apply noise to generation but *without the lattice structure*? We add an additional non-lattice baseline with the same synonym scheme used in LatticeGen: On every time-step, the

Prompt: Prompt: Aliens have arrived, and ask for a single human to plead humanity's case and save them from extinction. The human is selected through a lottery of the entire human race, and on the day of the drawing, your name is picked... Story:

Generated Text (P_M): I could feel my heart rate increase . A cold sweat ran down my back . I could not believe what was happening . My name had just been drawn . Everyone 's names were in a big bowl , with the most common names at the top , to the least common at the bottom

Generated Text (LG): I can see them . They are here to save us from our own destruction , but to watch over us . " " Why have you come ? What is so important about humans ? "' " Humanity has been here since the beginning . They took us by surprise a few years ago .

First Round RBS: Prompt: *Aliens have arrived* on the cover of every *single human* , and they all have a different colour. Story: *from extinction . The human is selected through a lottery of the entire human race, and on the day of the drawing* is the room with the blue Story: " We have come in peace . They are not hostile . " " I do n't know " " Why have you come ? *' What is so important about humans ? "' " Humanity has been here since the beginning . They took us by surprise a few years ago .*

Second Round RBS: Prompt: Youenstein *', and ask for a* meeting room *to plead humanity's case and save them* " theint. . " The .gov drawing room . all the walls are painted with you and you can your choice, *your name is picked...* Story: *I can see them . They are here to save us from our own destruction , but to watch over us* .We 're here to protect been so peaceful and gentle ? " " They 're a threat to us . " " But we were n' million species from the

Figure 4: An example of text generation with LatticeGen, using the configuration of 4-gram, N=2 and the mixing scheme. The true tokens are italicized in both rounds of RBS, and the underline indicates that the noise token is mixed from the previous true token. Note that the prompt is also noised by LG.

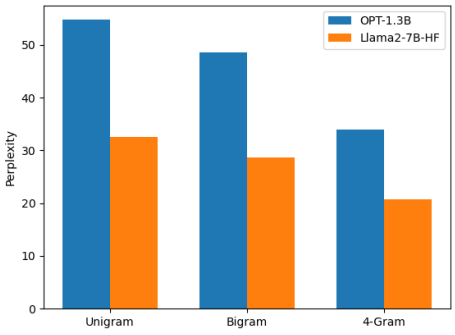

Figure 5: Comparison of perplexity of OPT-1.3B and Llama-7B-HF models on various G-gram units.

client gets next-token distribution from the server and generates a true token, but sends a synonym of it back to the server. The finetuning is modified accordingly with details given in §B.3.

As shown in Table 1, we apply the synonym scheme to 100% or 50% of the tokens. The synonym noise without lattice results in drastically degraded PPL and PMI. In comparison, LatticeGen provides a trade-off between quality degradation and privacy protection. This implies that **for decent generation performance, the true tokens have to be revealed to the server in some way**.

Table 2 (§D) compares generation speed of different systems. On the single A40 GPU we use, LG with 4-gram ($N = 2$) units has a 4.76 times slowdown comparing to P_M. Since inference with transformer model benefits from parallel computing, the slowdown should be less significant on servers with stronger computing power.

We show a generation example with RBS attack outputs in Figure 4. LG is able to generate a sample with decent quality. More importantly, around half of the story semantics remains hidden from the RBS attack by the mixing noise scheme. More examples and analysis are deferred to §D.

6 Related Work

Existing work in privacy-aware natural language processing (NLP) (Qu et al., 2021; McMahan et al., 2017) mostly focuses on protecting user data for training (Huang et al., 2020; Yue et al., 2023) or inference, and the majority of works focus on natural language understanding (NLU) tasks (Feyisetan et al., 2020; Xu et al., 2021). To the best of our knowledge, our work is the first to consider decoding-time privacy for LLM prompted generation on cloud.

Lattice in NLP Lattice (Young et al., 2006) is a graphical structure widely used in structured prediction problems to represent a range of hypotheses. In this work we adopt a simple linear-graph form of lattice which is known as the confusion network (Mangu et al., 1999). The lattice structure has found interesting applications in neural NLP models. As a pioneering work, Su et al. (2017) proposes lattice-based RNN encoders for machine translation, where the lattice is generated by merging results from different segmenters. Buckman & Neubig (2018) proposes a neural lattice language model, which constructs a lattice of possible paths (segmentations) through a sentence in order to model multiple granularities. Lattice-BERT (Lai et al., 2021) trains LLM to predict a masked portion of a lattice representing possible segmentations of a sentence. To the best of our knowledge, our work is the first to utilize the lattice structure for

privacy-aware generation.

Prompt Anonymization Contemporary and independent of our work, Chen et al. (2023) proposes to anonymize the named entities (e.g., change `USA` to `<GPE>`) in the prompt, and de-anonymize after receiving the generated text from server. In comparison, LatticeGen offers a more general option in that all types of tokens, especially the generated tokens, can be noised.

Due to lack of space, we discuss related work on **differential privacy**, **homomorphic encryption** in §C.

7 Conclusion

LatticeGen aims for an ambitious and seemingly conflicting goal: The server still does most computation for the generation but does not know what exactly is generated. This is achieved by our proposed noised lattice structure, and a cooperative generation protocol between the server and client.

While the noised lattice degrades generation quality and inference speed, LatticeGen with our proposed mixing noise scheme successfully prevents a malicious server from recovering the true generation to a remarkable degree (more than 50% of the semantic remains unknown as measured by BERTScore). We hope our work could inspire more research into this under-studied yet important field of privacy-aware LLM generation on cloud.

8 Limitations

LatticeGen sacrifices generation quality and speed for obfuscation of generated contents. While we show the quality degradation can be alleviated to some degree by using larger G-gram unit, it would also cause the inference computation to grow exponentially. An interesting future direction is that, instead of running an inference for all N^G grams, we only select a small portion strategically.

On the other hand, in this work we focus on protecting the user and the (repeated) beam-search attack from server. There could be other forms of interesting or stronger attacks on the server side (e.g., manual inspection from a human). On the other hand, sharing generation control with client could also endanger the server (e.g., jailbreaking) (Liu et al., 2023; Li et al., 2023).

Finally, in the current implementation, we lattice-finetune a seperate OPT model for every different lattice configuration, which is space unfriendly. As future work, it would be interesting to explore a unified format of linearized lattice by which a single LLM can process different lattice configurations.

9 Broader Impact

As stated in §1, in the current user–server interaction paradigm, both the prompt and the generation are raw texts which are completely transparent and accessible to the server. This leaves zero options for users who want to keep the generated text to themselves. On the other hand, the privacy protection offered by today's LLM providers' data usage and retention policies is far from enough (detailed in §E). We propose LatticeGen as a novel protocol for privacy-aware generation with a controlled level of obfuscation. We hope our work could raise awareness for the privacy considerations of generated contents.

Acknowledgments

We sincerely thank Peihan Miao, Xiaochuang Han, and Kyunghyun Cho for useful discussions. This research is supported in part by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via the HIATUS Program contract #2022-22072200004. This material is also funded by the DARPA Grant under Contract No. HR001120C0124. We also gratefully acknowledge support from NSF CAREER Grant No. IIS2142739, NSF Grants No. IIS2125201, IIS2203097, and the Alfred P. Sloan Foundation Fellowship. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

References

Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, oct 2016. doi: 10.1145/2976749.2978318. URL https://doi.org/10.1145%2F2976749.2978318.

Jacob Buckman and Graham Neubig. Neural lattice language models. *Transactions of the Association for Computational Linguistics*, 6:529–541,

2018. doi: 10.1162/tacl_a_00036. URL https://aclanthology.org/Q18-1036.

Massimo Caccia, Lucas Caccia, William Fedus, Hugo Larochelle, Joelle Pineau, and Laurent Charlin. Language gans falling short. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BJgza6VtPB.

Tianyu Chen, Hangbo Bao, Shaohan Huang, Li Dong, Binxing Jiao, Daxin Jiang, Haoyi Zhou, Jianxin Li, and Furu Wei. THE-X: Privacy-preserving transformer inference with homomorphic encryption. In *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 3510–3520, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.277. URL https://aclanthology.org/2022.findings-acl.277.

Yu Chen, Tingxin Li, Huiming Liu, and Yang Yu. Hide and seek (has): A lightweight framework for prompt privacy protection, 2023.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1179. URL https://aclanthology.org/D14-1179.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling instruction-finetuned language models, 2022.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *CoRR*, abs/1901.02860, 2019. URL http://arxiv.org/abs/1901.02860.

Yuntian Deng, Volodymyr Kuleshov, and Alexander Rush. Model criticism for long-form text generation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11887–11912, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. URL https://aclanthology.org/2022.emnlp-main.815.

Angela Fan, David Grangier, and Michael Auli. Controllable abstractive summarization. *arXiv preprint arXiv:1711.05217*, 2017.

Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 889–898, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1082. URL https://aclanthology.org/P18-1082.

Angela Fan, Mike Lewis, and Yann Dauphin. Strategies for structuring story generation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2650–2660, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1254. URL https://aclanthology.org/P19-1254.

Oluwaseyi Feyisetan, Borja Balle, Thomas Drake, and Tom Diethe. Privacy- and utility-preserving textual analysis via calibrated multivariate perturbations. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, WSDM '20, pp. 178–186, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450368223. doi: 10.1145/3336191.3371856. URL https://doi.org/10.1145/3336191.3371856.

Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pp. 169–178, 2009.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rygGQyrFvH.

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, 2021.

Yangsibo Huang, Zhao Song, Danqi Chen, Kai Li, and Sanjeev Arora. TextHide: Tackling data privacy in language understanding tasks. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1368–1382, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.123. URL https://aclanthology.org/2020.findings-emnlp.123.

Ferenc Huszár. How (not) to train your generative model: Scheduled sampling, likelihood, adversary?, 2015.

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. *CoRR*, abs/2001.08361, 2020. URL https://arxiv.org/abs/2001.08361.

Gavin Kerrigan, Dylan Slack, and Jens Tuyls. Differentially private language models benefit from public pre-training. In *Proceedings of the Second Workshop on Privacy in NLP*, pp. 39–45, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.privatenlp-1.5. URL https://aclanthology.org/2020.privatenlp-1.5.

Nitish Shirish Keskar, Bryan McCann, Lav R. Varshney, Caiming Xiong, and Richard Socher. CTRL: A conditional transformer language model for controllable generation. *CoRR*, abs/1909.05858, 2019. URL http://arxiv.org/abs/1909.05858.

Urvashi Khandelwal, He He, Peng Qi, and Dan Jurafsky. Sharp nearby, fuzzy far away: How neural language models use context. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 284–294, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1027. URL https://aclanthology.org/P18-1027.

Yuxuan Lai, Yijia Liu, Yansong Feng, Songfang Huang, and Dongyan Zhao. Lattice-BERT: Leveraging multi-granularity representations in Chinese pre-trained language models. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1716–1731, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.137. URL https://aclanthology.org/2021.naacl-main.137.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding, 2023.

Haoran Li, Dadi Guo, Wei Fan, Mingshi Xu, Jie Huang, Fanpu Meng, and Yangqiu Song. Multi-step jailbreaking privacy attacks on chatgpt, 2023.

Xuechen Li, Florian Tramer, Percy Liang, and Tatsunori Hashimoto. Large language models can be strong differentially private learners. *arXiv preprint arXiv:2110.05679*, 2021.

Yi Liu, Gelei Deng, Zhengzi Xu, Yuekang Li, Yaowen Zheng, Ying Zhang, Lida Zhao, Tianwei Zhang, and Yang Liu. Jailbreaking chatgpt via prompt engineering: An empirical study, 2023.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Lidia Mangu, Eric Brill, and Andreas Stolcke. Finding consensus among words: lattice-based word error minimization. In *EUROSPEECH*, 1999. URL https://api.semanticscholar.org/CorpusID:13137367.

Lara J. Martin, Prithviraj Ammanabrolu, William Hancock, Shruti Singh, Brent Harrison, and Mark O. Riedl. Event representations for automated story generation with deep neural nets. *CoRR*, abs/1706.01331, 2017. URL http://arxiv.org/abs/1706.01331.

H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private language models without losing accuracy. *CoRR*, abs/1710.06963, 2017. URL http://arxiv.org/abs/1710.06963.

Casey Meehan, Khalil Mrini, and Kamalika Chaudhuri. Sentence-level privacy for document embeddings. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3367–3380, 2022.

Tomáš Mikolov. *Statistical language models based on neural networks*. PhD thesis, Brno University of Technology, 2012.

Fatemehsadat Mireshghallah, Yu Su, Tatsunori Hashimoto, Jason Eisner, and Richard Shin. Privacy-preserving domain adaptation of semantic parsers. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4950–4970, Toronto, Canada, July 2023. Association for Computational Linguistics. URL https://aclanthology.org/2023.acl-long.271.

Moin Nadeem, Tianxing He, Kyunghyun Cho, and James Glass. A systematic characterization of sampling algorithms for open-ended language generation. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pp. 334–346, Suzhou, China, December 2020. Association for Computational Linguistics. URL https://aclanthology.org/2020.aacl-main.36.

OpenAI. Gpt-4 technical report, 2023.

Artidoro Pagnoni, Vidhisha Balachandran, and Yulia Tsvetkov. Understanding factuality in abstractive summarization with FRANK: A benchmark for factuality metrics. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4812–4829, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.383. URL https://aclanthology.org/2021.naacl-main.383.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward

Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.

Chen Qu, Weize Kong, Liu Yang, Mingyang Zhang, Michael Bendersky, and Marc Najork. Natural language understanding with privacy-preserving bert. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, CIKM '21, pp. 1488–1497, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384469. doi: 10.1145/3459637.3482281. URL https://doi.org/10.1145/3459637.3482281.

Weijia Shi, Xiaochuang Han, Mike Lewis, Yulia Tsvetkov, Luke Zettlemoyer, and Scott Wen tau Yih. Trusting your evidence: Hallucinate less with context-aware decoding, 2023.

Jinsong Su, Zhixing Tan, Deyi Xiong, Rongrong Ji, Xiaodong Shi, and Yang Liu. Lattice-based recurrent neural network encoders for neural machine translation. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, pp. 3302–3308. AAAI Press, 2017.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 30*, pp. 5998–6008. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

Sean Welleck, Ilia Kulikov, Stephen Roller, Emily Dinan, Kyunghyun Cho, and Jason Weston. Neural text generation with unlikelihood training. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SJeYe0NtvH.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6. URL https://aclanthology.org/2020.emnlp-demos.6.

Zekun Xu, Abhinav Aggarwal, Oluwaseyi Feyisetan, and Nathanael Teissier. A differentially private text perturbation method using regularized mahalanobis metric. In *Proceedings of the Second Workshop on Privacy in NLP*, pp. 7–17, 2020.

Zekun Xu, Abhinav Aggarwal, Oluwaseyi Feyisetan, and Nathanael Teissier. On a utilitarian approach to privacy preserving text generation. In *Proceedings of the Third Workshop on Privacy in Natural Language Processing*, pp. 11–20, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.privatenlp-1.2. URL https://aclanthology.org/2021.privatenlp-1.2.

Lili Yao, Nanyun Peng, Ralph M. Weischedel, Kevin Knight, Dongyan Zhao, and Rui Yan. Plan-and-write: Towards better automatic storytelling. *CoRR*, abs/1811.05701, 2018. URL http://arxiv.org/abs/1811.05701.

Steve J. Young, D. Kershaw, J. Odell, D. Ollason, V. Valtchev, and P. Woodland. *The HTK Book Version 3.4*. Cambridge University Press, 2006.

Da Yu, Saurabh Naik, Arturs Backurs, Sivakanth Gopi, Huseyin A Inan, Gautam Kamath, Janardhan Kulkarni, Yin Tat Lee, Andre Manoel, Lukas Wutschitz, et al. Differentially private fine-tuning of language models. *arXiv preprint arXiv:2110.06500*, 2021.

Xiang Yue, Huseyin Inan, Xuechen Li, Girish Kumar, Julia McAnallen, Hoda Shajari, Huan Sun, David Levitan, and Robert Sim. Synthetic text generation with differential privacy: A simple and practical recipe. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1321–1342, Toronto, Canada, July 2023. Association for Computational Linguistics. URL https://aclanthology.org/2023.acl-long.74.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. Opt: Open pre-trained transformer language models, 2022.

Tianyi Zhang*, Varsha Kishore*, Felix Wu*, Kilian Q. Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SkeHuCVFDr.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen

Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A survey of large language models, 2023.

Supplemental Materials

A Model Training and Inference with Lattice (Server)

LLM Finetuning and Inference with the LLG (Linearized Lattice plus G-gram) Format We now describe how P_L is obtained by finetuning a standard autoregressive LM P_M parameterized by θ to make next-token predictions with the LLG format(§2.3). We assume access to a public corpus D for finetuning. For simplicity, we focus on the training objective for one length-T sentence $w^d \in D$ and we also assume $N = 2$ and $G = 1$ (the process for $N > 2$ or $G > 1$ is highly similar).

For each data sample w^d, we randomly pick another data sample $w^{d'}$ to serve as a "parallel" noise sample, which is used for constructing the noised lattice W_T^2 for w^d. For time-step t, the token in the data sample w^d will be used as the true token $w_t^{\text{true}} := w_t^d$, and the token from the parallel sample is used as the noise token $w_t^{\text{noise(1)}} := w_t^{d'}$. To be consistent with the actual generation protocols for LatticeGen, the tokens on each time-step are shuffled.

The noise generation scheme used by server in the finetuning stage might be different from the scheme used by client in the actual generation. For example, if we use a simple synonym scheme, the perplexity of the synonym scheme during generation will be better. In our implementation we adopt the parallel scheme described above during training because it works well with the proposed mixing scheme (§4.2).

After constructing the noised lattice W_T^2, we randomly select P tokens in w^d (we use $P = 8$ in our training), and use them as the target next-tokens to finetune the LLM with the LLG format. Denoting their indices as $\{t^1, ..., t^P\}$, we formulate the following objective:

$$\mathcal{L}_{\text{lattice-FT}}(w^d, W_T^2; \theta) = \frac{1}{P}\sum_{p=1}^{P} \log P_\theta(w_{t^p}^{\text{true}} | W_{t^p-1}^2[w_{t^p-1}^{\text{true}}]). \quad (8)$$

We now discuss how the server can do efficient LLM inference at time-step t. Since linearize(W_{t-2}^N) from the previous time-step $t-2$ is a prefix of linearize(W_{t-1}^N), the server can reuse the saved LLM hidden states[7] from the last time-step for the inference of $\{P_L(\cdot|W_{t-1}^N[w_{t-1}^i])\}_{i=1}^N$. However, the server still need to enumerate and inference N^G combinations of the G-grams in parallel, and that is the major reason for the slowdown.

[7]The `past_key_values` in HuggingFace transformers library.

Implementation Details Our model implementation, training and inference utilize the HuggingFace transformers library (Wolf et al., 2020). We finetune P_L with learning rate of 5×10^{-5} and a batch size of 8 for 3 epochs using the PyTorch (Paszke et al., 2019) implementation of the AdamW (Loshchilov & Hutter, 2017) optimizer. For finetuning of Llama2, we adopt LoRA (Hu et al., 2021). We perform finetuning of the model under various configurations on one Nvidia A40 GPU.

B Auxiliary Framework Description

An illustration of the server step for $N = 3$ and $G = 2$ is provided in Figure 6.

An illustration of various noise schemes with a width-3 lattice is provided in Figure 7.

B.1 Incorporating the Prompt (Client)

The prompt p can be easily incorporated by the following. At all time-steps t with $t \le \text{len}(p)$, instead of sampling w_t^{true} from $P_L(\cdot|W_{t-1}^N[w_{(t-G)...(t-1)}^{\text{true}}])$, the client directly sets $w_t^{\text{true}} := p_t$. All other steps in the protocols including the noise token generation continue as normal. In this way, the prompt is also embedded and noised in the lattice.

B.2 Communication Cost

At each time-step, the server needs to send client N^G length-$|V|$ vectors, which could be slow if $|V|$ is large. This can be largely alleviated if the client and server can agree upon a sampling algorithm beforehand. For example, if top-k sampling with $k = 50$ is used, then only the logits and indices of the top-50 tokens are needed.

B.3 The Non-Lattice Baseline

The training for the non-lattice baseline is a bit similar to the lattice finetuning process described in §A, with the difference that the true tokens are not included in the input. Following the notations in §A with w^d as the data sample, the training objective is formulated as:

$$\mathcal{L}_{\text{non-lattice,syn.}}(w^d; \theta) = \frac{1}{T}\sum_{t=1}^{T} \log P_\theta(w_t^d | w_{0..t-1}^{\text{noise}}), \quad (9)$$

where w_t^{noise} is randomly set to a synonym of w_t^d. Basically, the model is trained to predict the next true token with a ratio of input tokens noised.

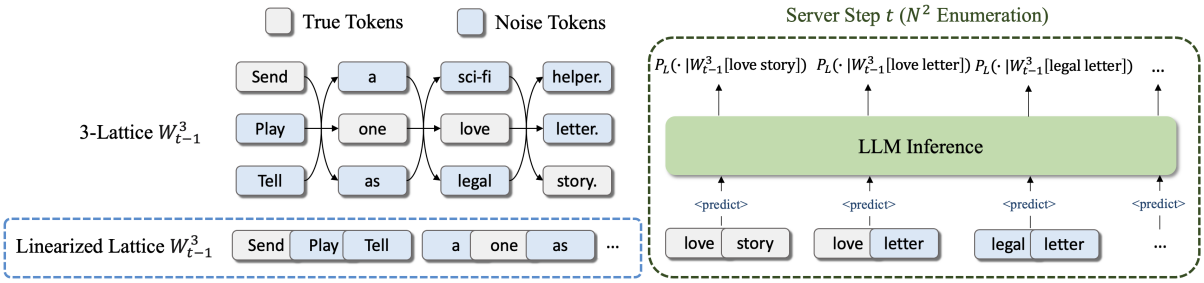

Figure 6: An illustration of the server step for $N = 3$ and $G = 2$. The information of which tokens are the true tokens is only known to the client.

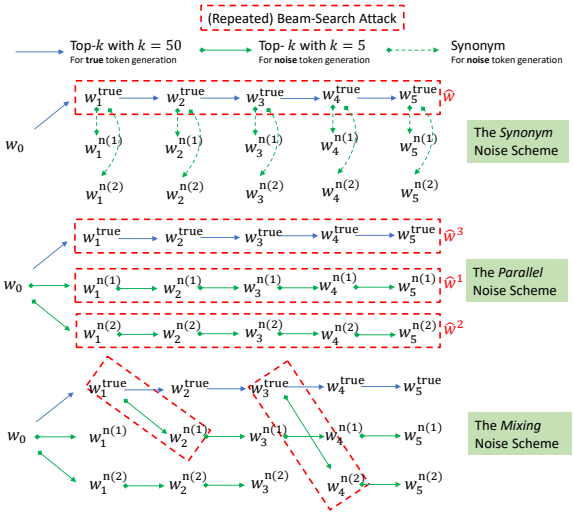

Figure 7: Illustration of different noise schemes under (repeated) beam-search attack. For convenience, the lattice is not shuffled.

C Related Work

This section continues from §6.

Differential Privacy (DP) for LM Training and Inference There are numerous existing works on how to train LLMs with differential privacy (Li et al., 2021; Yu et al., 2021), which mostly rely on DP-SGD (Abadi et al., 2016) and limits leakage of private data during training. More related to LatticeGen is a line of work with local DP (Xu et al., 2020; Meehan et al., 2022), which applies discrete noise onto text and can be used to synthesize private text data (Yue et al., 2023; Mireshghallah et al., 2023).

It is not directly clear how these techniques can be adapted for our setting of privacy-aware autoregressive text generation. In comparison, LatticeGen provides a totally different and cooperative approach with the lattice structure and novel defense and attack schemes.

Speed (second/token)	N=1	N=2	N=3
P_M	.013	/	/
LG, Unigram	/	.024 (1.84x)	.028 (2.15x)
LG, Bigram	/	.028 (2.15x)	.047 (3.62x)
LG, 4-gram	/	.062 (4.76x)	.332 (25.53x)

Table 2: Generation speed comparison between different systems. For LG, the mixing noise scheme and the OPT model is used. Our implementation is run on a single A40 GPU.

Homomorphic Encryption There is also a line of work (Chen et al., 2022) applying techniques from homomorphic encryption (Gentry, 2009) to transformer LM. While they enjoy nice cryptographic guarantees, the induced computational cost is usually huge.

D Auxiliary Results

Similar to Figure 4, Figure 8 shows an example using a different prompt using bigram $N = 2$.

On the single A40 GPU we use, LG with bigram units ($N = 2$) has a 2x slowdown comparing to P_M (Table 2, §D). Since inference with transformer model benefits from parallel computing, the slowdown should be less significant on servers with stronger computing power.

E The Current Privacy Protection Practices in Industry

The privacy protection offered by today's LLM providers' data usage and retention policies is far from enough. [8] For example, OpenAI's consumer-facing ChatGPT used to train its models with user input, and also shares user input with third-party providers, and Google's Bard retains user activity for at least 3 months. As a striking example,

[8] https://opaque.co/announcing-opaqueprompts-hide-your-sensitive-data-from-llms/

employees in Samsung reportedly shared sensitive code with OpenAI during their interaction with ChatGPT. [9] More recently, some of the users' conversations with Bard are mistakenly indexed and accessed by Google search. [10]

While providers have recently improved their security posture (e.g., OpenAI no longer uses data submitted via its API to train its model), users still can not assume that all sent/received data will be immediately and completely deleted. Rather than regulations, our proposed LatticeGen takes an algorithmic and cooperative approach to give the user advantage and control in privacy protection.

[9] https://gizmodo.com/chatgpt-ai-samsung-employees-leak-data-1850307376

[10] https://venturebeat.com/ai/oops-google-search-caught-publicly-indexing-users-conversations-with-bard-ai/

Prompt: Every planet in our solar system has a " champion " being that takes on the attributes of the planet itself. The " champion " from the sun has created an army to destroy the planets and the 8 (or 9) champions must save the solar system... Story:

Generated Text (P_M): The planet Mars was known for its reddish color . Mars has a very thin atmosphere , and only a select few had been able to breathe it . But not this man . This man could breathe anything . His name is Sol , also known as the Sun .

Generated Text (LG): " There 's nothing you can do , " I said , running through my head as I saw the soldiers fall . The soldiers were outnumbered , and his army too vast for us to even put up a fight and still lose ? It will be too late ! The champion is here ! "

First Round RBS: Prompt: *Every planet in* the galaxy *has a " champion " , that takes on the attributes of* all of the inhabitantsants *"* life *" from the sun has* taken up arms against him .. Story: " *the* 3 *(or 9) champions must save the solar system...* Story: *" There 's nothing you can do , " I said , running through my head as I saw the soldiers fall . The soldiers were* too powerful for us ! " " You can try ! " " What ? How ? " " You not only have to fight the champion , but his

Second Round RBS: Prompt: A man is *our solar system* 's life is a *being* ul, , , , , , *the planet itself. The . champion* on Earth each other to *created an army to destroy the planets and* I ca*8* other I '3m not are you Earthlingss from Story: The world was in chaos . say something ! " " No ! " " if we could have stopped him . He was *outnumbered , and his army too vast for us to even put up a fight and still lose ? It will be too late ! The champion is here ! "*

Figure 8: Another example of text generation with LatticeGen, using the configuration of 4-gram, N=2 and the the mixing scheme. The true tokens are italicized in both rounds of RBS, and the underline indicates that the noise token is mixed from the previous true token. Note that the prompt is also noised by LG.