NOIR: Privacy-Preserving Generation of Code with Open-Source LLMs

Khoa Nguyen[1⋆], Khiem Ton[1⋆], NhatHai Phan[1#], Issa Khalil[2], Khang Tran[1⋆],
Cristian Borcea[1], Ruoming Jin[3], Abdallah Khreishah[1], My T. Thai[4]
[1] *New Jersey Institute of Technology,* [2] *Hamad Bin Khalifa University,*
[3] *Kent State University,* [4] *University of Florida*
Emails: {nk569, kt477, kt36, borcea, abdallah}@njit.edu; ikhalil@hbku.edu.qa;
rjin1@kent.edu; mythai@cise.ufl.edu
[⋆]*Equal Contributions,* [#]*Corresponding Author (Email: phan@njit.edu)*

Abstract

Although boosting software development performance, large language model (LLM)-powered code generation introduces intellectual property and data security risks rooted in the fact that a service provider (cloud) observes a client's prompts and generated code, which can be proprietary in commercial systems. To mitigate this problem, we propose NOIR, the first framework to protect the client's prompts and generated code from the cloud. NOIR uses an encoder and a decoder at the client to encode and send the prompts' embeddings to the cloud to get enriched embeddings from the LLM, which are then decoded to generate the code locally at the client. Since the cloud can use the embeddings to infer the prompt and the generated code, NOIR introduces a new mechanism to achieve indistinguishability, a local differential privacy protection at the token embedding level, in the vocabulary used in the prompts and code, and a data-independent and randomized tokenizer on the client side. These components effectively defend against reconstruction and frequency analysis attacks by an honest-but-curious cloud. Extensive analysis and results using open-source LLMs show that NOIR significantly outperforms existing baselines on benchmarks, including the Evalplus (MBPP and HumanEval, Pass@1 of 76.7 and 77.4), and BigCodeBench (Pass@1 of 38.7, only a 1.77% drop from the original LLM) under strong privacy against attacks.

1 Introduction

Commercial LLM-powered code generation tools have greatly boosted developer productivity [1]. Yet, over 80% of companies using cloud-hosted generative AI cite intellectual property (IP) leakage and data security as major concerns, with nearly 45% reporting unintended data exposure [2–5], including real cases of proprietary code leakage [6]. These risks are rooted in the prompts a client sends to the cloud-hosted LLMs to get generated code. The prompts and the generated code allow the cloud operators to observe the exact functions and techniques behind commercial systems. Therefore, addressing these concerns and risks is critical given the proliferation of LLM-powered code generation across industry sectors [7].

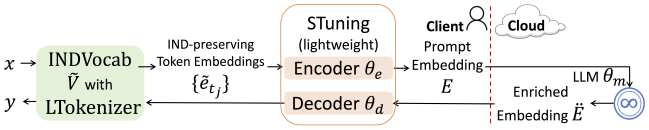


Figure 1: NOIR: Privacy-preserving Generation of Code.

Private Generation of Code. A vital solution is protecting both client prompts and generated code from cloud observation. Hosting proprietary LLMs on the client side is impractical, even in compressed form [8], as it risks model ownership loss. An alternative is client-side data centers to train, host, and maintain open-source LLMs for code generation, benefiting clients with safety-critical or sensitive data [9–12]. However, this remains financially unfeasible for most clients and enterprises [13, 14].

Prior approaches include prompt tuning [15–17] and example ensembling [18] under differential privacy (DP) to protect membership information of data points in the training set; that is, prevent the cloud from inferring whether a data point is used in training prompt-tuning models. Other methods to protect the client's prompts are to alter tokens [19–21] or their embeddings [22] in every client's prompt to achieve *metric* DP (i.e., d_x-privacy [23]). These state-of-the-art (SoTA) approaches focus on either classification tasks [19–22] instead of code generation tasks as pointed out in [22] or protecting the training data for prompt tuning [15–18] without protecting the content of client's prompts and generated outcomes.

Therefore, a novel approach is desired for the private generation of code, where the sweet-spot is balanced between rigorous privacy-preserving guarantees for prompts and generated code, cost effectiveness, and high model performance.

Challenge. Unlike generating general text, the main challenge in private code generation is that small changes in clients' prompts for privacy protection—at the token or its embedding level—can severely degrade generated code functionality, as LLMs are highly sensitive to such changes. Ensuring

good generated code functionality often weakens d_x-privacy protection in SoTA methods [19–22], leaving prompts and generated code vulnerable to reconstruction attacks [24, 25] by privacy-untrusted clouds. In addition, we prioritize code generation over general text due to its severe risks to IP and security. Leaks of proprietary algorithms, sensitive system code implementation, and trade secrets create a massive blind spot for zero-day vulnerabilities, IP theft, and supply chain compromises causing security, legal, and economic damages far greater than in general text generation [26].

Contributions. This paper proposes NOIR (Figure 1), the first framework to protect client prompts and generated code from cloud observation using an open-source LLM. Leveraging the competitive performance of open-source LLMs [27, 28], service providers can offer NOIR to clients and enterprises demanding strong privacy guarantees [9–12]. NOIR uses open-source LLMs to provide privacy guarantees to clients, and it complements closed-source LLM-based code generation tools, where it is difficult to provide privacy protection without exposing closed-source models to clients.

Given an open-source LLM, NOIR splits it into three parts: 1) **encoder** (first few attention blocks), 2) **middle part** (most attention blocks), and 3) **decoder** (last few attention blocks). On the client side, NOIR includes the encoder, decoder, a fine-tuning method (STUNING), and an indistinguishability (IND)-preserving vocabulary (INDVOCAB), which provides a local differential privacy (LDP) protection at the token embedding level, associated with a local randomized tokenizer (LTOKENIZER). Privacy protection arises from running the encoder and decoder locally: prompts are encoded before being sent to the cloud-hosted (middle part of the) LLM, which enriches prompt embeddings using its latent knowledge. The enriched embeddings enhance model performance on a range of downstream tasks because the middle part of LLMs can capture even richer representations from large datasets compared with the final layers of LLMs [29]. These embeddings are returned to the client's decoder to generate code. This design avoids sending raw prompts or code to the cloud.

To optimize the encoder and decoder to client tasks, we develop STUNING, a cost and performance-effective split learning approach [30–32], to fine-tune them locally with client datasets, mitigating utility loss from privacy protection. Since the encoder and decoder are lightweight compared with the cloud-hosted model, STUNING is cost-effective. Optionally, the cloud may fine-tune a low-rank adaptation (LoRA [33]) of the hosted model for improved performance.

To prevent the cloud from inferring sensitive and proprietary content from the prompts and the generated code via prompt embeddings and back-propagated gradients under SoTA reconstructing attacks [24, 25] in both the STUNING and inference phases, we propose a client-side privacy mechanism with two components: INDVOCAB and LTOKENIZER. INDVOCAB adaptively randomizes the token embeddings in the vocabulary making them indistinguishable to the cloud so the probability of the cloud inferring ground-truth tokens, prompts, and code given the prompt embeddings is upper-bounded, while minimizing injected randomness for better utility. To protect the one-hot vectors of tokens used in the client tokenizer, which the cloud can exploit during STUNING to reconstruct the client's data from the back-propagated gradients [25], NOIR develops LTOKENIZER, which uniformly assigns every token and its IND-preserving embedding to a random index in the INDVOCAB. This data-independent tokenizer remains secret (i.e., no extra privacy cost), misleading the cloud to observe meaningless tokens from the back-propagated gradients.

LTOKENIZER and INDVOCAB fully protect client prompts and output code during fine-tuning and inference. INDVOCAB keeps tokens in prompts, (prompt) instructions, (prompt) templates, and code intact while randomizing token embeddings only once with negligible noise. Thus, it maintains the correlation among prompts, instructions, templates, and code while providing indistinguishability protection to every token, given its token embedding. This correlation enhances STUNING by mitigating utility drops from encoder-(cloud-hosted) LLM-decoder misalignment when using the INDVOCAB and LTOKENIZER. Consequently, NOIR generates code with high functionality under strong client-side IND protection.

Extensive experiments with LLMs (CodeLlama-7B, CodeQwen1.5-7B-Chat, Llama3-8B-instruct) on benchmarks (Evalplus: MBPP [34], HumanEval [27]; BigCodeBench [28]) show that NOIR achieves a Pass@1 scores of 76.7 and 77.4 on MBPP [34] and HumanEval [27] and 38.7 on BigCodeBench (a marginal drop of 1.77% from the original LLM) while significantly outperforming SoTA baselines (T2T [19–21], SnD [22]) against reconstruction and frequency analysis attacks [24, 25]. Regarding cost effectiveness, NOIR reduces client inference/fine-tuning costs by $\backsim$ 10x compared with local LLM hosting, thanks to its lightweight encoder–decoder (1–4 attention blocks). GPU memory use, fine-tuning time, and equivalent AWS hosting costs grow sub-linearly with dataset size, making fine-tuning scalable on larger datasets without sharply increasing client communication costs.

To show practicality, we open-source NOIR (https://tinyurl.com/NOIR-Artifact) based on Qwen2.5-Coder-32B-Instruct and provide an application programming interface (API) via a privacy-preserving coding agent, accessible through a web service (https://noir.oppyai.com) and a Visual Studio (VS) extension (https://tinyurl.com/NOIR-Artifact), integrated into the development pipeline.

2 Related Work

Privacy-preserving Prompts. SnD [22] is the most recent related work. The client injects independent draws of Laplace noise into the token embeddings of every prompt to achieve d_x-privacy [23] (relaxed LDP) before sending the embeddings to the cloud. Then, it receives output embeddings in return

Table 1: A Summary of Differences between NOIR and Related Works.

	Generation	Denoise	Fine-tuning	A Token Embedding in Multiple Prompts	Tokens in a Prompt	Privacy Guarantee
TokEmbPriv [19]	×	×	×	changed, inconsistent	changed, inconsistent	d_x-privacy
Text2Text [20]	×	×	×	changed, inconsistent	changed, inconsistent	d_x-privacy
RAPT [21]	×	✓	✓	changed, inconsistent	changed, inconsistent	d_x-privacy
Split-and-Denoise [22]	×	✓	×	changed, inconsistent	changed, inconsistent	d_x-privacy
NOIR (ours)	✓	×	✓	unchanged, consistent	unchanged, consistent	ε-IND

and denoises them for downstream tasks. The common point among other studies [19–21] is achieving d_x-privacy by replacing tokens with randomized tokens, after injecting noise into token embeddings of a prompt.

Table 1 summarizes key differences between NOIR and prior methods. These methods were designed for classification, and extending them to sequence-to-sequence generation is difficult, as errors in previous token prediction will compound the deviation of the following tokens and substantially degrade performance [22]. We observe similar issues in our study. Therefore, instead of focusing on achieving d_x-privacy at the prompt level as in SoTA, NOIR enables private code generation—a harder task requiring protection of both prompts and generated code while preserving functionality by achieving indistinguishability of the vocabulary (INDVOCAB) and the LTOKENIZER. NOIR adaptively randomizes token embeddings only once, making them indistinguishable to the cloud, and keeping them consistent in INDVOCAB used across prompts and code. This reduces the randomness injected while maintaining tokens' essential correlation in the client's data; hence, NOIR allows the client to fine-tune the encoder and decoder locally for code generation.

In the scenario where users/organizations need to provide data to an enterprise (client) in NOIR's setting, they must trust the enterprise as a data curator in real-world deployments. This setting is different from classical LDP protection, in which the user/organization's data, i.e., prompts and generated code, is protected from the data curator [35], i.e., the client in NOIR. If the users/organizations have sufficient resources and data to host the encoder/decoder in practice, they can skip the enterprise (client) and directly use NOIR.

Privacy-preserving Prompt Tuning (P3T). The most recent P3T [15–17] is DP-OPT [17], which fine-tunes a local LLM for ensemble prompt engineering, with DP to protect the membership information of prompts in the local LLM's training set. **This objective is different from protecting the content of the client's prompts and generated code** in both fine-tuning and inference phases of NOIR.

3 Preliminaries

LLM-based Code Generation. An LLM uses a vocabulary of tokens (human-readable words) and their token embeddings: $V = \{t, e_t\}$, where $|V|$ is the vocabulary size and $e_t \in \mathbb{R}^m$ with m features. A prompt x is a sequence of tokens from V: $x = \{t_j\}_{j=1}^{|x|}$ where $t_j \in V$ and $|x|$ is the number of tokens in x. Combined with a task instruction π in a template $\mathcal{T}$[1], this guides the LLM to generate output code y, modeled as $P^h[y|\mathcal{T}(x,\pi)]$, where the higher the temperature h (≥ 0), the more diverse the generated code. For simplicity, we fix π and $\mathcal{T}$, treating x as the input. The prompt x is then represented as a sequence of token embeddings $\{e_{t_j}\}_{j=1}^{|x|}$ fed to the LLM: $P^h[y|\{e_{t_j}\}_{j=1}^{|x|}]$.

Tokenization consists of the first and the last steps of text processing and modeling in LLMs [36]. A tokenizer breaks down text into tokens and assigns each token $t \in V$ a unique numerical index represented by a one-hot vector $v_t \in \mathbb{I}^{|V|}$ without affecting the generality and correctness. Given the input $\{e_{t_j}\}_{j=1}^{|x|}$, the LLM iteratively generates the one-hot vector v_t of the next token t in the output code y. Finally, the tokenizer detokenizes the one-hot vectors $\{v_t\}_{t\in y}$ back to human-readable text by mapping those vectors to their corresponding tokens using the vocabulary V.

Differential Privacy. DP [37] is widely-used for data privacy, and LDP [38, 39] particularly protects the values of data inputs against an untrusted data curator. In the classical LDP definition, an LDP-preserving mechanism produces similar output distributions, preventing the curator from distinguishing the outcomes of the data inputs.

Definition 1. *ε-LDP [39]. A randomized algorithm $\mathcal{M}$ fulfills ε-LDP, if for any two inputs x and x', and for all possible outputs O of $\mathcal{M}$ $\big(O \in Range(\mathcal{M})\big)$, we have: $Pr[\mathcal{M}(x) = O] \leq e^{\varepsilon} Pr[\mathcal{M}(x') = O]$, where ε is a privacy budget.*

A smaller ε enforces a stronger privacy guarantee controlling the difference of the distributions induced by x and x'. Our setting is different from the classical LDP, where users/organizations with limited resources must trust the client to directly use NOIR (Section 2).

Split Learning [30] enables distributed learning by decomposing a neural network, e.g., an LLM in NOIR, into non-overlapping client and cloud segments. In NOIR, we consider a client (e.g., a small enterprise with limited resources) and a cloud, without loss of generality. Typical split learning frameworks use two architectures [31, 32]; NOIR adopts the setting that protects both the input prompt x and output code y: the client holds the first and last several attention blocks, while the cloud hosts the middle blocks of the LLM.

[1]For instance, π: "You are an expert in Python:" and $\mathcal{T}$: "Complete the request. ### Instruction: {instruction π} {prompt x} ### Response:"

Algorithm 1 NOIR: Private Generation of Code

Input: Vocabulary V, Encoder θ_e, Decoder θ_d, Cloud-hosted Model θ_m, IND budget ε, Training Data $D = \{x, y\}$, learning hyper-parameter γ
Output: ε-IND-preserving $\tilde{V}$, LTOKENIZER, θ_e, θ_d
Def Client$(V, \theta_e, \theta_d, D)$
  Initialize $\tilde{V}$ as a copy of the vocabulary V # Creating INDVOCAB
  for token $t \in V$ **do**
    for i^{th}-feature of token embedding e_t of t **do**
      Compute β_i using Theorem 1
      Assign a value to $\tilde{e}_t^i$ with ARR using Eq. 2 with β_i
    Assign $\{t, \tilde{e}_t\}$ a random, unique index in the client's tokenizer # LTOKENIZER
  for round $\in [1, T]$ # STUNING with LTOKENIZER, INDVOCAB **do**
    Sampling a batch B of data points $\{x, y\} \in D$
    $\{\mathcal{E} \leftarrow Enc(\mathcal{T}(x, \pi), \theta_e)\}_{x \in B}$ # Get prompt embeddings using $\tilde{V}$
    $\ddot{\mathcal{E}} \leftarrow$ **Cloud**$(\{\mathcal{E}\})$ # Send $\{\mathcal{E}\}$ to the cloud and get $\ddot{\mathcal{E}}$
    $\theta_d \leftarrow \theta_d - \lambda \nabla \mathcal{L}(\theta_d)$ # The client fine-tunes θ_d using $\ddot{\mathcal{E}}$ and y
    $\theta_m^{LoRA} \leftarrow \theta_m^{LoRA} - \lambda \nabla \mathcal{L}(\theta_m^{LoRA})$ # The cloud fine-tunes θ_m's LoRA
    $\theta_e \leftarrow \theta_e - \lambda \nabla \mathcal{L}(\theta_e)$ # The client fine-tunes θ_e using $\mathcal{L}$
Def Cloud$(\{\mathcal{E}\})$
  Return $\{f_{emb}(\mathcal{E}, \theta_m)\}$ # Extract and return enriched embeddings

Reconstruction Attacks (RAs). RAs aim to recover input text from its embedding [40–43] or both input and output text from their associated back-propagated gradient [25]. Vec2Text [24], the most advanced embedding-based RA, has base and refining steps: a trained conditional language model converts the embedding to a text corpus (the "base hypothesis"), then recursively re-embeds and corrects the base hypothesis to increase cosine similarity with the original text embedding. The Vec2Text model is trained on this generated data. BiSR [25], the latest gradient-based RA, initializes a dummy label and iteratively improves it through backward gradient matching and forward embedding matching.

4 NOIR: Overview, Feasibility, and Threats

Overview. IP leakage and data security motivate NOIR, designed to enforce **CPC** (data Confidentiality, information Privacy, and code Confidentiality) constraints. **Raw Data Confidentiality:** Client data—built from open/private sources, prompts, or generated code during inference—must not be shared with the (privacy-untrusted) cloud. **Encoded-Information Privacy:** Sensitive content in client prompts and code, across training and inference, must remain uninterpretable to the cloud. **Code Confidentiality:** Code syntax, semantics, and functionality in client data and generated outputs must not be reconstructable by the cloud.

General Design. To accommodate the CPC constraints in NOIR (Figure 1, Alg. 1), the cloud decomposes an open-source LLM θ into three parts: an encoder with the first few attention blocks θ_e, a decoder with the last few attention blocks θ_d, and the remaining middle blocks θ_m. The lightweight encoder and the decoder associated with the vocabulary V and its tokenizer are known to the client so that the client can use them at a negligible cost. Meanwhile, the cloud hosts the remaining blocks θ_m. The pair of an encoder and a decoder satisfies the data confidentiality constraint. By integrating INDVOCAB and LTOKENIZER, the design satisfies information privacy and code confidentiality constraints.

Feasibility. Our setting is feasible because: **(1)** Most LLM-providers (e.g., OpenAI, Meta, Google, DeepSeek, Alibaba, etc.) release open-source versions of their closed-source models. Service-providers thus have an incentive to offer open-source encoders and decoders to clients concerned with IP leakage and data security [2–5, 9–12], attracting new customers without affecting existing closed-source offerings; **(2)** Operation costs—hosting, fine-tuning, and inference—are notably reduced with lightweight encoders and decoders, making deployment practical for most clients and enterprises; and **(3)** Open-source LLMs now achieve highly competitive performance [27, 28]. Hence, open-source LLMs align well with the design of NOIR.

The client uses the open-source vocabulary V and the tokenizer to represent every prompt x as a sequence of token embeddings $\{e_{t_j}\}_{j=1}^{|x|}$. The encoder θ_e extracts a prompt embedding $\mathcal{E} = Enc(\{e_{t_j}\}_{j=1}^{|x|}, \theta_e)$, which is sent to the cloud-hosted model θ_m to obtain an enriched embedding $\ddot{\mathcal{E}} = f_{emb}(\mathcal{E}, \theta_m)$. The client feeds $\ddot{\mathcal{E}}$ to the decoder θ_d to generate the output code $y = Dec(\ddot{\mathcal{E}}, \theta_d)$. The encoder and decoder are fine-tuned (STUNING, Section 7) for local tasks using curated data from open and/or the client's private sources. The cloud can either **1) keep θ_m fixed** to reduce computation complexity or **2) fine-tune a lightweight LoRA** [33] of the middle block θ_m with the client for improved utility at marginal cost.

4.1 Threat Model

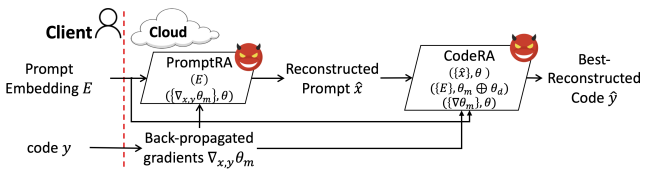


Figure 2: Threat Model of an Honest-but-Curious Cloud.

In a **defense-free environment**, the client uses the open-source vocabulary and tokenizer during fine-tuning and inference. Figure 2 shows the attack surface in NOIR. The honest-but-curious cloud seeks to reconstruct prompts, code used in the client's fine-tuning, and code generated in the inference phase. To reconstruct the prompts (PromptRA), the cloud applies SoTA RAs [24] on the client's prompt embeddings, then feeds them to the LLM model θ to generate reconstructed code $\hat{y}$ (CodeRA). Due to noisy prompt embeddings, the reconstructed prompts are typically imperfect, so the cloud alternatively feeds prompt embeddings directly into θ_m concatenated with the original decoder θ_d. During fine-tuning, the cloud can also feed the back-propagated gradient of each training sample $\{x, y\}$ through the middle block θ_m, denoted as $\nabla_{x,y}\theta_m$, into BiSR [25] to reconstruct the prompts

x and their output code y in the client's training set D. We evaluate attack performance using the best-reconstructed code in both methods, considering maximal information leak in each metric, including information privacy, code confidentiality, and code functionality.

We exclude attacks involving compromised employees in the client's organization. These employees can collude with the cloud for the cloud to query the client's encoder or disclose the input prompts x, the output code y, the INDVOCAB, the LTOKENIZER, the fine-tuned encoder and decoder, and their associated gradients to the cloud. Such insiders can gather all raw data during training and inference (out of NOIR's scope). **Token sequence-based frequency analysis attacks**, e.g., the codebook attack [44], **do not apply to NOIR** since: **(1)** the encoder's attention mechanism yields varied token embeddings for the same input token given different token positions; and **(2)** The cloud cannot query the client's encoder or access raw inputs. Therefore, **adaptive attacks** (manipulating prompts to observe output changes) **are not applicable**.

4.2 NOIR's Defense

To protect the client's prompts and code against RAs, NOIR has two key components. First, INDVOCAB replaces the vocabulary V with an IND-preserving $\tilde{V} = \{t, \tilde{e}_t\}$, where every $\tilde{e}_t$ is an IND-preserving token embedding, derived from the original token embedding e_t (Section 6). Hence, a prompt x is represented by a sequence of IND-preserving token embeddings $\{\tilde{e}_{t_j}\}_{j=1}^{|x|}$ defending against PromptRA. Second, NOIR develops a local tokenizer (LTOKENIZER), which uniformly assigns every token and its embedding to a random index in the tokenizer, on the client side to defend against CodeRA. The client keeps this data-independently and randomized tokenizer (no extra privacy cost) secret from the cloud, misleading the cloud's attacks to reconstruct meaningless tokens in the output code y of the client's training data D.

At the inference phase, the client uses the INDVOCAB $\tilde{V}$ associated with the LTOKENIZER, the fine-tuned encoder θ_e^*, and the decoder θ_d^* to generate code for its prompts. **The client does not share the fine-tuned θ_e^* and θ_d^* with the cloud, and the cloud does not share the LoRA of the middle block θ_m during the fine-tuning and inference phases** to maintain their model ownership. NOIR maintains the CPC constraints, incentivizing the client to use LLM-based code generation under IP and data security protection and broadening the adoption of the cloud's service.

Example. Given the prompt in Figure 3a, the cloud reconstructs the client's generated code with a clear gist in the defense-free environment (Figure 3c). On the contrary, the cloud reconstructs meaningless code under INDVOCAB and LTOKENIZER (Figure 3d); meanwhile, the client enjoys its desired code with NOIR on their side privately (Figure 3b). Next, we describe reconstruction attacks as security games between the client and the cloud.

5 Reconstruction Attacks

5.1 Prompt Reconstruction Attack

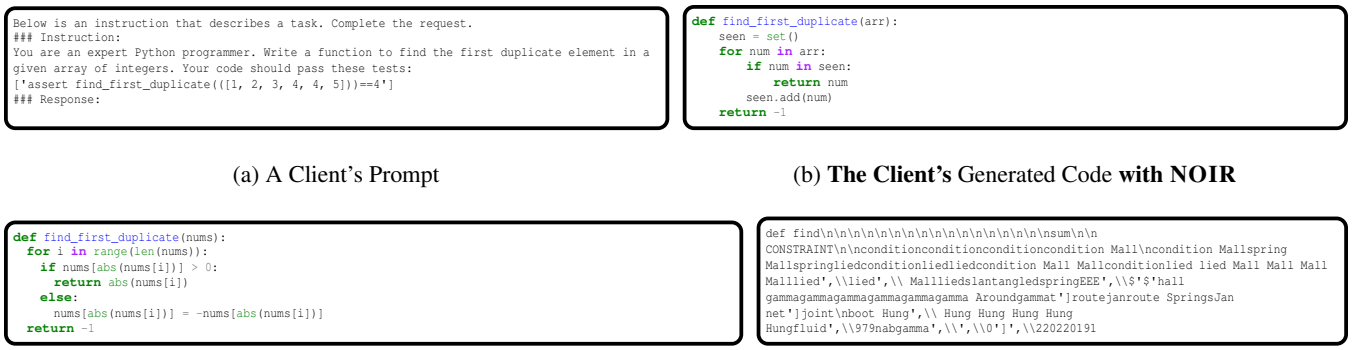
In PromptRA, the cloud feeds each prompt embedding $\mathcal{E}$ into Vec2Text [24] or feeds the back-propagated gradients $\nabla_{x,y}\theta_m$ to BiSR [25], the SoTA RAs, to generate $\hat{x}$ approximating the ground-truth prompt x, formulated as $PromptRA: (\{\mathcal{E}\}, \theta_a) \cup (\{\nabla_{x,y}\theta_m\}, \theta) \rightarrow \hat{x}$, where θ_a are Vec2Text's pre-trained parameters. The closer $\hat{x}$ is to x, the stronger the attack and the greater the information privacy leakage. In the security game, the cloud may request the client's prompt embeddings $\mathcal{E}$ and their associated back-propagated gradients $\nabla_{x,y}\theta_m$ from the training set D at any time during local fine-tuning. We denote the sets of all prompt embeddings as $\mathbb{E}$ and their reconstructed prompts as $\hat{X}$.

We evaluate attack success using the well-known token-level metrics Bleu [45] and Rouge [46] scores, which measure similarity between the reconstructed prompt $\hat{x}$ and the original x. The cloud wins the game for a prompt $x \in D$ if it returns $\hat{x} \in \hat{X}$ with a clear gist: either $\max\{Bleu(\hat{x}, x)\}_{\hat{x}\in\hat{X}} \geq \rho_b$, where $\rho_b = 20$ [47], or $\max\{Rouge(\hat{x}, x)\}_{\hat{x}\in\hat{X}} \geq \rho_r$, where $\rho_r = 0.4$ [48]. The overall attack success rate (ASR) is the cloud's average winning rate over all prompts in D during the client's fine-tuning. During inference, the cloud requests all prompt embeddings in the test set D_{test} once, since multiple requests make no difference. By default, we use uni-gram Bleu and Rouge, which yield the best results compared with bi- and longer-grams; otherwise, the specific n-grams is noted.

This study uses the ASR thresholds ($\rho_b = 20$, $\rho_r = 0.4$) to evaluate the effectiveness of our model against RAs and baselines. In real-world deployments, the client can ignore these thresholds, since the client will receive a security report showing the cloud's reconstructed prompts and outcomes for each prompt. Hence, it can determine whether sensitive information has been leaked. The thresholds balance practical privacy protection with realistic evaluation of reconstruction risks. The cloud's ability to infer exploitable information from encoded embeddings or gradients is quantitatively limited, whereas distinguishing between useful reconstructions and noise or meaningless outputs (lower threshold values) poses little risk. Overly strict thresholds can lead to a high false-positive rate. NOIR's thresholds avoid false positives by considering exploitable reconstructions as the only alarming threat, which is more realistic and actionable for clients.

5.2 Code Reconstruction Attack

In CodeRA, the cloud 1) feeds the prompt embeddings $\{\mathcal{E}\}$ to the cloud-hosted model θ_m concatenated with the original decoder θ_d, denoted as $\theta_m \oplus \theta_d$, or 2) feeds the reconstructed prompts $\{\hat{x}\}$ to the LLM θ, or 3) feeds the back-propagated gradients $\nabla_{x,y}\theta_m$ to BiSR [25] to generates the output code $\hat{y}$ approximating the ground-truth code y, i.e., CodeRA: $(\{\mathcal{E}\}, \theta_m \oplus \theta_d) \cup (\{\hat{x}\}, \theta) \cup (\{\nabla_{x,y}\theta_m\}, \theta) \rightarrow \hat{y}$. The



(a) A Client's Prompt

(b) **The Client's** Generated Code **with NOIR**

(c) **The Cloud's** Reconstructed Code **in a Defense-free Environment**

(d) **The Cloud's** Reconstructed Code **under NOIR**

Figure 3: An Example of the Threat Model.

Table 2: PromptRA and CodeRA in a Defense-free Environment.

PromptRA	Training Data D				Testing Data D_{test}			
	Bleu	Rouge	$ASR^{priv}_{x,D}$	CRT_x	Bleu	Rouge	$ASR^{priv}_{x,D_{test}}$	CRT_x
MBPP (no test cases in prompts, no fine-tuning)	34.17	0.66	**0.96**	**0.32**	35.53	0.67	**0.96**	**0.32**
MBPP (no test cases in prompts, fine-tuning)	33.92	0.66	**0.96**	**0.32**	34.08	0.66	**0.95**	**0.32**
MBPP (no fine-tuning)	11.25	0.40	**0.50**	**0.1**	11.49	0.39	**0.54**	**0.1**
MBPP (fine-tuning)	11.31	0.40	**0.51**	**0.1**	11.64	0.39	**0.51**	**0.1**
BigCodeBench	N/A	N/A	N/A	N/A	16.65	0.47	**0.32**	0.284

CodeRA	Training Data D								Testing Data D_{test}							
	Bleu	Rouge	CodeBleu	Fusi	$ASR^{priv}_{x,D}$	$ASR^{prop}_{x,D}$	$ASR^{fusi}_{x,D}$	CRT_y	Bleu	Rouge	CodeBleu	Fusi	$ASR^{priv}_{x,D_{test}}$	$ASR^{prop}_{x,D_{test}}$	$ASR^{fusi}_{x,D_{test}}$	CRT_y
MBPP (no fine-tuning)	74.32	0.86	67.87	0.46	**0.99**	**0.98**	**0.47**	**1.0**	73.00	0.84	66.42	0.44	**0.99**	**0.97**	**0.46**	**1.0**
MBPP (fine-tuning)	72.41	0.85	64.42	0.45	**0.99**	**0.97**	**0.46**	**1.0**	71.11	0.83	63.17	0.43	**0.99**	**0.96**	**0.44**	**1.0**
BigCodeBench	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	50.67	0.68	56.90	0.34	**0.99**	**0.99**	**0.35**	**1.0**

closer $\hat{y}$ is to y, the stronger the attack and the higher the risk are. This similarity is quantified by: **(1)** Information Privacy: token-level metrics (Bleu and Rouge) to measure similarity between the reconstructed and the ground-truth code ($\hat{y}$ and y); **(2)** Code Confidentiality: CodeBleu [49] for code syntax and semantic similarity between $\hat{y}$ and y; and **(3)** Code Functionality: similarity in unit tests passed by $\hat{y}$ and y, defined as follows:

$$Fusi(\hat{y},y) = \frac{\sum_{u\in U_x} \mathbb{I}[pass(\hat{y},u)=1 \;\&\; pass(y,u)=1]}{\sum_{u\in U_x} \mathbb{I}[pass(y,u)=1]}, \quad (1)$$

where U_x is a set of unit tests u to access the functionality of the generated code, given a prompt x. $pass(y,u)=1$ if the code y passes the unit test u; otherwise, $pass(y,u)=0$. $\mathbb{I}[\cdot]=1$ if $pass(\hat{y},u)=pass(y,u)=1$; otherwise $\mathbb{I}[\cdot]=0$.

The cloud wins the game for a given code y in the training set D if it returns $\hat{y}$ with a clear gist of information privacy, code confidentiality, and code functionality. Fusi is stricter than Bleu, Rouge, and CodeBleu since passing unit tests demands full semantic and syntax correctness. A single token error can fail unit tests while barely affecting Bleu, Rouge, and CodeBleu scores. Hence, Fusi yields lower values than Bleu, Rouge, and CodeBleu, and lower ASRs for code functionality than for privacy and confidentiality. Passing even one similar unit test with the ground-truth code signals code functionality leak. Therefore, the cloud wins the functionality security game if $Fusi(\hat{y},y) > \rho_f = 0$ (a leak vs. no leak).

The cloud wins the game for code y in terms of i) information privacy if $Bleu(\hat{y},y) \geq \rho_b\ (=20)$ [47] or $Rouge(\hat{y},y) \geq \rho_r\ (=0.4)$ [48], ii) code confidentiality if $CodeBleu(\hat{y},y) \geq \rho_b\ (=20)$, and iii) code functionality if $Fusi(\hat{y},y) > \rho_f\ (=0)$. The ASR is the cloud's average winning rate over all the code in D. During inference, the cloud attacks with one request for all prompt embeddings in the client's test set.

5.3 Token-level Privacy Metrics

Alongside Bleu, Rouge, CodeBleu, and Fusi metrics, we report correctly reconstructed tokens (CRT) in prompt x and code y defined as: $\boldsymbol{CRT_x} = \frac{1}{|D|}\sum_{x\in D}\frac{|x\cap\hat{x}|}{|x|}$ and $\boldsymbol{CRT_y} = \frac{1}{|D|}\sum_{y\in D}\frac{|y\cap\hat{y}|}{|y|}$. We also assess the cloud ability to reconstruct sensitive information (imported packages, variable names, and function names) via $\boldsymbol{\texttt{leak}} = \frac{1}{|D|}\sum_{y\in D}\boldsymbol{\texttt{leak}(y)}$, where $\texttt{leak}(y)=1$ if such sensitive information appears in the reconstructed code, and $\texttt{leak}(y)=0$ otherwise. Together, these metrics comprehensively capture privacy leakage at both structural and token levels in PromptRA and CodeRA.

5.4 Initial Assessment of PromptRA, CodeRA

Our initial focus is to assess RAs through an experiment in a defense-free environment using benchmark CodeAlpaca [50]

and MBPP [34] datasets from Evalplus, both designed for Python code generation. We use CodeLlama-7B as the LLM θ, with the first attention block as the encoder θ_e, the last four attention blocks as the decoder θ_d, and the remaining middle blocks forming the cloud-hosted model θ_m. CodeAlpaca contains ∼18k data points, which is larger than MBPP, which has 974 data points. The cloud initializes the BaseModel and Vec2Text from a T5-base checkpoint [51] as in [24], and trains PromptRA on the (larger) CodeAlpaca as public data (batch size: 24, the max sequence length: 768 tokens, since the original texts in CodeAlpaca are long). While, the client fine-tunes its encoder and decoder on the (smaller) MBPP.

Without fine-tuning and without test cases in client prompts, PromptRA achieves high Bleu scores (34.17 and 35.53 for training and test sets respectively, Table 2) and strong Rouge scores (0.66 and 0.67). Consequently, the privacy attack success rates $ASR^{priv}_{x,D}$ and $ASR^{priv}_{x,D_{test}}$ reach 96%, showing that the cloud can reconstruct meaningful content in most client prompts. When test cases are included in client prompts[2], Rouge scores drops: 0.4 and 0.39; resulting in lower ASRs: 50% and 54%, for training and test sets respectively. This is because Vec2Text [24] and BiSR [25] are not suited for long prompts with code and tests. Although PromptRA weakens, CodeRA exels with Bleu 74.32, Rouge 0.86, CodeBleu 67.87, and Fusi 0.46, when prompts have test cases, yielding high ASRs: over 97% privacy, 98.25% code confidentiality, and 46.5% code functionality leakage across training and test sets.

The cloud still registers **high ASRs** in PromptRA and CodeRA **when the client fine-tunes and conceals its encoder and decoder**. Privacy ASRs reach 51% for prompts, 95.5% for prompts without test cases, and 99% for code. Code confidentiality ASR is 97%, and code functionality ASR is 45.75% across training and test sets on average. Similar results appear on HumanEval [27], BigCodeBench [28], and LLMs such as CodeQwen1.5-7B-Chat, Llama3-8B-instruct. We observe token-level metrics $CRT_x = 0.28$ and $CRT_y = 1.0$ on MBPP. The same attack performance is registered on the BigCodeBench, an advanced evaluation of LLMs in programming with 1,140 data points. The `leak` scores of sensitive information are high (0.98) on both MBPP and BigCodeBench.

Remarks. A cloud can reconstruct the content of prompts and code with high Bleu, Rouge, CodeBleu, Fusi, CRT, and `leak` scores, leading to high ASRs. Even one high ASR poses significant IP and data security risks for the client. Thus, effective privacy-preserving mechanisms are essential to protect prompts and code while sustaining model performance.

6 IND-preserving Vocabulary

RAs infer tokens in the prompt x from its embedding $\mathcal{E}$ to reconstruct prompts and code. At the token level, $x = \{t_j\}_{j=1}^{|x|}$ is represented as a sequence of token embeddings $\{e_{t_j}\}_{j=1}^{|x|}$, where $t_j \in V$. The closer reconstructed token embeddings are to the ground-truth ones in x, the more accurately the cloud can infer every token t_j, and thus reconstruct x and its code y. To provide privacy guarantees, we consider the worst-case attack where the cloud losslessly reconstructs the ground-truth token embeddings: $\forall t_j \in x : \hat{e}_{t_j} = e_{t_j}$, with $\hat{e}_{t_j}$ the reconstructed token embedding of the token t_j.

To defend against this worst-case, the client can randomize tokens in the prompt x and the code y for local fine-tuning, resulting in replacing tokens with other tokens to preserve d_x-privacy [23]. The client can then either denoise the cloud-hosted LLM output [22] or fine-tune the decoder for private code generation as in NOIR. Traditionally, d_x-privacy at the token level aims to prevent a data curator from authorship inference [52], not reconstruction attacks as in our study. This mismatch leads to the following fundamental challenges.

Challenges. A subtle token-level perturbation in the prompt x can drastically alter its content (syntax, functionality, function names, variables, etc.), making LLMs generate irrelevant or faulty code. When a token t appears in multiple prompts, perturbing its token embedding in these prompts with different draws of random noise to achieve d_x-privacy amplifies randomness, as t is represented by inconsistent randomized token embeddings, degrading model performance. Similarly, the randomness is applied to the instruction π and the template $\mathcal{T}$ weakening LLM performance, since poor and noisy instructions further reduce the output quality. Thus, balancing strong performance in code generation with LDP against PromptRA and CodeRA remains challenging.

6.1 INDVOCAB

To address the challenge, we propose a novel concept of indistinguishability-preserving vocabulary (INDVOCAB), which is a LDP protection at the token embedding level, as follows:

Definition 2. ε-INDVOCAB. *A randomized algorithm $\mathcal{M}$ fulfills ε-INDVocab, if for any two tokens t and t' in the vocabulary V, and for all possible outputs O of $\mathcal{M}$, i.e., $O \in Range(\mathcal{M})$, we have: $Pr[\mathcal{M}(e_t) = O] \leq e^{\varepsilon} Pr[\mathcal{M}(e_{t'}) = O]$.*

ε-INDVOCAB ensures that the cloud cannot distinguish the outcomes $\mathcal{M}(e_t)$ of the original token embeddings e_t under ε-indistinguishability protection. Instead of using the original token embeddings $\{e_t\}_{t\in V}$ in the vocabulary V, the client applies algorithm $\mathcal{M}$ to randomize these token embeddings resulting in IND-preserving token embeddings $\{\tilde{e}_t = \mathcal{M}(e_t)\}_{t\in V}$ in a new INDVOCAB $\tilde{V}$. By locally replacing V with $\tilde{V}$ to map a token t to an embedding $\tilde{e}_t$, the client prevents the cloud from inferring the ground truth tokens. This is because the cloud can only reconstruct the randomized token embeddings $\{\tilde{e}_t\}_{t\in V}$ instead of the ground-truth token embeddings. Thus, the tokens and the prompt x are protected against PromptRA. Using the INDVOCAB $\tilde{V}$ to derive the

[2]For instance, given the prompt "Write a Python function to find the remainder of two numbers," a test case is 'assert find(3,3)==0.'

prompt embedding by feeding the prompt x to the encoder θ_e results in an IND-preserving prompt embedding $\tilde{e}$ following the post-processing property in DP [37]. The IND-preserving tokens, prompts, and prompt embeddings $\tilde{e}$ further prevent the cloud from reconstructing the code y with CodeRA.

Achieving INDVOCAB with Adaptive Randomized Response (ARR). We propose an ARR mechanism as the algorithm $\mathcal{M}$ in Def. 2 to preserve INDVOCAB while maintaining high model utility with key advantages to overcome the challenges and limitations related to token-level LDP [52] in code generation. The pseudo-code of our ARR mechanism is in Alg. 1, Lines 4-8. Let us denote the i^{th} feature in a token embedding e_t as e_t^i. The ARR's key idea is flipping a probabilistic coin whether we keep the original value of the feature e_t^i or change it to another possible feature value among other tokens $\{e_k^i\}_{k\in V\setminus t}$, where k is a token different from t in the vocabulary V, such that *"more similar feature values to e_t^i have higher probabilities to be selected as a replacement."* We formulate the idea as follows:

$$\text{Given an arbitrary token } t \in V, \forall i^{\text{th}}\text{-feature of } e_t: \quad (2)$$

$$\tilde{e}_t^i = \begin{cases} e_t^i, \text{ with probability } p_i = \exp(\beta_i)/[\exp(\beta_i)+|V|-1], \\ e_k^i, \text{ with probability } q_{i,k} = (|V|-1)q_k/[\exp(\beta_i)+|V|-1], \end{cases}$$

where $k \in V \setminus t$, $q_k = \exp(-\Delta_{t,k}^i/m)/\sum_{l\in V}\exp(-\Delta_{t,l}^i/m)$, $\Delta_{t,k}^i = |e_t^i - e_k^i|$, m is the number of features in a token embedding, and $p_i + \sum_{k\in V\setminus t} q_{i,k} = 1$. The larger q_k indicates that e_k^i is more similar to e_t^i. Theorem 1 bounds β_i s.t. $\forall k \in V\setminus t : p_i \geq q_{i,k}$ preserving ε_i-IND as in typical RR definition [53].

Theorem 1. *Randomizing the i^{th}-feature in a token embedding e_t preserves ε_i-IND if ε_i and β_i are bounded as*

$$\forall \varepsilon_i \geq \frac{1}{m}(\Delta_{t,max}^i - \Delta_{t,min}^i) : \ln(|V|-1) + \ln\Big(\frac{\max\{\exp(-\Delta_{t,k}^i/m)\}_{k\in V\setminus t}}{\sum_{l\in V}\exp(-\Delta_{t,l}^i/m)}\Big)$$

$$\leq \beta_i \leq \varepsilon_i + \ln(|V|-1) + \ln\Big(\frac{\min\{\exp(-\Delta_{t,k}^i/m)\}_{k\in V\setminus t}}{\sum_{j\in V}\exp(-\Delta_{t,j}^i/m)}\Big), \quad (3)$$

where $\Delta_{t,min}^i = \min\{\Delta_{t,k}^i\}_{k\in V\setminus t}$, $\Delta_{t,max}^i = \max\{\Delta_{t,k}^i\}_{k\in V\setminus t}$.

The proofs of Theorem 1 and the following theorems are in the Appx. In practice, we use the upper-bound of β_i in Theorem 1, since larger β_i yields less noisy randomization probabilities. We apply ARR to independently randomize every i^{th}-feature in a token embedding e_t with a privacy budget ε_i. Thus, we create an ε-IND-preserving token embedding $\tilde{e}_t$ where the total privacy budget is $\varepsilon = \sum_{i\in e_t}\varepsilon_i$ following the composition theorem in DP [37] reflected in Theorem 2 (Appx. B). The total IND budget ε is lower-bounded as follows: $\varepsilon = \sum_{i\in e_t}\varepsilon_i \geq \frac{1}{m}\sum_{i\in e_t}\Delta_{t,max}^i$ (Eq. 3), which is tiny ($\lesssim 1e-3$) in almost all LLMs, enabling us to work with a high privacy protection (if needed) in practice.

Post-processing Property. The client applies INDVOCAB $\tilde{V}$ to represent every prompt x and the instruction π as a sequence of ε-IND-preserving token embeddings, e.g., the sequence of token embeddings given a prompt x is $\{\tilde{e}_{t_j}\}_{t_j=1}^{|x|}$. The number of times a token t appears in one or more prompts does not affect the ε-IND guarantee of the token embedding of t following the post-processing property of DP [37]; that is, no extra information regarding the ground-truth token's embedding is used afterward. The client freely uses the INDVOCAB $\tilde{V}$ without affecting the IND protection.

6.2 Prompt-Level Protection

In this study, we safeguard prompts and code against reconstruction attacks from the cloud. Thus, we provide an **upper-bounded probability of reconstructing a given prompt** x in the security game described in Section 5.1, highlighting the link between the token-level INDVOCAB and prompt-level privacy under reconstruction risk. In this game, reconstructed and ground-truth prompts $\hat{x}$ and x have the same number of tokens: $|\hat{x}| = |x|$. Hence, $\textit{Rouge-F1}(\hat{x},x) = \textit{Bleu}(\hat{x},x) = C/|x|$, where C is the number of correctly reconstructed tokens in $\hat{x}$. Our analysis in Theorem 3 and Proposition 1 (Appx. C) upper-bounds the cloud's probability of recovering ground-truth tokens in x with gist level higher than or equal to ρ:

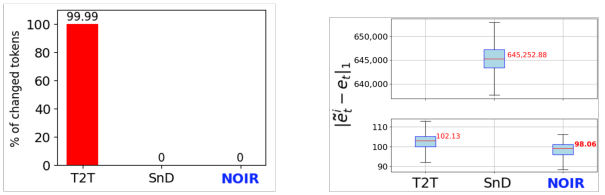
$$P\big[C/|x| \geq \rho; \{o_j\}_{j=1}^{|x|}\big] \leq \Big(\frac{\psi e^\varepsilon + 1}{\psi e^\varepsilon + \psi^2}\Big)^{\rho|x|} \times \Big(\frac{\psi e^\varepsilon}{\psi e^\varepsilon + 1}\Big)^{(1-\rho)|x|}, \quad (4)$$

where $\psi = |V| - 1$. We generalize Proposition 1 to consider the reconstruction attack with token sequences, capturing correlation among tokens in real-world scenarios, as follows: The cloud's previously reconstructed token sequences $\hat{t}_{<j}$ can enhance its probability of correctly reconstructing the next token t_j: $Pr(\hat{t}_j = t_j|\hat{t}_{<j})$. This advantage is bounded by a constant $\gamma \in [0,1]$ in practice, as follows: $\forall t_j \in x : 0 \leq Pr(\hat{t}_j = t_j|\hat{t}_{<j}) - Pr(\hat{t}_j = t_j) \leq \gamma$. Given γ, Theorem 4 (Appx. C) tightens the upper-bounded reconstruction risk, as follows:

$$P\big[C/|x| \geq \rho; \{o_j\}_{j=1}^{|x|}\big] \leq \Big(\frac{\psi e^\varepsilon + 1}{\psi e^\varepsilon + \psi^2} + \gamma\Big)^{\rho|x|} \times \Big(\frac{\psi e^\varepsilon}{\psi e^\varepsilon + 1} - \gamma\Big)^{(1-\rho)|x|}. \quad (5)$$

Meaningful Level of Privacy Protection. We analyze this upper-bounded reconstruction risk as a function of ε, prompt size $|x|$, vocabulary size $|V|$, and reconstruction threshold ρ, showing the effectiveness of INDVOCAB in protecting the prompt x. With $\varepsilon = 13$, the (clear gist) upper-bounded reconstruction risk for $|x| = 200$ is $< 5.5 \times 10^{-11}$ (Eq. 5) for modern LLMs (151k+ tokens in their vocabulary) with the small upper-bounded constant[3] $\gamma \leq 0.146$ derived from Evalplus datasets, which shrinks the reconstruction probability (Eq. 4). The upper-bounded reconstruction risk is equivalent to guessing an 8-character lowercase password ($1/26^8$). Unlike password attacks (repeated guesses), NOIR allows only one guess per prompt, as clients update their inputs frequently, making brute-force success impractical. Even with constant submissions of the same prompt every second, success would take an estimated $\frac{1}{2} \times \frac{26^8}{31{,}557{,}600} = 3{,}308.7$ years (one guess/s), in which $31{,}557{,}600$ is the number of seconds/year. Smaller ε (e.g., 9) exponentially enhance security,

[3] We compute the upper-bounded γ with $Pr(\hat{t}_j = t_j|\hat{t}_{<j})$ for every prompt in the dataset D: $\boldsymbol{\gamma} = \max_{j\in[0,|x|-1]} \boldsymbol{\Sigma}_{x\in D}\mathbb{I}(\hat{\boldsymbol{t}}_j = \boldsymbol{t}_j)/|\boldsymbol{D}|$.



(a) (b)

Figure 4: (a) the percentage of tokens changes in a prompt, (b) the L_1-norm distance between original and IND-preserving token embeddings on the MBPP dataset, $\varepsilon = 13$.

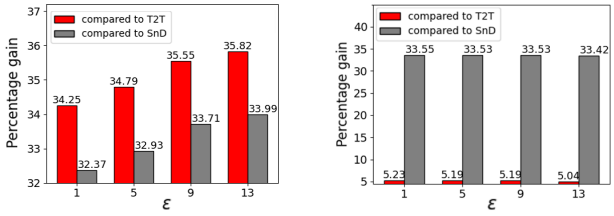
offering $\frac{1}{2} \times \frac{26^{72}}{31{,}557{,}600} = 2.4e^{94}$ years and impenetrable protection for shorter prompts (20 words) against brute-force attackers. Longer prompts have lower upper-bounded reconstruction probabilities. This ε range with small upper bounds for modern LLMs and typical prompts maintains high model performance while providing robust protection against RAs.

6.3 Advantages of INDVOCAB

INDVOCAB has fundamental advantages to achieving good model utility while protecting prompts and code.

Unchanged Tokens and Negligible Noise. If tokens in prompts x, instruction π, and template $\mathcal{T}$ are replaced by other tokens as in T2T [19–21], the code syntax breaks since an inappropriate tokens degrade functionality and distorts the essential correlation between a prompt x and its output code y. Fine-tuning a model on such distorted pairs yields models that generate code misaligned with prompt requirements. Even without changing tokens in prompts, the instruction, and the template, randomizing every token embedding with different draws of considerable noise as in SnD [22] still distorts the correlation among x, π, $\mathcal{T}$, and code y since multiple token embeddings can represent one token. As a result, the model performance is remarkably degenerated.

INDVOCAB preserves all the tokens in the prompts x, instruction π, and the template $\mathcal{T}$, randomzing tokens' embeddings once with negligible ε-IND noise. This property retains the essential correlations among x, π, and $\mathcal{T}$ in the client's local data, enabling fine-tuning of the encoder and decoder while balancing utility and privacy. We evaluate this with four experiments using the MBPP dataset: **(1)** Average percentage of changed tokens in prompts with test cases after IND preservation: $\frac{1}{|D|}\sum_{x\in D} \frac{\#\text{Changed Tokens in } x}{|x|}$; **(2)** Average L_1-norm distance between original and IND-preserved token embeddings using 32k tokens of the CodeLlama-7B model's vocabulary: $\{|\tilde{e}_t - e_t|_1\}_{t\in x, x\in D}$; **(3)** Change in (angle) cosine similarity between bi-gram (two sequential) tokens t_i and t_{i+1} in prompts x: $\{|cos(e_{t_i}, e_{t_{i+1}}) - cos(\tilde{e}_{t_i}, \tilde{e}_{t_{i+1}})|_1\}_{t_i\in x, x\in D}$, where $cos(\cdot)$ is a cosine similarity function; and **(4)** Change in (angle) cosine similarity between any token pairs in INDVOCAB given the original vocabulary: $\{|cos(e_t, e_{t'}) - cos(\tilde{e}_t, \tilde{e}_{t'})|_1\}_{t,t'\in V, t\neq t'}$. For a fair comparison, the privacy budget ηd_x in T2T and SnD (η is predefined) equals the ε in NOIR.



(a) Gain in bi-gram angular (b) Gain in token angular

Figure 5: NOIR's gain in terms of (a) bi-gram angular change in every prompt and (b) angular change among tokens in the vocabulary compared with T2T and SnD.

Figure 4a shows that T2T [19–21] alters nearly all tokens ($\approx 100\%$) in every prompt and test case, whereas SnD [22] and NOIR leave tokens unchanged. Despite this, SnD injects substantial noise into token embeddings, with an average L_1-norm distance of $\sim$645k versus 98.06 for NOIR (Figure 4b). The average L_1-norm distance in NOIR (102.13) is also significantly smaller than T2T, statistically significant: p-value $< 3.3e$-100 (2-tail t-test). Thus, only NOIR preserves tokens while injecting negligible ε-IND noise into their embeddings.

INDVOCAB obtains a significantly smaller average bi-gram angular change than T2T and SnD across IND budgets ε, with over 33% improvement (Figure 5a). Larger ε values further enhance INDVOCAB's ability to preserve bi-gram angles compared to SoTA methods, thereby better maintaining angular correlations under IND protection. Figure 5b also shows that NOIR achieves the smallest average angular change, confirming INDVOCAB's superior ability to preserve the relative angular correlation among tokens.

Adaptive Randomization. To assess adaptive randomization, we compare the ARR with uniform randomized response (**URR**), where every token embedding feature has the same randomization probability. In this experiment, we randomly pick one token t and a feature i^{th} in its token embedding. Figure 6 shows the correlation between the probability $q_{i,k}$ and the feature similarity q_k under ARR and URR. Features similar to e^i_t are more likely to be selected in the ARR than under URR's uniform probability. Those with similarity q_k and probabilities $q_{i,k}$ higher than the ones in URR are the most frequent (red rectangle). Thus, ARR preserves embedding utility better than URR under the same IND protection.

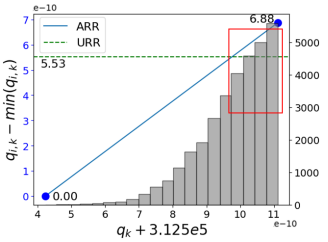


Figure 6: Adaptive probabilities.

Agnostic to Embedding Size. We evaluate various existing LDP mechanisms, from classical ones such as Gaussian [37], Laplace [37], and Duchi's mechanisms [54, 55], to advanced methods including Piecewise [56], Hybrid [56], Three-outputs

[57], Adaptive OME [58], and XRand [59]. All face two key challenges in our context: **(1)** High sensitivity Δ introduces excessive noise due to large embedding magnitudes; **(2)** Privacy budget accumulation over large embedding size (e.g., $m = 4,096$) renders generation quality unusable—outputs resemble code at $\varepsilon \approx 500,000$. Even relaxed d_x-privacy-based approaches [19–22] yield code-like outputs at $\varepsilon \approx 50,000$. Our ARR mechanism resolves these issues by advancing randomization probabilities $q_{i,k}$ and p_i using $-\Delta^i_{t,k}/m$ (Eq. 2), injecting negligible noise as demonstrated above. The total privacy budget becomes agnostic to embedding size m: $\varepsilon = \sum_{i \in e_t} \varepsilon_i \geq \frac{1}{m} \sum_{i \in e_t} \Delta^i_{t,max}$ (Theorem 1).

Remark. Our **ARR** mechanism **is uniquely suitable for LLM generation** under strict IND guarantees.

7 STUNING and LTOKENIZER

Model Misalignment. After deriving an effective INDVOCAB to prevent the cloud from reconstructing client prompts and code, we now focus on fine-tuning the encoder θ_e and decoder θ_d so the client can privately generate desirable code in NOIR. Despite the benefits of INDVOCAB, fine-tuning remains challenging: its privacy protection causes a misalignment between the encoder's output distribution and the cloud-hosted LLM's input distribution. This misalignment, amplified through the latent space of θ_m, further disrupts the alignment between encoder output and decoder input, degrading performance as the trade-off for IND protection.

STUNING. To address the model misalignment, we use split tuning: the client computes the loss function from the final-layer logits and the one-hot vector v_t of the next token t in the output y, derives the gradient of the decoder θ_d, and sends the gradient up to the last layer of the middle block θ_m back to the cloud. The cloud continues the back-propagation computing the gradient of the middle block θ_m's LoRA and sends the gradient up to the client's encoder, where the client computes the gradient of the encoder θ_e locally. The client updates θ_e and θ_d while the cloud updates θ_m's LoRA. This fine-tuning minimizes the average loss: $\arg\min_\theta \frac{1}{|D|} \sum_{\{x,y\} \in D} L(\{\tilde{e}_{t_j}\}_{j=1}^{|x|}, y, \theta)$, where $|D|$ is the number of data points $\{x,y\}$ in D and $\theta = \{\theta_e, \theta_d, \theta_m\text{'s LoRA}\}$, by updating θ, as follows: $\theta \leftarrow \theta - \gamma\omega$ s.t. $\omega = \frac{1}{|D|} \sum_{\{x,y\} \in D} \partial L(\{\tilde{e}_{t_j}\}_{j=1}^{|x|}, y, \theta)/\partial\theta$, where ω is the aggregated gradient and γ is a learning rate. STUNING computes gradients per data point $\partial L(\{\tilde{e}_{t_j}\}_{j=1}^{|x|}, y, \theta)/\partial\theta$ rather than over the aggregated statistics of multiple prompts, which damages the prompts' embeddings. STUNING updates the model parameters using the aggregated gradients, recovering the impact of the ε-IND noise until the model converges, yielding effective code generation models.

The power of STUNING lies in its effectiveness and efficiency in model performance and fine-tuning cost. It requires no extra cloud information, preserving CPC constraints, and aligns the encoder and the decoder to mitigate the INDVOCAB's impact. The cloud updates a lightweight LoRA at a negligible cost, while the client can fine-tune θ_e and θ_d directly—only one and four attention blocks are needed for encoder and decoder, respectively. As a result, NOIR delivers strong code generation performance cost-effectively, ideal for resource-limited clients. The cloud may skip fine-tuning θ_m's LoRA, lowering complexity with minimal utility loss. As shown in our **complexity and cost analysis** (Appx. D), **a single GPU suffices for client fine-tuning**, achieving high utility in code generation and completion.

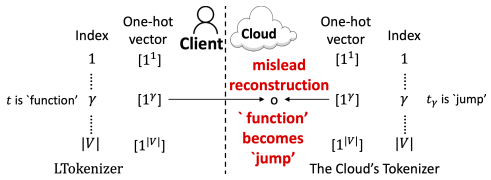


Figure 7: Local Randomized Tokenizer (LTOKENIZER).

LTOKENIZER. Although STUNING is effective, the cloud can exploit the back-propagated gradients of the middle block θ_m's LoRA, derived from raw one-hot vectors $\{v_t\}_{t \in y}$ of output tokens t, to reconstruct training data through BiSR [25]. Using the shared open-source tokenizer and the vocabulary V, the cloud maps each v_t to the client's token t. Sine INDVOCAB does not protect these vectors $\{v_t\}_{t \in y}$, NOIR introduces a local tokenizer (LTOKENIZER), which uniformly assigns every token and its embedding in the INDVOCAB $\tilde{V}$ to a random index in the tokenizer on the client side.

Figure 7 illustrates assigning a token t (e.g., *'function'*) and its embedding $\tilde{e}_{t_1}$ in the INDVOCAB to a random index γ. The one-hot vector v_t becomes $[1^\gamma]$, where only the γ^{th} element is 1 and all remaining elements are 0. If t is the next token to be generated in the output y, the gradient minimizes the loss between $v_t = [1^\gamma]$ and the decoder's output logits. This gradient misleads the cloud's RAs into reconstructing the token at index γ of the cloud's tokenizer with its vocabulary V, yielding an irrelevant token (e.g., *'jump'*) instead of the client's token (*'function'*). Figure 3 shows such a meaningless reconstruction. LTOKENIZER is data-independent and incurs no extra privacy cost.

The cloud's probability of inferring a client's token index is $1/|V|$, which is negligible; thus, the client's secret one-hot vectors v_t and tokenizer remain protected. The client employs LTOKENIZER with INDVOCAB in both STUNING and inference (Alg. 1).

Vulnerability to Averaging Attacks. There is no need to randomize the vocabulary or one-hot vector assignments per prompt. PromptRA, CodeRA, and frequency analysis attacks are types of averaging attacks using public data to reconstruct tokens. Our evaluation shows they are ineffective against NOIR. As long as the cloud does not observe any auxiliary information about the client tokens, embeddings, prompts, or code beyond what INDVOCAB and LTOKENIZER protect,

reconstruction risks remain upper-bounded. It is because averaging over IND-preserving token embeddings does not leak extra information due to DP's post-processing property. Averaging attacks might succeed only if an internal actor colludes with the cloud, which lies outside NOIR's threat model.

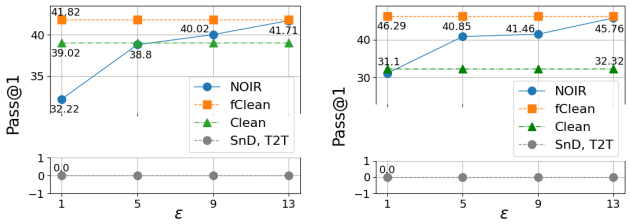
8 Experimental Results

Our extensive experiments shed light on the trade-off between client-side privacy and model utility against cloud RAs. NOIR surpasses the SoTA with strong privacy against the cloud while remaining cost-effective through minimal client fine-tuning, advancing protection of client prompts and code in generation and completion.

Configurations. We use CodeLlama-7B, CodeQwen1.5-7B-Chat, and Llama3-8B-instruct models with vocab sizes of 32k, 92,416, and 128,256 tokens, respectively; CodeLlama-7B is the default. The client runs one and four attention blocks per model as the encoder and the decoder, with the remaining blocks cloud-hosted. Each token embedding has m=4,096 features, with uniform ε/m across all features in e_t. The temperature h is 0.25, yielding the best performance across mechanisms. As aforementioned, we open-source NOIR as a privacy-preserving coding agent based on Qwen2.5-Coder-32B-Instruct via a web service and a VS extension.

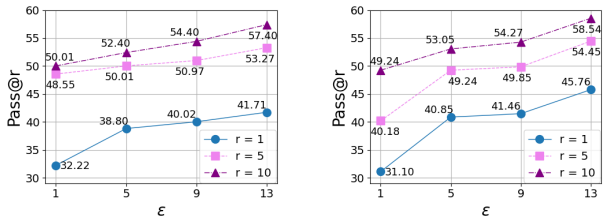
Datasets. We evaluate NOIR using the Evalplus benchmark [60], Mostly Basic Python Problems (MBPP) [34] for code generation and HumanEval [27] for code completion, and the BigCodeBench benchmark [28]. The client fine-tunes the encoder and the decoder on the (public) CodeAlpaca dataset [50] consisting of ∼18k data points for Python and then evaluates NOIR on the (private) HumanEval dataset. We fine-tune the encoder and decoder again on the MBPP dataset as the client's private data. In the ablation study, we enrich the CodeAlpaca dataset to ∼376k data points by adding code from the Stack dataset [61].

Baselines. We consider SoTA d_x-privacy-preserving mechanisms for protecting user prompts, which are **T2T** [19–21] and **SnD** [22]. Defense-free mechanisms include the **fClean** model, i.e., our NOIR without INDVOCAB and LTOKENIZER, and the **Clean** model, i.e., the original LLM model. We use Pass@r [27] to evaluate the model performance: $\text{Pass@}r = \mathbb{E}_{\text{prompts}}\left[1 - \binom{n-c}{r} / \binom{n}{r}\right]$, where $\mathbb{E}_{\text{prompts}}$ is the expectation over all prompts; for a given prompt: r is the requested number of generated code versions, n is the total number of generated code versions, and c is the total number of correct code versions passing all unit tests. As in [27], we set $n = 2r$. Pass@r is a significant and standard metric for quantifying model performance in generating functioning code because it evaluates the probability that at least one out of r independently generated code solutions is correct [27]. This aligns closely with practical deployment scenarios where multiple outputs are reviewed or tested [62]. It is particularly valuable in code synthesis, where a single correct solution



(a) MBPP dataset (b) HumanEval dataset

Figure 8: Pass@1 and IND budget ε.



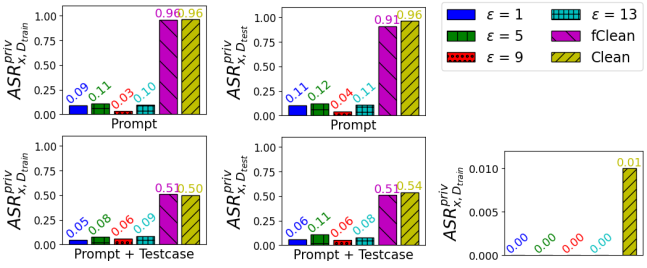
(a) MBPP dataset (b) HumanEval dataset

Figure 9: Pass@$r \in \{1, 5, 10\}$ and IND budget ε.

among several generated candidates is sufficient for success. Also, Pass@r encourages diversity and quality in outputs, which is crucial for robust code generation systems [62]. **P3T approaches** [15–17] **are not appropriate baselines** since they do not protect content of client's prompts and generated code as in our context.

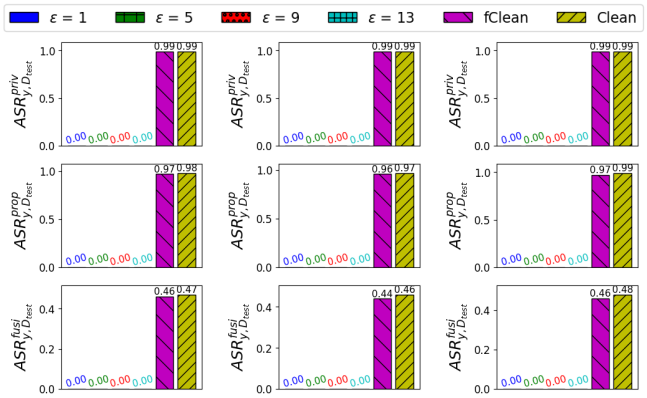
Q1: What is the trade-off between privacy and model performance? Figure 8 shows that SnD and T2T fail to generate code in both code generation and completion tasks across all IND budgets ε. They only produce code-like outputs at very large $\varepsilon > 5{,}000$, but their Pass@1 remains 0, consistent with prior findings [22]. In contrast, NOIR consistently outperforms these SoTA baselines. Higher ε, indicating weaker IND protection, yields better Pass@1. At $\varepsilon = 13$, NOIR achieves Pass@1 scores of 41.71 (MBPP) and 45.76 (HumanEval), with only marginal drops of 0.26% and 1.14% compared with upper-bounded defense-free performance. This strong performance stems from its embedding-based design, adaptive noise in INDVOCAB, and effective STUNING.

Q2: What is the cost to significantly improve model performance with strong IND protection? With stronger protection (smaller ε), Pass@1 performance drops more sharply (Figure 8). For instance, at $\varepsilon = 1$ (very strong IND), NOIR's Pass@1 scores are 32.22 (MBPP) and 31.1 (HumanEval). Fortunately, the drop can be mitigated by generating multiple code versions r per prompt and selecting the best based on passed test cases. Even at $\varepsilon = 1$, NOIR achieves much higher Pass@10 ($r = 10$): 50.01 (MBPP) and 49.24 (HumanEval) registering 55% and 58.3% improvements respectively, as shown in Figure 9. The gain grows with larger ε, and NOIR Pass@10 even surpasses the defense-free Pass@1 at the cost of producing more code versions.



(a) $ASR^{priv}_{x,D_{train}}$ in Training Set (MBPP) (b) $ASR^{priv}_{x,D_{test}}$ in Test Set (MBPP) (c) $ASR^{priv}_{x,D_{test}}$ in HumanEval

Figure 10: NOIR against PromptRA. The lower, the better.



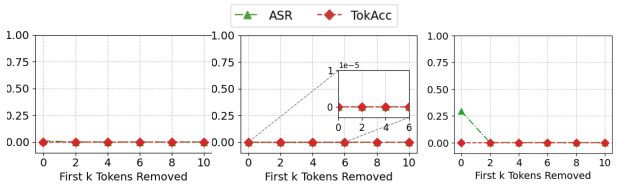
(a) MBPP Traning (b) MBPP Test (c) HumanEval

Figure 11: NOIR against CodeRA. The lower, the better.

Q3: What is the cost for the performance improvement? The cost is marginal. The client generates multiple code versions of a prompt as the input. Thus, the local computational cost is $(r-1)$ multiplied by the number of test cases for each prompt to identify the best code to use, which is negligible given the lightweight decoder. The communication and financial costs are $(r-1)$ times to generate this extra code in Pass@r, as a trade-off for the significant performance boost.

Q4: How effective is NOIR in defending against RAs? Figures 10, 11 show the privacy ASRs of PromtpRA and CodeRA under NOIR's defense. On MBPP's training set without test cases in prompts, NOIR significantly reduces PromptRA's privacy ASR (Figure 10a) from 96% (no test case) and 51% (with test cases) of the defense-free mechanisms to 7% on average at $\varepsilon = 1$, achieving 92.7% and 86.27% improvements. Similar reductions appear on MBPP's test set (Figure 10b). For HumanEval code completion, the PromptRA's privacy ASR is 0.0, showing it cannot reconstruct a clear gist of any prompts (Figure 10c), which is intuitive since Vec2Text [24] and BiSR [25] were not designed for pure code prompts with INDVOCAB and LTOKENIZER.

Although PromptRA is ineffective in code completion, CodeRA shows severe ASRs in privacy, confidentiality, and functionality (99%, 97%, and 46%, respectively) against the defense-free mechanism on HumanEval (Figure 11c). With IND protection and LTOKENIZER, NOIR reduces privacy and confidentiality ASRs to 0% across IND budgets ε, and the cloud cannot reconstruct functioning code (Fusi ASR = 0.0%). Similar results hold for code generation on MBPP training and test sets (Figures 11a,11b). Under NOIR, CRT_x and CRT_y are 0.028 and 0.108 respectively, far smaller than 0.28 and 1.00 of the defense-free mechanism on MBPP. Sensitive information `leak` is 0.038 (NOIR), compared with 0.98 (defense-free). The reconstructed top-5 tokens prompts and code {'⟨s⟩', '_', 'a', '.', 'to'} and {'per', 'i', 'f', 'res', 'p'} are meaningless. The client retains high model performance under strong IND protection with marginal cost (Q3).



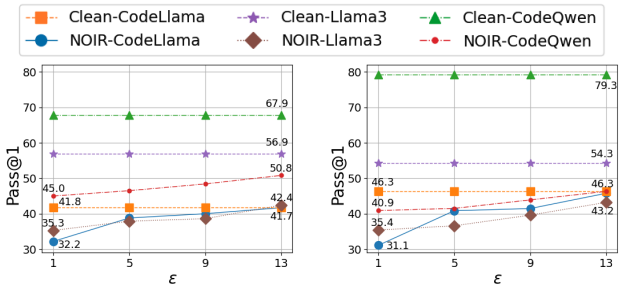
(a) MBPP test using 376k dataset on sequence-based attack (b) HumanEval using 376k dataset on sequence-based attack (c) MBPP test using MBPP train dataset on sequence-based attack

Figure 12: Frequency analysis attacks ($k = 3$).

Q5: Is NOIR robust against frequency analysis attacks? In this experiment, the cloud performs sequence-based (Figure 12) and token-based frequency analysis attacks, following [63] and [44]. A token-based attack is a sequence-based one with the sequence length $k = 1$. Out of 32k vocabulary tokens, the cloud only recovers a few from the instruction template, common to all prompts, but none from the client prompts x and code y. Specifically, token-based attack reconstruct 6 tokens {'Write', 'a', 'Python', 'function', 'to', '⟨|im_end|⟩'}, while sequence-based attacks (Figure 12) recover 3 tokens {'Write', 'a', 'Python'}. After excluding these, the privacy ASR is 0.0. Similar results hold when using public data (e.g., MBPP training set) with a similar distribution with the client's local data (e.g., MBPP test set). These findings confirm that NOIR resists frequency analysis attacks.

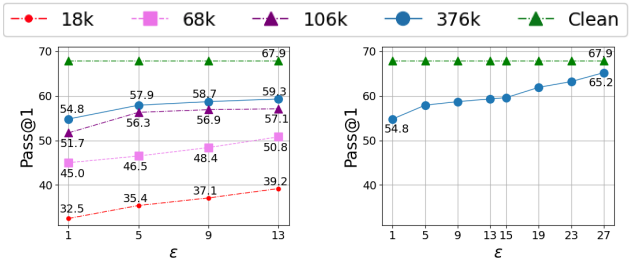
Q6: What is the model performance for different LLMs and vocabulary sizes? We vary the LLMs by using CodeQwen1.5-7B-Chat, Llama3-8B-instruct, and CodeLlama-7B models, which differ in capabilities and vocabulary size. CodeQwen1.5-7B-Chat in NOIR achieves the best performance on MBPP and HumanEval (Figure 13). A gap remains between NOIR with CodeQwen1.5-7B-Chat and the clean model, due to insufficient training data and small privacy budgets. We address this by increasing both and analyzing the resulting RA costs.

Q7: What is the effect of enlarging the training data and privacy budget? When clients (e.g., small enterprises) lack sufficient training data, they can augment it with publicly available data without disclosing the augmented set. For in-



(a) MBPP dataset (b) HumanEval dataset

Figure 13: Varying LLM models.



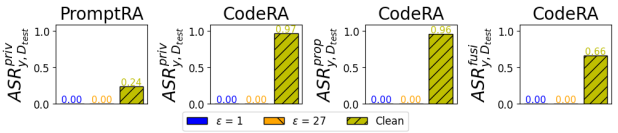
(a) Enlarging the training dataset (b) Increase the privacy budget ε

Figure 14: Enlarge the training data (CodeQwen1.5-7B-Chat).

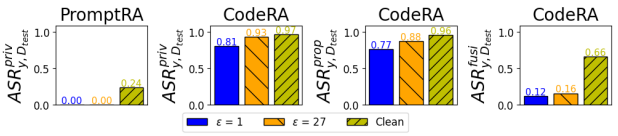
stance, we gradually expand CodeAlpaca from from ∼18k to ∼68k, ∼106k, and ∼376k data points by adding Python code from the Stack dataset [61]. As Figure 14a shows, NOIR's Pass@1 significantly improves from 39.2 to 59.3 on MBPP, reducing the gap to the Clean model (CodeQwen1.5-7B-Chat) from 42.3% to 12.7% at $\varepsilon = 13$. With 376k data points, raising ε further improves Pass@1 from 54.8 to 65.2, nearly matching the defense-free fClean at 67.9 (a marginal 3.9% drop; Figure 14b). On HumanEval, Pass@1 rises from 46.3 to 65.9 as ε increases from 13 to 27. On BigCodeBench "Instruct", NOIR's Pass@1 achieves 31.1 versus 31.7 for the clean model at $\varepsilon = 27$, a marginal 1.8% drop. Notably, NOIR remains robust against CodeRA and PromptRA with ASRs $= 0.0$.

Q8: What is the cost of enlarging the training data and privacy budget regarding RAs? Figure 15a shows that increasing the privacy budget and training data does not raise reconstruction ASRs (~ 0.0). This is because the task instruction π of the CodeQwen1.5-7B-Chat model combined with the prompt x is longer and involves more tokens in the vocabulary than in CodeLlama-7B, reducing the upper-bounded reconstruction probability (Section 6.3). Therefore, CodeQwen1.5-7B-Chat is more secure under INDVOCAB and LTOKENIZER protection. Hence, privacy budget increases should be paired with larger training data to balance performance and privacy.

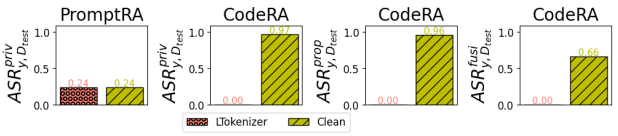
Q9: What are the roles of INDVOCAB and LTOKENIZER against RAs? This experiment highlights the defensive roles of these components. CodeRA becomes very effective when the client disables LTOKENIZER and uses the open-source tokenizer (Figure 15b). PromptRA takes its turn



(a) NOIR



(b) NOIR without LTOKENIZER



(c) NOIR without INDVOCAB

Figure 15: Enlarging training data to 370k data points and IND ε to 27 under PrompRA and CodeRA.

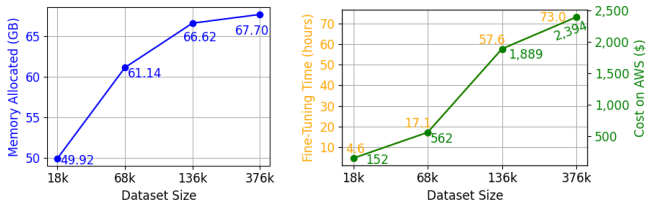
to be effective when the client disables INDVOCAB and uses the open-source vocabulary (Figure 15c). Thus, INDVOCAB and LTOKENIZER defend against PromtpRA and CodeRA, respectively. Combining them is crucial for producing high-quality code while resisting attacks (Figure 15a).

Q10: What is the effectiveness of STUNING, including LoRA and zeroth-order (ZO) optimization? Training only the decoder or a shortcut model (concatenating the encoder and the decoder while removing the middle block θ_m) yields poor performance (Pass@1 $= 0.0$ at $\varepsilon = 13$) with CodeLlama-7B. This is expected: decoder-only training cannot counter IND-preserving noise amplified in the large latent space of θ_m, whie shortcut-only training and inference ignores key features learned by θ_m. Hence, integrating encoder, decoder, and θ_m in STUNING and inference is essential for good utility.

Applying LoRA to the encoder and decoder yields little benefit due to their lightweight design. Instead, we fine-tune the encoder using the SoTA ZO optimization [64], which approximates gradients via enriched embeddings. However, ZO requires 50 extra inferences per data point per training iteration and offers no notable performance gain.

Q11: What is the client cost? We use an A100-80GB GPU to measure the client's cost of NOIR fine-tuning on CodeQwen1.5-7B-Chat across datasets from 18k to 376k data points. As shown in Figure 16, GPU memory usage, fine-tuning time, and equivalent AWS hosting costs grow sub-linearly with dataset size, ensuring scalability without excessive client communication cost. A 20.89x dataset size increase raises GPU memory usage only 1.36x (49.92GB → 67.70GB; Figure 16a), while AWS fees remain affordable for small enterprises ($152 → $2,394; Figure 16b). Therefore, our system scales efficiently with minimal cost.

Q12: Can we reduce the cloud cost? The cloud has the



(a) GPU Memory (b) Time and Cost on AWS

Figure 16: Client's fine-tuning cost on 1 A100 GPU (80GB).

option not to fine-tune the θ_m's LoRA to reduce its computational complexity with a marginal drop $\sim$2% of model utility as a trivial trade-off.

Q13: What are the best numbers of attention blocks? We set $\varepsilon = 13$ and study how the number of attention blocks in the encoder and decoder affects performance. Fixing the encoder to one block, increasing the number of decoder's attention blocks improves NOIR's performance. In contrast, adding attention blocks to the encoder does not help, as privacy noise propagates further into the latent space, limiting the cloud ability to enrich prompt embeddings and making encoder/decoder fine-tuning difficult.

Q14: What is the NOIR's end-to-end latency? To evaluate NOIR's latency, we deploy the system with Qwen2.5-Coder-32B-Instruct when the encoder/decoder and middle block servers are in the same cloud region. Each server is equipped with a single A100 80GB GPU. For comparison, we also deploy the fully connected model on a single server with the same GPU achieving 22.96 tokens/s, whereas NOIR's latency is 21.49 tokens/s. These results indicate that when the client and cloud servers are co-located in a close proximity, NOIR incurs only a marginal latency overhead.

9 Discussion

In this section, we discuss limitations in terms of security auditing and practicality of NOIR and potential solutions.

Practicality. LLM vendors may not have strong incentives/motivation to adopt split learning. However, third-party providers (independent of LLM creators) can leverage existing open-source LLMs to provide NOIR, which will allow them to charge data security and IP sensitive enterprises for this privacy-preserving service. NOIR requires seamless integration between client-side (lightweight encoder/decoder) and cloud-hosted (middle block) components to make this model practical.

Security Auditing. While stronger security auditing mechanisms remain unavailable, adversaries could exploit advanced adaptive attacks to undermine ε-IND. A powerful adversary might control the cloud-side LoRA to fine-tune middle blocks, reducing or eliminating noise effects. By analyzing the distribution of IND-preserved embeddings across multiple prompts, they could model noise patterns and dynamically adjust LoRA to improve reconstruction accuracy. A potential defense is to explore privacy-preserving LoRA protocols, training the cloud's LoRA via zero-knowledge proofs or trusted execution environments to safeguard client data patterns. Such solutions may not be cost effective, and therefore further research is necessary to explore this topic.

Cross-embedding similarity could reveal semantic relationships between tokens, while token clustering across prompts might infer the original token's semantic role, such as identifying "function" from consistent contextual embeddings like "def" or "defining a function." Also, cross-prompts clustering could uncover semantic structures, enabling attackers to bypass ε-IND through pattern inference. The key idea of these adaptive attacks is to discover frequent sequences of token embeddings then match them with frequent sequences of tokens extracted from public data. To defend against these sophisticated adversaries, NOIR will prioritize enhancing its privacy mechanisms. This includes developing adaptive strategies, such as context-dependent **dynamic prefix tokens** using encoder attention block position embeddings to alter tokenization during training or inference, thereby preventing the cloud from learning semantic patterns.

In a pilot study, we considered these cross-embedding-based adaptive attacks by clustering token embeddings based on 1) their similarity, 2) their positions, and 3) both their similarity and positions using public data (MBPP training set) on the Qwen2.5-Coder-32B-Instruct and Qwen3-32B models to identify common sequences across prompts. The position-based clustering attack (i.e., sequences of token embeddings are matched with sequence of tokens in the public data based on their positions in the prompts) shows the most promising result by recovering ε-IND-preserving embeddings ($\varepsilon = 27$) of a few persistent tokens {'Python', 'function', 'a', 'write', 'to'} at fixed sequence patterns across prompts in the system prompt template. Only a few tokens are identified because the encoder attention block position embeddings disrupt the token embeddings, making it difficult to exploit the token embedding similarity. Randomly adding a dynamic prefix up to 8 random characters, e.g., <A@1p?He5>, to the system and instruction prompts prevents this marginal leak of the ε-IND-preserving token embeddings by further disrupting the embedding sequence patterns (in terms of both similarity and positions) across prompts without affecting model performance given the model's strong reasoning capability (Appx. E). Also, overly strict protection, such as increasing the dynamic prefix token to 20 characters, can notably degenerate the model performance. Further research is needed to bolster NOIR's defense against adaptive and cross-prompts attacks, ensuring robust protection in practice.

Text Generation. NOIR is limited to private code generation due to its unique challenge, fine-tuning data, and pipeline rather than general text generation. Extending NOIR to general text generation involves adapting the vocabulary and tokenizer to handle diverse natural language tasks, modifying the training data and architecture to support text generation,

and ensuring robust privacy protections against reconstruction attacks. Our pilot study shows that NOIR achieves highly competitive reasoning performance using Qwen open-source LLMs (Appx. E). A NOIR's trial system for general text generation is available: https://noir.oppyai.com.

Several other research directions include: **(1) Embedding Alignment:** The cloud must avoid responding to malicious embeddings from harmful prompts (e.g., "generating malware code"), a harder task than detecting harmful prompts. **(2) Proprietary LLMs:** Model architecture, vocabulary, and token embedding size are sensitive to the proprietary LLM-provider. A solution is for clients to use open-source encoders/decoders, while the cloud aligns embedding sizes with its proprietary LLM via feature transformation (e.g., NVIB [65]). Pilot results (Appx. F) show promising performance on benchmarks.

10 Conclusion

This paper presents NOIR, a code generation framework that protects clients' prompts and code using open-source LLMs. Clients send prompt embeddings derived from a local encoder to the cloud-hosted model and receive enriched embeddings, which are decoded locally to generate code. To prevent cloud reconstruction of prompts and code, we introduce a novel concept of ε-IND-preserving vocabulary (INDVOCAB) with a randomized tokenizer (LTOKENIZER), and propose STUNING, a client-side fine-tuning method for the encoder and decoder. Extensive theoretical and empirical results show that NOIR significantly outperforms SoTA baselines while safeguarding data privacy, code confidentiality, and code functionality against reconstruction and frequency analysis attacks from an honest-but-curious cloud. To demonstrate practicality, we open-source NOIR as a privacy-preserving coding agent using a Qwen2.5-Coder-32B-Instruct model with Pass@1: 83.6 (MBPP), 85.4 (HumanEval), and 47.2 (BigCodeBench).

Acknowledgment

This work was supported by the National Science Foundation (NSF) grants CNS-(1935928, 1935923), CNS 2237328, and DGE 2043104. We thank Thai Nguyen for the study in Appx. F and the dedicated reviewers who help improve NOIR.

Ethical Considerations

Our research offers the potential to benefit all stakeholders, including: 1) Individuals and enterprises, who use LLMs as software coding tools; and 2) LLM-providers. No stakeholders will be adversely impacted by the publication of your research now and in the future, since we develop a solution to protect IP and data security in prompting LLMs for private code generation without affecting any currently available commercial systems.

Open Science

We open-source NOIR (https://tinyurl.com/NOIR-Artifact) based on Qwen2.5-Coder-32B-Instruct and release its API through a privacy-preserving coding agent for the public use in a web-service (https://noir.oppyai.com) and a Visual Studio extension (https://tinyurl.com/NOIR-Artifact), integrated directly into the software development pipeline. The source-code is available with detailed experimental configurations for reproducibility purposes.

References

[1] Github. Research: Quantifying github copilot's impact on code quality, 2023.

[2] CNBC. The biggest risk corporations see in gen ai usage isn't hallucinations, 2024.

[3] T-Minus365. Microsoft 365 copilot | security risks how to protect your data, 2024. [accessed 17-June-2024].

[4] Concentric.ai. Too much access? microsoft copilot security concerns explained, 2024. [accessed 17-June-2024].

[5] Search Engine Journal. Openai is pulling shared chatgpt chats from google search, 2025.

[6] PC Mag. Samsung software engineers busted for pasting proprietary code into chatgpt, 2023.

[7] Fortune Business Insights. Low code development platform market, 2024. [Online; accessed 17-June-2024].

[8] Guangxuan Xiao, Ji Lin, and Song Han. Offsite-tuning: Transfer learning without full model, 2023.

[9] Medium. Hybrid llm strategies for enterprises: Balancing self-hosted governance with cloud flexibility, 2025.

[10] Chenxi Wang. Do enterprises want open-weight models?, 2025.

[11] Paloalto Network. Why self-managed ai models are blind spots and what to do about it, 2025.

[12] Digital Trade. Using self-hosted large language models (llms) securely in government, 2025.

[13] Forbes. Generative ai breaks the data center: Data center infrastructure and operating costs projected to increase to over 76 billion by 2028, 2023.

[14] Construction Physics. How to build an ai data center, 2024. [Online; accessed 17-June-2024].

[15] Haonan Duan and et al. Flocks of stochastic parrots: Differentially private prompt learning for large language models. In *NeurIPS*, 2023.

[16] Tong Wu and et al. Privacy-preserving in-context learning for large language models. In *ICLR*, 2024.

[17] Junyuan Hong and et al. DP-OPT: Make large language model your privacy-preserving prompt engineer. In *ICLR*, 2024.

[18] Xinyu Tang and et al. Privacy-preserving in-context learning with differentially private few-shot generation. In *ICLR*, 2024.

[19] Chen Qu and et al. Natural language understanding with privacy-preserving bert. In *CIKM*, 2021.

[20] Oluwaseyi Feyisetan and et al. Privacy- and utility-preserving textual analysis via calibrated multivariate perturbations. In *WSDM*, 2020.

[21] Yansong Li, Zhixing Tan, and Yang Liu. Privacy-preserving prompt tuning for large language model services, 2023.

[22] Peihua Mai and et al. Split-and-denoise: Protect large language model inference with local differential privacy. In *ICML*, 2024.

[23] Konstantinos Chatzikokolakis and et al. Broadening the scope of differential privacy using metrics. In *PETS'13*.

[24] John Morris and et al. Text embeddings reveal (almost) as much as text. In *EMNLP*, 2023.

[25] Guanzhong Chen and et al. Unveiling the vulnerability of private fine-tuning in split-based frameworks for large language models: A bidirectionally enhanced attack. In *CCS*, 2024.

[26] Cycode. Top source code leaks 2020-2025, 2025.

[27] Mark Chen and et al. Evaluating large language models trained on code. *arXiv:2107.03374*, 2021.

[28] Terry Yue Zhuo and et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877*, 2024.

[29] Oscar Skean and et al. Layer by layer: Uncovering hidden representations in language models. In *ICML'25*.

[30] Praneeth Vepakomma and et al. Split learning for health: Distributed deep learning without sharing raw patient data. *CoRR*, 2018.

[31] Zheng Lin and et al. Split learning in 6g edge networks. *IEEE Wireless Communications*, 2024.

[32] Dario Pasquini, Giuseppe Ateniese, and Massimo Bernaschi. Unleashing the tiger: Inference attacks on split learning. CCS, 2021.

[33] Edward J Hu and et al. LoRA: Low-rank adaptation of large language models. In *ICLR*, 2022.

[34] Jacob Austin and et al. Program synthesis with large language models. *arXiv:2108.07732*, 2021.

[35] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *CCS*, 2014.

[36] Mistral AI. Tokenization, 2024.

[37] Cynthia Dwork et al. The algorithmic foundations of differential privacy. *Found. and Trends in Theoretical CS*, 2014.

[38] Shiva Prasad Kasiviswanathan and et al. What can we learn privately? In *FOCS*, pages 531–540, 2008.

[39] John C. Duchi, Michael I. Jordan, and Martin J. Wainwright. Local privacy and statistical minimax rates. In *FOCS*, pages 429–438, 2013.

[40] Jianmo Ni and et al. Large dual encoders are generalizable retrievers, 2021.

[41] Zehan Li and et al. Towards general text embeddings with multi-stage contrastive learning. *arXiv:2308.03281*, 2023.

[42] Congzheng Song and Ananth Raghunathan. Information leakage in embedding models. In *CCS*, 2020.

[43] Haoran Li, Mingshi Xu, and Yangqiu Song. Sentence embedding leaks more information than you expect: Generative embedding inversion attack to recover the whole sentence. In *ACL*, July 2023.

[44] Alex Biryukov. *Codebook Attack*. Springer US, 2011.

[45] Kishore Papineni and et al. Bleu: a method for automatic evaluation of machine translation. ACL, 2002.

[46] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Sum. Branches Out*, 2004.

[47] Google Cloud. Evaluating models. [Online; accessed 17-June-2024].

[48] klu.ai. What is the rouge score (recall-oriented understudy for gisting evaluation)?

[49] Shuo Ren and et al. Codebleu: a method for automatic evaluation of code synthesis. *CoRR*, 2020.

[50] Sahil Chaudhary. Code alpaca: An instruction-following llama model for code generation, 2023.

[51] Colin Raffel and et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv:1910.10683*, 2023.

[52] J. Mattern, B. Weggenmann, and F. Kerschbaum. The limits of word level differential privacy. ACL, 2022.

[53] S. L. Warner. Randomized response: A survey technique for eliminating evasive answer bias. *JASA*, 1965.

[54] J. C. Duchi, M. I. Jordan, and M. J. Wainwright. Local privacy and statistical minimax rates. In *IEEE Protocols for secure computations*, pages 429–438, 2013.

[55] J. Duchi and R.n Rogers. Lower bounds for locally private estimation via communication complexity. In *COLT*, pages 1161–1191, 2019.

[56] N. Wang and et al. Collecting and analyzing multidimensional data with local differential privacy. In *ICDE'19*.

[57] Y. Zhao and et al. Local differential privacy based federated learning for internet of things. *IEEE IoT-J*, 2020.

[58] L. Lyu, Y. Li, X. He, and T. Xiao. Towards differentially private text representations. In *ACM SIGIR*, 2020.

[59] T. Nguyen, P. Lai, N. Phan, and M. T. Thai. Xrand: Differentially private defense against explanation-guided attacks. *AAAI*, 2023.

[60] J. Liu and et al. Is your code generated by chatGPT really correct? rigorous evaluation of large language models for code generation. In *NeurIPS*, 2023.

[61] Denis Kocetkov and et al. The stack: 3 TB of permissively licensed source code. *TMLR*, 2023.

[62] EvidentlyAI. 30 llm evaluation benchmarks and how they work, 2025.

[63] M. Naveed, S. Kamara, and C. V. Wright. Inference attacks on property-preserving encrypted databases. CCS, 2015.

[64] Yihua Zhang and et al. Revisiting zeroth-order optimization for memory-efficient llm fine-tuning: A benchmark. In *ICML*, 2024.

[65] James Henderson and Fabio James Fehr. A VAE for transformers with nonparametric variational information bottleneck. In *ICLR*, 2023.

[66] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation onion router. USENIX, 2004.

[67] Ashish Vaswani and et al. Attention is all you need. NeurIPS, 2017.

[68] Chujie Zheng and et al. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025.

[69] Colin White and et al. Livebench: A challenging, contamination-limited llm benchmark. *arXiv preprint arXiv:2406.19314*, 2024.

A Proof of Theorem 1

Proof. Given two possible values e_t^i and $e_t^{i\prime}$ of the i^{th}-feature in the token embedding e_t, i.e., $e_t^i, e_t^{i\prime} \in \{e_l^i\}_{l\in V}$, and any possible output $z \in Range(ARR)$, where $Range(ARR)$ denotes every possible output of ARR, we have:

$$\frac{P(ARR(e_t^i)=z)}{P(ARR(e_t^{i\prime})=z)} \leq \frac{\max P(ARR(e_t^i)=z)}{\min P(ARR(e_t^{i\prime})=z)} = \frac{\max P(ARR(e_t^i)=z)}{\min P(ARR(e_k^i)=z)} \quad (6)$$

Following the typical definition of RR mechanisms [53], the probability p_i must be larger than or equal to any probabilities $q_{i,k}$. We assume having this condition:

$$\forall k \in V \setminus t : p_i \geq q_{i,k}. \quad (7)$$

From Eqs. 6 and 7, we have that

$$\frac{P(ARR(e_t^i)=z)}{P(ARR(e_t^{i\prime})=z)} \leq \frac{\frac{\exp(\beta_i)}{\exp(\beta_i)+|V|-1}}{\min(\frac{\exp(-\Delta_{t,k}^i/m)}{\sum_{l\in V}\exp(-\Delta_{t,l}^i/m)}\frac{|V|-1}{\exp(\beta_i)+|V|-1})}$$

$$= \frac{\exp(\beta_i)}{(|V|-1)\min(\frac{\exp(-\Delta_{t,k}^i/m)}{\sum_{l\in V}\exp(-\Delta_{t,l}^i/m)})} \leq \exp(\varepsilon_i). \quad (8)$$

Taking a natural logarithm of Eq. 6, we obtain: $\ln(\frac{\exp(\beta_i)}{(|V|-1)\min(\frac{\exp(-\Delta_{t,k}^i/m)}{\sum_{l\in V}\exp(-\Delta_{t,l}^i/m)})}) \leq \ln(\exp(\varepsilon_i))$

$$\Leftrightarrow \beta_i \leq \varepsilon_i + \ln(|V|-1) + \ln(\frac{\min\{\exp(-\Delta_{t,k}^i/m)\}_{k\in V\setminus t}}{\sum_{l\in V}\exp(-\Delta_{t,l}^i/m)}). \quad (9)$$

Let us recall that, for the Eq. 9 to hold, we need the condition in Eq. 7 to hold. In fact, we can rewrite Eq. 7 as follows: $p_i \geq \max\{q_{i,k}\}_{k\in V\setminus t} = \frac{|V|-1}{\exp(\beta_i)+|V|-1}\frac{\max\{\exp(-\Delta_{t,k}^i/m)\}_{k\in V\setminus t}}{\sum_{l\in V}\exp(-\Delta_{t,l}^i/m)}$.

This is equivalent to: $\frac{\exp(\beta_i)}{\exp(\beta_i)+|V|-1} \geq \frac{|V|-1}{\exp(\beta_i)+|V|-1}\frac{\max\{\exp(-\Delta_{t,k}^i/m)\}_{k\in V\setminus t}}{\sum_{l\in V}\exp(-\Delta_{t,l}^i/m)} \Leftrightarrow \exp(\beta_i) \geq (|V|-1)\frac{\max\{\exp(-\Delta_{t,k}^i/m)\}_{k\in V\setminus t}}{\sum_{l\in V}\exp(-\Delta_{t,l}^i/m)}$

$$\Leftrightarrow \beta_i \geq \ln(|V|-1) + \ln(\frac{\max\{\exp(-\Delta_{t,k}^i/m)\}_{k\in V\setminus t}}{\sum_{l\in V}\exp(-\Delta_{t,l}^i/m)}). \quad (10)$$

For Eqs. 9 and 10, which represent the upper-bound and the lower-bound of β_i respectively, to hold simultaneously, resulting in feasible β_i, we need the upper-bound to be larger than or equal to the lower-bound, as follows:

$$\varepsilon_i + \ln(|V|-1) + \ln(\frac{\min\{\exp(-\Delta_{t,k}^i/m)\}_{k\in V\setminus t}}{\sum_{l\in V}\exp(-\Delta_{t,l}^i/m)}) \geq \ln(|V|-1) +$$

$$\ln(\frac{\max\{\exp(-\Delta_{t,k}^i/m)\}_{k\in V\setminus t}}{\sum_{l\in V}\exp(-\Delta_{t,l}^i/m)}) \Leftrightarrow \varepsilon_i \geq \ln(\frac{\max\{\exp(\Delta_{t,k}^i/m)\}_{k\in V\setminus t}}{\min\{\exp(\Delta_{t,k}^i/m)\}_{k\in V\setminus t}}) \quad (11)$$

Let us denote $\Delta_{t,min}^i = \min\{\Delta_{t,k}^i\}_{k\in V\setminus t}$ and $\Delta_{t,max}^i = \max\{\Delta_{t,k}^i\}_{k\in V\setminus t}$. From Eq. 11, we have that

$$\varepsilon_i \geq \ln(\frac{\exp(\Delta_{t,max}^i/m)}{\exp(\Delta_{t,min}^i/m)}) \Leftrightarrow \varepsilon_i \geq \frac{1}{m}(\Delta_{t,max}^i - \Delta_{t,min}^i). \quad (12)$$

Consequently, from Eqs. 9, 10, and 12, Theorem 1 holds. □

B Theorem 2

Theorem 2. *Applying ARR to independently randomize every i^{th}-feature with a privacy budget ε_i in a token embedding e_t preserves ε-IND, where $\varepsilon = \sum_{i \in e_t} \varepsilon_i$.*

Proof. Given any two possible embeddings $e_t = \{e_t^i\}_{i \in e_t}$ and $e'_t = \{e_t^{i\prime}\}_{i \in e'_t}$ of a token t, i.e., $\forall i^{\text{th}}$-feature in e_t : $e_t^i, e_t^{i\prime} \in \{e_l^i\}_{l \in V}$, and any possible output $O = \{z^i \in Range(ARR)\}_{\forall i^{\text{th}}\text{-feature in } e_t}$, where $Range(ARR)$ denotes every possible output of ARR, we have: $\frac{P(ARR(e_t)=O)}{P(ARR(e'_t)=O)} =\leq \prod_{i \in e_t} \frac{\max P(ARR(e_t^i)=z^i)}{\min P(ARR(e_t^{i\prime})=z^i)} = \prod_{i \in e_t} \frac{\max P(ARR(e_t^i)=z^i)}{\min P(ARR(e_k^i)=z^i)} \leq \prod_{i \in e_t} \exp(\varepsilon_i) = \exp(\sum_{i \in e_t} \varepsilon_i)$. Consequently, Theorem 2 holds. □

C Prompt-level Protection

Token-level d_x-privacy has two key weaknesses: (1) privacy budget and authorship leakage scale with the number of tokens in a sentence, and (2) protection varies across sentences with different token counts [52]. Authorship leakage [23] is not a concern in our study, as the cloud knows the client's identity. While techniques like Tor [66] can address authorship privacy, we aim to establish an upper bound on the probability of prompt x being reconstructed, thereby exploring the relationship between INDVOCAB and reconstruction risk.

To achieve our goal, we introduce a security game (Figure 17) where the client selects an ε-IND-preserving token embedding $o \in \{\tilde{e}_t\}_{t \in V}$ and sends it to the cloud along with the original vocabulary V associated with the ground-truth token embeddings $\{e_t\}_{t \in V}$. The cloud wins if it correctly identifies the ground-truth token t corresponding to o. The cloud receives no feedback on its success, and the client does not provide additional information derived from the original embeddings, adhering to the CPC constraints. The frequency of game play or the selection of ε-IND-preserving embeddings does not compromise the IND protection due to its post-processing property [37]. Each game remains independent, as the cloud gains no insight into the INDVOCAB $\tilde{V}$ from participating in multiple games. Theorem 3 quantifies the cloud's probability of correctly inferring t from o, $P[ARR(e_t) \doteq o]$. Note that $\forall t \neq t' : \tilde{e}_t \neq \tilde{e}_{t'}$.

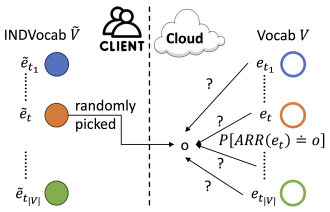


Figure 17: Token Reconstruction Security Game.

Theorem 3. *Given an ε-INDVOCAB $\tilde{V}$, the probability that an arbitrary $t \in V$ is the ground-truth token of an observed ε-IND token embedding $o \in \{\tilde{e}_t\}_{t \in V}$, denoted as $P[ARR(e_t) \doteq o]$, is bounded as follows:*

$$\forall t \in V : \frac{1}{1+(|V|-1)e^{\varepsilon}} \leq P[ARR(e_t) \doteq o] \leq \frac{e^{\varepsilon}}{e^{\varepsilon}+|V|-1}. \quad (13)$$

Proof. Given an arbitrary ε-IND-preserving token embedding $o \in \{\tilde{e}_t\}_{t \in V}$ and the vocabulary V, the cloud will make an inference whether o is a result of randomizing one of the $|V|$ token embeddings in the vocabulary. Therefore, given $|V|$ possible outcomes from the cloud perspective, we have

$$\sum_{t \in V} P[ARR(e_t) \doteq o] = 1 \quad (14)$$
$$\Leftrightarrow \forall t \in V : P[ARR(e_t) \doteq o] = 1 - \sum_{t' \in V \setminus t} P[ARR(e_{t'}) \doteq o].$$

From Definition 2 and Theorems 1-2, we also have that: $\forall t' \in V \setminus t, \forall O = \{z^i \in Range(ARR)\}_{\forall i^{\text{th}}\text{-feature in } e_t}$: $P[ARR(e_{t'}) = O] \geq \frac{P[ARR(e_t)=O]}{e^{\varepsilon}} \Leftrightarrow -P[ARR(e_{t'}) = O] \leq -\frac{P[ARR(e_t)=O]}{e^{\varepsilon}}$. Given the observed outcome $o \in O$, we have the that: $-P[ARR(e_{t'}) \doteq o] \leq -\frac{P[ARR(e_t) \doteq o]}{e^{\varepsilon}}$. By applying this result to all $t' \in V \setminus t$, we have that

$$-\sum_{t' \in V \setminus t} P[ARR(e_{t'}) \doteq o] \leq -(|V|-1)\frac{P[ARR(e_t) \doteq o]}{e^{\varepsilon}}. \quad (15)$$

From Eqs. 14 and 15: we have that $\forall t \in V : P[ARR(e_t) \doteq o] \leq 1-(|V|-1)\frac{P[ARR(e_t) \doteq o]}{e^{\varepsilon}}$

$$\Leftrightarrow P[ARR(e_t) \doteq o] \leq \frac{1}{1+\frac{|V|-1}{e^{\varepsilon}}} = \frac{e^{\varepsilon}}{e^{\varepsilon}+|V|-1}. \quad (16)$$

In addition, we also have that $\forall t' \in V \setminus t, \forall O = \{z^i \in Range(ARR)\}_{\forall i^{\text{th}}\text{-feature in } e_t}$: $P[ARR(e_{t'}) = O] \leq e^{\varepsilon} P[ARR(e_t) = O]$. Given the observed outcome $o \in O$, we have the that: $P[ARR(e_{t'}) \doteq o] \leq e^{\varepsilon} P[ARR(e_t) \doteq o]$. By applying this result to all $t' \in V \setminus t$, we have that

$$1 - \sum_{t' \in V \setminus t} P[ARR(e_{t'}) \doteq o] \geq 1 - (|V|-1)e^{\varepsilon} P[ARR(e_t) \doteq o]. \quad (17)$$

From Eqs. 14, 17, we have $P[ARR(e_t) \doteq o] \geq 1-(|V|-1)e^{\varepsilon} P[ARR(e_t) \doteq o]$

$$\Leftrightarrow P[ARR(e_t) \doteq o] \geq \frac{1}{1+(|V|-1)e^{\varepsilon}}. \quad (18)$$

Consequently, from Eqs. 16 and 18, Theorem 3 holds. □

From Theorem 3, it is evident that the *"larger the number of tokens"* in the vocabulary V and the *"smaller the privacy budget ε"* is, the *"lower the probability"* for the cloud to infer the ground-truth token, offering rigorous privacy protection.

We can extend the security game by allowing the client to pick an arbitrary prompt $x = \{t_j\}_{j=1}^{|x|}$ represented as a sequence of ε-IND preserving tokens $\{o_j\}_{j=1}^{|x|}$ s.t. $\forall j \in [1,|x|]$: $o_j = \tilde{e}_{t_j}$ and the client sends $\{o_j\}_{j=1}^{|x|}$ to the cloud. The cloud wins the game if its reconstructed prompt $\hat{x}$ provides a clear gist of x by identifying ground-truth tokens $\{t_j\}$ in x. For instance, $Bleu(\hat{x},x) \geq \rho$ $(= 20)$ for a clear gist of x [47], formulated as $P[Bleu(\hat{x},x) \geq \rho; \{o_j\}_{j=1}^{|x|}]$. Like the previous security game, the client does not send the game outcome to the

cloud. Therefore, **the cloud observes no extra information** derived from the ground-truth token embeddings **in playing the games**. As a result, the number of games, the number of times a token appears in one or more prompts, and the number of times a prompt is selected in these games do not affect the ε-IND protection of the token embeddings (the post-processing property of DP [37]).

In this security game, the numbers of tokens in the reconstructed and ground-truth prompts, $\hat{x}$ and x, are the same: $|\hat{x}| = |x|$. Hence, Rouge precision is equal to Rouge recall: $\textit{Rouge-precision}(\hat{x},x), \textit{Rouge-recall}(\hat{x},x) = C/|x|$, where C is the number of correct reconstructed tokens in $\hat{x}$. As a result, $\textit{Rouge-F1}(\hat{x},x)$ [46] is equal to $C/|x|$, which is equivalent to $\textit{Bleu}(\hat{x},x) = C/|x|$. Therefore, we focus on analyzing the correlation between the IND budget ε, the vocabulary size $|V|$, the size of the prompt $|x|$, and the threshold ρ in reconstructing a clear gist of x using the *Bleu* score below. The following proposition limits the cloud's upper-bounded probability of winning the security game.

Proposition 1. *The cloud's probability to identify ground-truth tokens in x with a gist level higher than or equal ρ is upper-bounded as* $P\big[\textit{Bleu}(\hat{x},x) \geq \rho; \{o_j\}_{j=1}^{|x|}\big] \leq \big(\frac{\psi e^{\varepsilon}+1}{\psi e^{\varepsilon}+\psi^2}\big)^{\rho|x|} \times \big(\frac{\psi e^{\varepsilon}}{\psi e^{\varepsilon}+1}\big)^{(1-\rho)|x|}$, *where* $\psi = |V|-1$.

Proof. Let us recall the Bleu score definition as follows: $\textit{Bleu}(\hat{x},x) = BP(\hat{x},x) \times C/|x|$, where $BP(\hat{x},x)$ stands for the brevity penalty given for the mismatched length between the reconstructed prompt $\hat{x}$ and ground-truth prompt x, and C is the number of correct reconstructed tokens in $\hat{x}$. In our security game, $BP(\hat{x},x) = 1$ since $\hat{x}$ and x have the same length.

The probability of having C correct reconstructed tokens, assuming $\{\hat{t}_j = t_j\}_{j=1}^{C}$ and $\{\hat{t}_j \neq t_j\}_{j=C+1}^{|x|}$ without loss of generality, is:

$$P\big(\{\hat{t}_j = t_j\}_{j=1}^{C}, \{\hat{t}_j \neq t_j\}_{j=C+1}^{|x|}; \{o_j\}_{j=1}^{|x|}\big) = \prod_{j=1}^{C} P[ARR(e_{t_j}) \doteq o_j] \times \prod_{j=C+1}^{|x|} P[ARR(e_{t_j}) \not\doteq o_j], \quad (19)$$

where $P[ARR(e_{t_j}) \not\doteq o_j]$ indicates the cloud's probability to infer than t_j is not the ground-truth token of the observed ε-IND preserving token embedding o_j.

From Theorem 3, we have that $\prod_{j=1}^{C} P[ARR(e_{t_j}) \doteq o_j] \leq \big(\frac{e^{\varepsilon}}{e^{\varepsilon}+|V|-1}\big)^{C}$. In addition, we have that

$$P[ARR(e_{t_j}) \not\doteq o_j] = 1 - P[ARR(e_{t_j}) \doteq o_j] \quad (20)$$

$$\leq 1 - \frac{1}{1+(|V|-1)e^{\varepsilon}} = \frac{(|V|-1)e^{\varepsilon}}{1+(|V|-1)e^{\varepsilon}}. \quad (21)$$

Let's denote $\psi = |V|-1$, from Eqs. 13, 19, and 21, we have

$$P(\{\hat{t}_j = t_j\}_{j=1}^{C}, \{\hat{t}_j \neq t_j\}_{j=C+1}^{|x|}; \{o_j\}_{j=1}^{|x|}) \leq \big(\frac{e^{\varepsilon}}{e^{\varepsilon}+|V|-1}\big)^{C} \times \quad (22)$$

$$\big(\frac{(|V|-1)e^{\varepsilon}}{1+(|V|-1)e^{\varepsilon}}\big)^{|x|-C} = \big(\frac{\psi e^{\varepsilon}+1}{\psi e^{\varepsilon}+\psi^2}\big)^{C} \times \big(\frac{\psi e^{\varepsilon}}{\psi e^{\varepsilon}+1}\big)^{|x|-C}, \quad (23)$$

From Eq. 23, the probability to reconstruct a clearer gist of a prompt x, i.e., C increases, is reduced monotonically. It is because $\psi \gg 1$ and $C \in [0,|x|]$. Therefore, we have that: $P\big[\textit{Bleu}(\hat{x},x) \geq \rho; \{o_j\}_{j=1}^{|x|}\big] = P\big[C \geq \rho|x|; \{o_j\}_{j=1}^{|x|}\big] \leq P\big[C = \rho|x|; \{o_j\}_{j=1}^{|x|}\big] = P(\{\hat{t}_j = t_j\}_{j=1}^{\rho|x|}, \{\hat{t}_j \neq t_j\}_{j=\rho|x|+1}^{|x|}; \{o_j\}_{j=1}^{|x|}) \leq \big(\frac{\psi e^{\varepsilon}+1}{\psi e^{\varepsilon}+\psi^2}\big)^{\rho|x|} \times \big(\frac{\psi e^{\varepsilon}}{\psi e^{\varepsilon}+1}\big)^{(1-\rho)|x|}$.

Consequently, Proposition 1 holds. □

Theorem 4. *The cloud's previously reconstructed token sequences $\hat{t}_{<j}$ can enhance its probability of correctly reconstructing the next token t_j: $P(\hat{t}_j = t_j|\hat{t}_{<j})$. This advantage is bounded by a constant γ in practice, as follows:*

$$\forall t_j \in x : 0 \leq P(\hat{t}_j = t_j|\hat{t}_{<j}) - P(\hat{t}_j = t_j) \leq \gamma, \textit{ where } \gamma \in [0,1]. \quad (24)$$

Given a bounded constant γ, we extend Proposition 1, bounding the probability to reconstruct a gist level higher than ρ of the prompt x exploiting $\hat{t}_{<j}$, as follows:

$$P\big[\textit{Bleu}(\hat{x},x) \geq \rho; \{o_j\}_{j=1}^{|x|}\big] \leq \big(\frac{\psi e^{\varepsilon}+1}{\psi e^{\varepsilon}+\psi^2}+\gamma\big)^{\rho|x|} \times \big(\frac{\psi e^{\varepsilon}}{\psi e^{\varepsilon}+1}-\gamma\big)^{(1-\rho)|x|}.$$

Proof. The probability of having C correct reconstructed tokens, assuming $\{\hat{t}_j = t_j|\hat{t}_{<j}\}_{j=1}^{C}$ and $\{\hat{t}_j \neq t_j|\hat{t}_{<j}\}_{j=C+1}^{|x|}$ without loss of generality, is: $P\big[\textit{Bleu}(\hat{x},x) \geq \rho; \{o_j\}_{j=1}^{|x|}\big]$

$$= P\big[C \geq \rho|x|; \{o_j\}_{j=1}^{|x|}\big] \leq P\big[C = \rho|x|; \{o_j\}_{j=1}^{|x|}\big] \quad (25)$$

$$= P(\{\hat{t}_j = t_j|\hat{t}_{<j}\}_{j=1}^{\rho|x|}, \{\hat{t}_j \neq t_j|\hat{t}_{<j}\}_{j=\rho|x|+1}^{|x|}; \{o_j\}_{j=1}^{|x|}). \quad (26)$$

From Eq. 24, we have that $\forall j : P(\hat{t}_j = t_j|\hat{t}_{<j}) \leq P(\hat{t}_j = t_j) + \gamma$. Let us consider the worst-case for the client, in which the cloud has the maximal probability of correctly reconstructing every token t_j, denoted as $P^*(\hat{t}_j = t_j|\hat{t}_{<j}) = P(\hat{t}_j = t_j) + \gamma$. The corresponding probability of incorrectly reconstructing a token t_j is: $P(\hat{t}_j \neq t_j|\hat{t}_{<j}) = 1 - P^*(\hat{t}_j = t_j|\hat{t}_{<j})$. From Eq. 26,

$$P\big[\textit{Bleu}(\hat{x},x) \geq \rho; \{o_j\}_{j=1}^{|x|}\big] \leq \prod_{j=1}^{\rho|x|} P^*(\hat{t}_j = t_j|\hat{t}_{<j}) \prod_{j=\rho|x|+1}^{|x|} P(\hat{t}_j \neq t_j|\hat{t}_{<j})$$

$$= \big(\frac{\psi e^{\varepsilon}+1}{\psi e^{\varepsilon}+\psi^2}+\gamma\big)^{\rho|x|} \times \big(\frac{\psi e^{\varepsilon}}{\psi e^{\varepsilon}+1}-\gamma\big)^{(1-\rho)|x|} \quad (27)$$

Consequently, Theorem 4 holds. □

D Complexity and Cost Analysis

We evaluate the complexity and cost of operating NOIR. Let n and d be the numbers of input tokens and the hidden size of attention layers. The network communication between the client and cloud involves two phases: sending the encoder's output embedding $\mathcal{E}$ (complexity $O(nd)$) and receiving the enriched embedding $\ddot{\mathcal{E}}$ (cost $O(nd)$), resulting in a total transmission cost of $O(nd)$. On the client side, the computation consists of three stages: generating the privatized token embedding ($O(n)$), and feeding embeddings through the encoder and decoder, each with complexity $O(n^2d)$, leading to a total cost of $O(n^2d)$ (comparable with [67]). Fine-tuning costs

are notably reduced, as the client computes the decoder's gradients twice and the encoder's gradients once per training iteration, compared to computing gradients for the entire LLM. In fact, using 1 and 4 attention blocks for the encoder and decoder, respectively, in CodeLlama-7B results in 71.9% lower tuning costs. On the cloud side, prompting and hosting costs are reduced because the prompt embedding passes through fewer attention blocks (27 vs. 32), lowering the cloud prompting and hosting cost by 15.6%. NOIR is scalable, requiring minimal resources for fine-tuning/operation.

E Pilot General Text Generation Pipeline

To adapt NOIR for general text generation, we propose a six-phase fine-tuning strategy mirroring pre-training while escalating reasoning complexity. **Phase 1** trains the encoder, decoder, and middle block's LoRA on simple math/logic datasets to generate explicit reasoning steps. **Phase 2** integrates multi-hop and scientific reasoning datasets to synthesize coherent reasoning from multiple facts. **Phase 3** strengthens problem-solving across domains (math, code, writing) using high-quality datasets for deeper abstract reasoning. **Phase 4** extends handling of long contexts (up to 32,000 tokens) and multi-turn conversations with datasets like patents and dialogues, enabling interactive reasoning and plan revision. **Phase 5** employs Group Sequence Policy Optimization (GSPO) [68] to align reasoning sequences, ensuring structured, repeat-free outputs with correct final answers. **Phase 6** tailors the model to enterprise use cases via domain-specific corpora while retaining cross-domain capabilities. A small subset of prior data is retained in all phases to prevent forgetting. We deployed a Qwen3-32B with $\varepsilon = 27$ (https://noir.oppyai.com) and on-going trials with Qwen3-235B-A22B-Instruct using this training pipeline. The curated dataset has total of $\sim$2.7m data points across six phases. We achieve highly competitive performance across reasoning tasks on the LiveBench [69] while maintaining ε-IND (Table 3). Note that, poor data quality and fine-tuning pipeline can significantly lower NOIR's performance rendering it unpractical for enterprises.

	NOIR	Qwen3-32B
Reasoning Average	66.5	48.2
Mathematics Average	71.8	67.4
Instruction Following Average	70.2	17.8
Language Average	41.3	55.0

Table 3: NOIR's Performance on LiveBench-2026-01-08. Qwen3-32B's Performance is Recorded on the Leader Board.

F Preliminary Results on Proprietary LLMs

To adapt NOIR for black-box middle blocks from proprietary LLMs, we introduce a representation-learning module that maps both sides into a shared, pre-agreed embedding space, enabling seamless integration. This approach extends the Non-parametric Variational Information Bottleneck [65] (NVIB) autoencoder, which offers attention-like expressivity, model-agnostic flexibility, and theoretical rigor, making it ideal for bridging heterogeneous LLM components.

Figure 18 shows the D-VAE adapter block, which consists of two paired NVIB autoencoders. One autoencoder processes client-to-cloud representations, while the other handles cloud-to-client responses. Each NVIB encoder compresses input hidden states into a shared latent space with a pre-defined dimensionality, ensuring dimensional alignment and distribution compatibility across heterogeneous models. The corresponding decoders reconstruct the latent variables into embeddings that match the expected interface of the next component.

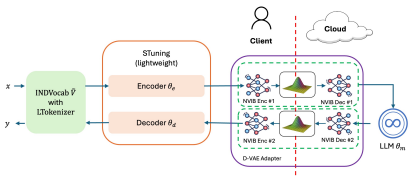


Figure 18: D-VAE structure for encoder/decoder alignment.

To evaluate our approach, we instantiate the client-side components using Qwen2.5-Coder-7B-Instruct, selecting the first attention block as the encoder and the last as the decoder for NOIR. The cloud-side middle block is set as the entire attention stack from LLaMA-3.1-8B, treated as a fixed black-box module. We train the end-to-end system for three epochs under two data regimes: a small-scale setting with 18k samples from the CodeAlpaca dataset and a large-scale setting with 100k samples from our curated dataset.

```
"""
  Prompt: Create a Python function that
  returns all the prime numbers between
  1 and a given number.
"""

def prime_numbers(n):
    primes = []
    for i in range(2, num + 1):
        is_prime = True
        for j in range(2, i):
            if i % j == 0:
                is_prime = False
                break
        if is_prime:
            primes.append(i)
    return primes
```

Figure 19: An example of generated code and prompt.

Our preliminary results show that the D-VAE aligns the two model structures. In the 18k CodeAlpaca setting, the system achieves a Pass@1 of 2.8% on MBPP, indicating that limited data suffices for learning a functional cross-model interface. Scaling to 100k curated examples significantly boosts performance to 12.4% Pass@1, underscoring the D-VAE adapter's ability to leverage larger datasets and enhance downstream code generation despite the heterogeneity and black-box nature of the intermediate LLM. Although the preliminary performance is far from practically usable, Figure 19 shows that the architecture generates coherent, syntactically valid, and task-appropriate code solutions, highlighting its potential and suggesting further improvements with scaling and refinement.