SharedRequest: Privacy-Preserving Model-Agnostic Inference for Large Language Models

Anonymous ACL submission

Abstract

With the widespread deployment of public large language models (LLMs) such as ChatGPT, protecting user prompt privacy has become an increasingly critical issue. Existing privacy-preserving inference methods often compromise either utility or computational cost. In this paper, we propose SharedRequest, a model-agnostic and privacy-preserving framework for LLM inference. SharedRequest is independent of the LLM architecture, requiring no model modifications or access to internal parameters. It obscures sensitive information by mixing original prompts with noisy variants and amortizes the inference cost over a large batch of queries. The LLM server only observes a shuffled mix of queries, including original and noisy, without revealing user identities. By clustering semantically equivalent instructions, our mechanism reduces per-prompt token charges with minimal impact on LLM response quality. Empirical results demonstrate that SharedRequest achieves over 20% higher utility compared to prior differential privacy–based techniques, and its shared-prompt mechanism reduces query cost by up to $5\times$ compared to non-batched inference.

1 Introduction

Rapid advancement of large language models (LLMs) has enabled transformative applications in healthcare, finance, and personal assistance. State-of-the-art public LLMs, including ChatGPT, Claude, and Gemini, are primarily deployed on the cloud platform, which protects proprietary model architectures but raises significant privacy concerns. User prompts often contain sensitive information that should remain confidential from any third party, underscoring the urgent need for effective privacy-preserving inference frameworks.

Existing approaches to private LLM inference face trade-offs among privacy, utility, and inference overhead. Secure multi-party computation (SMPC) solutions, such as Iron (Hao et al., 2022) and SecFormer (Luo et al., 2024), leverage cryptographic tools to ensure the secrecy of user input. However, these methods suffer from high communication and computation costs, limiting their practical deployments. Another line of work introduces randomization during data transmission to guarantee local differential privacy (LDP). For instance, DP-Forward (Du et al., 2023) and SnD (Mai et al., 2024) inject noise into intermediate representations in split learning. Nevertheless, these methods cause significant utility degradation and are primarily designed for classification tasks, making them unsuitable for text generation.

The aforementioned frameworks rely on LLM-specific architectural modifications. In contrast, current model-agnostic solutions focus on prompt perturbation to obfuscate sensitive contents. These methods enforce LDP by paraphrasing inputs using temperature-controlled decoding (Mattern et al., 2022), or replacing tokens based on the adjacency list (Chen et al., 2023). While protecting privacy in a black-box setting, such techniques often struggle to balance privacy with response utility.

Fortunately, LLMs typically process large batches of queries within a given time window, offering opportunities to amortize inference costs. Additionally, in many cases, sensitive information within user prompts is sparse, and not all tokens require protection. For example, in the query "Provide career suggestions for a person working in cybersecurity", only the word "cybersecurity" is sensitive. Based on these observations, it is desirable to leverage the batch processing and the sparsity of sensitive content to mitigate the trade-offs in privacy-preserving LLM inference.

In response, this paper proposes SharedRequest, a privacy-preserving LLM inference paradigm. Our approach samples noisy variants of sensitive content and mixes them with original prompts to obscure private information. By clustering seman-

tically equivalent generic instructions, we amortize the additional inference cost related to noisy queries across a large batch of requests. We design a lightweight cryptographic protocol to ensure secure communication among multiple parties. Our protocol offers the following advantages:

- **Model-agnostic**: requires no modification to the LLM architecture or access to model parameters. The service provider only needs to integrate lightweight decryption and masking modules around the LLM, treating the model as black-box.
- **Utility-preserving**: achieves substantially better utility than perturbation-based methods, with minor degradation due to clustering and prompt simplification.
- **Cost amortization**: amortizes inference cost across large batches, reducing per-user cost as batch size increases.

Our contributions are summarized as follows:

(1) We propose a novel paradigm for privacy-preserving LLM inference by injecting noisy queries and distributing the additional inference cost across a large batch of users.

(2) We design a lightweight cryptographic protocol that enables secure multi-party communication, ensuring that the LLM server observes only a shuffled set of requests (original and noisy) without learning user identities.

(3) To optimize user-side computation, we introduce a coordination mechanism with a third party for efficient noisy query sampling, with reduced complexity especially under high volume of private contents within each prompt.

2 Related Work

2.1 SMPC-based Private Transformer Inference

SMPC enables multiple parties to conduct transformer model inference while ensuring that the server learns nothing about user inputs (Yao, 1982). Iron (Hao et al., 2022) introduces a hybrid cryptographic protocol tailored for matrix multiplications and complex non-linear operations, including Softmax, GELU, and LayerNorm. MPCFormer (Li et al., 2023) integrates MPC with knowledge distillation to approximate expensive functions cryptographically. SecFormer (Luo et al., 2024) builds on this foundation by replacing nonlinear operations with optimized SMPC-friendly approximations and redesigning Softmax, GeLU, and LayerNorm. BOLT (Pang et al., 2024) reduces the payload through cryptographic optimizations, minimizing homomorphic encryption rotations via a baby-step giant-step algorithm and employs word-elimination heuristics. NEXUS (Zhang et al., 2024) introduces the first non-interactive SMPC protocol for transformer inference based on RNS-CKKS homomorphic encryption, reducing latency and interaction overhead compared to traditional multi-round protocols. Despite their strong privacy guarantees, these SMPC-based methods entail substantial computational and communication overhead, limiting their scalability in practical deployment.

2.2 Differentially Private Prompt Engineering

LDP mechanism has been explored to privatize user prompts before they are sent to cloud-based LLMs. Early methods convert tokens into embeddings, add manipulated noises, and map the embeddings back, effectively preserving differential privacy (Lyu et al., 2020a,b). To achieve better utility, recent approaches differentiate sensitive from non-sensitive tokens and replace sensitive tokens using constrained adjacency lists rather than full vocabulary sampling (Tong et al., 2025; Chen et al., 2023; Yue et al., 2021). For example, CusText assigns smaller replacement sets per token to reduce semantic drift while maintaining privacy. Other approaches include paraphrase-based mechanisms such as DP-Prompt, which produce privatized outputs via temperature-controlled sampling during text rewriting (Utpala et al., 2023; Mattern et al., 2022). While LDP-based prompt privatization methods can operate with black-box models, they generally face fundamental challenges in balancing trade-offs between utility and privacy.

	Text Generation	Model-Agnostic	Utility-Preserving
Iron	×	×	✓
MPCFormer	×	×	✓
BOLT	×	×	✓
NEXUS	×	×	✓
RanText	✓	✓	×
CusText	✓	✓	×
DP-Prompt	✓	✓	×
DP-Forward	×	×	×
SnD	×	×	×
SharedRequest	✓	✓	✓

Table 1: Comparison among transformer-based privacy-preserving inference frameworks.

3 Background

3.1 Problem Formulation

A service provider owns a proprietary generative language model $LM : \mathcal{V}^* \rightarrow \mathcal{V}^*$ such as ChatGPT and Gemini (OpenAI, 2023; Hurst et al., 2024; Team et al., 2023), where $\mathcal{V}^*$ represents the space of all token sequences over the vocabulary $\mathcal{V}$. Given a user prompt q, our framework aims to protect its private attributes $A_q = \{a_1, a_2, ..., a_{|A(q)|}\}$ against the service provider.

The user's prompt q consists of two parts: (i) the generic instruction $T_q \in \mathcal{T}$, and (ii) the personal attributes $A_q \in \mathcal{A}$. For example, given the prompt "What are the recommended restaurants in Britain?", the personal attribute is "Britain" and the generic instruction is "What are the recommended restaurants". For any generic instruction $T \in \mathcal{T}$, we define the plausible attribute set $\mathcal{A}(T) \subset \mathcal{A}$ as those $\mathcal{A}(T)$ that co-occurs with T with non-zero probability (e.g., various locations compatible with that instruction). In the following, we use A_q and $A(q)$ interchangeably to denote the private attributes of prompt q.

3.2 A Simple Construction

Instead of transmitting a single prompt to the cloud LLM, we propose sending a set of prompts, containing original and noisy prompts, to obscure the user's sensitive information.

Given a prompt q with private attributes A_q and generic instruction T_q, the user samples a set of noisy attributes $\mathcal{A}'(T_q)$ from the plausible attribute set $\mathcal{A}(T_q)$. Each sampled attribute $A'(T_q) \in \mathcal{A}'(T_q)$ is then embedded into generic instruction T_q to form noisy prompts. The noisy prompts, together with the original, are submitted to the server, and only the response to the true prompt q is retained locally.

While providing accurate responses and preserving privacy, such construction introduces two practical challenges. Firstly, the query cost scales by a factor of $|\mathcal{A}'(T_q)|$ compared to a non-private scenario. Secondly, for prompts containing multiple private attributes, it becomes resource-intensive for the user to generate a series of plausible combinations. In the following, we propose a framework to mitigate the query cost and user computation overhead.

4 Framework

4.1 Overview

Our protocol involves grouping prompts and sampling noisy queries through a third party. We posit that the service provider receives a large volume of requests within each time window, making it highly likely that a certain number of prompts share the same generic instruction. For example, ChatGPT processes more than 1 billion queries every day, which translates to more than 11,500 queries per second (Singh, 2025). By grouping requests based on their generic instructions, we can amortize query costs and server computation time.

As depicted in Figure 1, our framework consists of three parties:

Users hold their queries with private attributes to be kept hidden from the service provider. For each prompt p, they identify the private attributes A_p and sample noisy attributes for each individual private attribute $a \in A_p$.

Noise Sampler receives queries with encrypted private attributes from the user. It clusters prompts with the same generic instruction and samples noisy queries to obfuscate original prompt.

Service Provider receives queries from the noise sampler and generates answers per query. The answers are masked with users' secrets and transmitted back to the noise sampler.

Appendix A outlines the algorithm for our privacy-preserving inference framework.

4.2 Cryptographic Design

Our protocol aims to ensure that: (i) The noise sampler has no knowledge of any sensitive data and only observes the generic instruction. (ii) The service provider's view consists of an anonymous set of shuffled prompts, including genuine and noisy ones. A core design challenge lies in ensuring secure communication among users, noise sampler, and service provider, while also preserving the user's anonymity from the service provider.

4.2.1 Forward Transmission

The forward stage transmits of the message from the user to the service provider. During this phase, the user encrypts the private attributes using the service provider's public key pk_s, and the service provider decrypts the ciphertext with its private key sk_s. This asymmetric encryption scheme hides the private attribute from any third parties, including the noise sampler. Furthermore, any messages sent

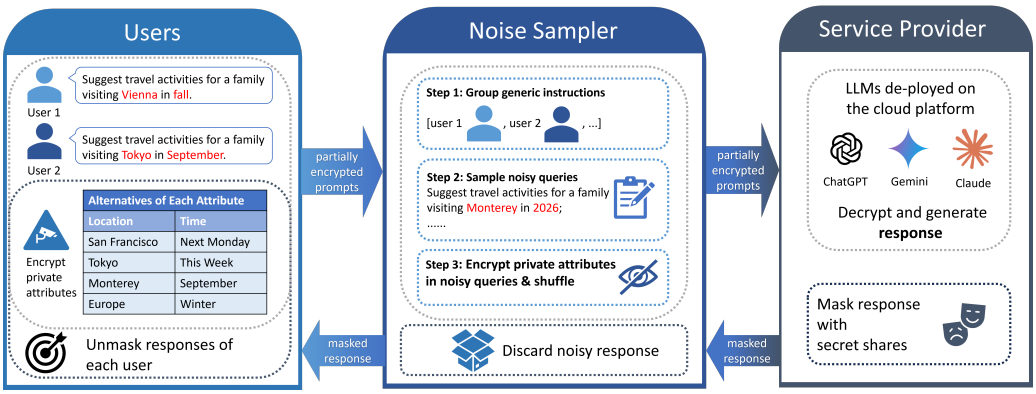

Figure 1: Overview of SharedRequest. The noise sampler groups user requests based on their generic instruction, samples noisy prompts with each group, and sends a shuffled mix to the server for inference. The inference costs related to noisy queries are amortized across users.

to the noise sampler are encrypted with the sampler's public key pk_n, preventing eavesdroppers (including the service provider) from learning their contents.

4.2.2 Backward Transmission

The backward stage transmits the response from the service provider to the user. The response may contain information related to private attributes, and thus it should be concealed from the noise sampler. This phase enables the user to decrypt the response without revealing their identity to the service provider. To accomplish this, we design a masking-based obfuscation scheme: (i) The user sends a seed s encrypted with the service provider's public key. (ii) The service provider decrypts the seed s and uses a pseudorandom generator (PRG) to derive a pseudo-random mask $e = PRG(s)$, which is used to obfuscate the response via modular addition $r_s = r + e$. (iii) Upon receiving r_s, the user reconstructs the response via modular subtraction $r = r_s - e$.

4.3 Grouping and Prompt Simplification

4.3.1 Grouping

To amortize query cost, the noise sampler groups prompts with semantically equivalent generic instructions to share the same plausible attribute sets. We leverage a sentence transformer to encode each instruction into embedding, and apply clustering to group similar embeddings into batches automatically. To improve clustering efficiency and downstream utility, the sampler first pre-groups prompts based on the number of sensitive attributes, enabling parallelized clustering within each subgroup.

4.3.2 Prompt Simplification

Users may optionally apply prompt simplification to improve efficiency by: (i) reducing private attributes, which lowers the sampling and query cost; (ii) standardizing generic instructions, enabling the plausible attributes to be amortized among more queries. Simplification can be achieved through manual prompt engineering or automated tools, such as lightweight local models.

4.4 Noisy Query Sampling

When a query involves multiple private attributes, sampling plausible attribute combinations becomes significantly more complex. If there are μ private attributes, each with k plausible alternatives, the total number of possible combinations grows to k^{μ}. To address this combinatorial explosion, we introduce a coordinated approach between the user and the noise sampler.

4.4.1 Candidates for Individual Attributes (User)

Rather than sampling full attribute *combinations*, the user independently selects alternative attributes for each *single* attribute: $\{\mathcal{A}'_1, \mathcal{A}'_2, ..., \mathcal{A}'_{|A(q)|}\}$, where $|A(q)|$ is the number of attributes within prompt q and $\mathcal{A}'_i$ is the set of alternatives for the ith attribute. These alternatives can be manually curated or drawn from a pre-constructed attribute-class database.

4.4.2 Combination Selection (Noise Sampler)

It would be prohibitively expensive to sample and send all combinations to the service provider, with payload exponential in the attribute size. To optimize the query cost, the noise sampler chooses combinations that look genuine within the context:

(i) randomly chooses a set of candidate combinations from the users' individual attribute lists; (ii) employs a pre-trained discriminator to score each combination's genuineness within context; (iii) selects only those combinations whose scores exceed a pre-specified threshold δ, obtaining qualified combination set $\mathcal{A}^n$.

To ensure adequate coverage, i.e., each of the μ attributes has all candidates with probability $\geq p$, the noise sampler draws m combinations satisfying:

$$m \geq (\log(1-p) - \log(\mu k)) \,/\, \log(1 - 1/k), \quad k = \max\{k_1, k_2, ..., k_\mu\}, \tag{1}$$

where k_i is the number of candidates for the ith attribute. The derivation is detailed in Appendix B. In practice, we can set $m = \alpha \cdot (\log(1-p) - \log(\mu k)) \,/\, \log(1 - 1/k)$, where α is a tuning factor to balance privacy guarantees and system overhead.

4.4.3 Combination Sampling (Noise Sampler)

Directly combining noisy attributes with real ones can inadvertently expose genuine queries. If a genuine attribute appears multiple times in a batch while a fake one appears only once, an adversarial server would distinguish real queries by frequency patterns. Since private attributes are encrypted, the noise sampler can not guarantee that noisy attributes will match the frequency of genuine ones.

To mitigate this issue, we propose a sampling method based on one-sided exponential mechanism (Rozanov, 2012; Kozubowski and Podgórski, 2000; Takagi et al., 2022). Instead of injecting each qualified combination exactly once, we add randomized counts $\mathcal{N}[v]$ for each attributes combination $v \in \mathcal{A}^n$. Specifically, for each v, the noise sampler draws λ_v from the one-sided exponential distribution:

$$f(\lambda_v) = \begin{cases} \frac{\epsilon}{2} \exp \frac{-\lambda_v \epsilon}{2} & \lambda_v \geq 0, \\ 0 & \lambda_v < 0, \end{cases} \tag{2}$$

with privacy parameter ϵ. The sampler then injects $\mathcal{N}[v] = \lfloor \lambda_v \rfloor$ instances of each combination $v \in \mathcal{A}^n$. Even though each query might appear multiple times, the query cost maintains the same as the service provider groups identical prompts and generates a single response per group for payload optimization (see Appendix F.3).

5 Privacy Analysis

5.1 Threat Model

We assume that both the service provider and noise sampler are curious-but-honest (Yang et al., 2019). Furthermore, we assume that there is no collusion between the noise sampler and service provider. In Appendix C.1, we extend our framework using multi-server architecture and signature scheme (Kaur and Kaur, 2012; Katz, 2010) to withstand stronger adversary models, including malicious noise sampler and collusion between servers.

5.2 Noise Sampler

The noise sampler's view consists of: (i) each user's generic instruction T_q; (ii) each user's encrypted private attributes and seed; (iii) candidate alternatives for each individual attribute; and (iv) masked response for each user. The IND-CPA security of the encryption scheme guarantees that the sampler cannot infer any sensitive information from ciphertexts. Moreover, given that the seed used for masking is chosen uniformly at random, the security of the PRG ensures that the random mask e and thus the obfuscated output r_s are computationally indistinguishable from random numbers. Therefore, the noise sampler gains no information about the user prompt except its generic instruction and candidate alternatives.

5.3 Service Provider

The service provider's view is a shuffled set of prompts, including both genuine and noisy queries. To formulate the privacy protection, we introduce the notion of $(\mathcal{A}^n, \epsilon)$-indistinguishability. Appendix C.3 elaborates its post-processing and composition rules.

Definition 1 ($\mathcal{A}^n$-Neighbors)**.** Denote A_q and T_q as, respectively, the private attributes and generic instruction of user's prompt q. Let $\mathcal{A}^n(T_q) \subset \mathcal{A}$ be the set of qualified alternative attributes for T_q. Two sets of prompts, Q' and Q, are $\mathcal{A}^n$-neighbors if Q' can be obtained from Q by changing one prompt q in Q on its private attributes from A_q to $A'_q \in \mathcal{A}^n(T_q)$. We denote $Q' \in N_{\mathcal{A}^n}(Q)$.

Definition 2 ($(\mathcal{A}^n, \epsilon)$-indistinguishability)**.** For any $\epsilon > 0$, a randomized mechanism $M : \mathcal{Q} \to \mathcal{Y}$ preserves $(\mathcal{A}^n, \epsilon)$-indistinguishability if for $\forall Q, Q' \in N_{\mathcal{A}^n}(Q)$, the following inequality holds:

$$\Pr[M(Q) \in S] \leq e^{\epsilon} \Pr[M(Q') \in S], \tag{3}$$

for all subsets S of the output space $\mathcal{Y}$.

Theorem 3. *Denote $M : \mathcal{P} \to \mathcal{Y}$ as our protocol that maps user prompt batches to the shuffled output seen by the service provider. For any $\epsilon > 0$, the mechanism M achieves $(\mathcal{A}^n, \epsilon)$-indistinguishability.*

This theorem ensures that the service provider's view remains indistinguishable with regards to the change in private attributes.

5.3.1 Connection to Differential Privacy

In the following, we show that the definition of $(\mathcal{A}^n, \epsilon)$-indistinguishability is a customization and relaxation of differential privacy (DP) (Dwork, 2006).

Definition 4 (Differential Privacy)**.** Given $\epsilon > 0$ and $\delta \in [0, 1)$, a randomized mechanism preserves (ϵ, δ)-differential privacy if and only if, for any neighboring sets of prompts $Q, Q' \in \mathcal{Q}$, the following inequality holds:

$$\Pr[M(Q) \in S] \leq e^{\epsilon} \Pr[M(Q') \in S] + \delta$$

for all subsets S of the output space $\mathcal{Y}$.

In standard DP, two datasets Q' and Q are considered neighbors if they differ by exactly one prompt, i.e., one query $q \in Q$ is replaced to any other prompt $q' \in Q'$. We can obtain $(\mathcal{A}^n, \epsilon)$-indistinguishability from standard DP by setting the constraints: (1) only change q's private attribute A_q, and (2) private attributes are replaced by the prompt's qualified alternatives $\mathcal{A}^n(T_q)$. Conversely, standard DP can be obtained from $(\mathcal{A}^n, \epsilon)$-indistinguishability by treating A_q as the full prompt q, and allowing $\mathcal{A}^n(T_q)$ to include all possible text values within the context length. Therefore, $(\mathcal{A}^n, \epsilon)$-indistinguishability can be considered as a user-defined variant of DP, with qualified alternatives and thus neighbors co-specified by users and noise sampler.

6 Complexity Analysis

The computation cost of our protocol is primarily determined by: (i) noise sampler's grouping and attribute combination filtering cost; (ii) optional user-side query simplification cost; (iii) service provider's LLM inference cost; (iv) encryption and decryption operations. Since (iv) is negligible relative to (i)-(iii) according to our empirical analysis, we focus on the costs related to the former two components.

6.1 Noise Sampler Computation Cost

The primary computational overhead for the noise sampler arises from batch grouping and attribute filtering. The batch grouping involves: (i) encoding each generic instruction into embedding; and (ii) clustering queries based on the embeddings. The first part is fully parallelizable across instructions, and the second part depends on the chosen clustering algorithm (Ester et al., 1996; Sumengen et al., 2021; Pelleg and Moore, 1999). Modern clustering algorithms, such as Reciprocal Agglomerative Clustering (RAC), enable parallel processing, scaling efficiently to billions of points. Additionally, combination filtering, which evaluates plausible attribute combinations, costs $O\left(\log(\mu k)/\log(1 - 1/k)\right)$ per generic instruction. In practice, this can also be parallelized across samples, further reducing running time.

6.2 User Simplification Cost

Prompt simplification is performed optionally and can be executed manually or via automated methods. For complexity analysis, we assume a local model deployed by the user. Consequently, the computational cost depends directly on the model's architecture and the lengths of the original and simplified prompts.

6.3 Query Cost from Service Provider

We consider query cost as the token-based charge from commercial LLM APIs, mostly associated with service provider's LLM inference cost. To mitigate this cost, the server aggregates identical prompts and generates the response once per group rather than handling each individually. The following theorem shows how query cost depends on the distribution of generic instructions.

Theorem 5. *Suppose a batch contains B genuine queries, each drawn independently from an underlying distribution. Denote c_i and p_i as, respectively, the average query cost and sampling probability for generic instruction $T_i, i \in [|\mathcal{T}|]$. Then the expected per-query cost C is:*

$$E(C) = \frac{1}{B} \sum_{i=1}^{|\mathcal{T}|} \left(1 - (1 - p_i)^B\right) c_i. \quad (4)$$

If $c_i = c$ is equal for all generic instructions $T_i, i \in [|\mathcal{T}|]$, then $E(C)$ increases with the entropy of the instruction distribution $H(T)$, and the

cost maximizes at uniform distribution:

$$\lim_{|\mathcal{T}|\to\infty} E(C) = \frac{|\mathcal{T}|}{B}(1 - e^{-B/|\mathcal{T}|})c. \quad (5)$$

Remark 6. Here, p_i refers to the real-world occurrence probability of generic instruction $T_i, i \in [|\mathcal{T}|]$, not related to the sampling mechanism of noisy queries. Semantically equivalent generic instructions are treated as the same T_i.

Similarly, the cost reduction compared with non-batched scenario is associated with the entropy of instruction distribution (see Appendix D). The above theorem implies that our sharing strategy is especially cost-effective when the query distribution is uneven, such as in long-tail scenarios. Empirical studies have observed such non-uniform distributions in real-world LLM usage (Kelly and Harsel, 2025).

7 Experiment

7.1 Datasets and Setting

Our method is tested on three privacy-sensitive datasets: (i) Legal-QA that contains professional question–answer pairs covering real-world legal scenarios (dzunggg, 2023); (ii) Medical-QA with medical problems (Chen et al., 2024); and (iii) MMLU-Biz that extracts the business-related questions from MMLU dataset (Hendrycks et al., 2021).

We evaluate two variants of our framework: (i) original: all users transmit the raw prompt, and (ii) simplified: all users apply prompt simplification to improve efficiency. In practice, mixed deployments across users will likely yield utility and inference overhead between these two extremes.

7.2 Utility Analysis

We compare the utility of our algorithm with: (1) non-private setting where users issue prompts directly to the LLM, and (2) three model-agnostic private inference approaches (see Table 1), including: (i) RanText (Tong et al., 2025) that introduces random adjacency list for text perturbation; (ii) CusText (Chen et al., 2023) that provides relaxed DP guarantees by customizing each input token's replacement set; (iii) CusText+ that is adapted to provide similar level of relaxed DP of SharedRequest (see Appendix E.3); (iii) DP-Prompt (Utpala et al., 2023) that applies DP sampling during paraphrase generation via temperature-controlled decoding.

As shown in Table 2, the original version achieves nearly identical utility to the non-private baseline (within 1%), where any minor loss stems from instruction clustering and is negligible. The simplified version incurs an average utility loss of approximately 4.9% relative to the non-private setting.

	MMLU-Biz	Medical-QA	Legal-QA
		GPT-3.5-Turbo	
Non-private	**0.671**$\pm$**0.003**	6.89$\pm$0.01	7.31$\pm$0.01
Ours (Original)	0.665$\pm$0.002	**6.91**$\pm$**0.01**	**7.32**$\pm$**0.02**
Ours (Simplified)	0.638$\pm$0.002	6.32$\pm$0.04	7.02$\pm$0.05
		GPT-4o-mini	
Non-private	**0.853**$\pm$**0.001**	**8.60**$\pm$**0.01**	**8.69**$\pm$**0.03**
Ours (Original)	0.851$\pm$0.004	8.58$\pm$0.00	8.63$\pm$0.02
Ours (Simplified)	0.817$\pm$0.001	8.28$\pm$0.03	8.27$\pm$0.04
		GPT-4o	
Non-private	0.899$\pm$0.001	**8.81**$\pm$**0.03**	**8.81**$\pm$**0.01**
Ours (Original)	**0.900**$\pm$**0.002**	8.74$\pm$0.04	8.79$\pm$0.01
Ours (Simplified)	0.848$\pm$0.003	8.40$\pm$0.02	8.46$\pm$0.06

Table 2: Utility comparison of non-private setting and our approach. We report F1 score for MMLU-Biz, and quality score rated by GPT-4o on a 1–10 scale for Medical-QA and Legal-QA.

Table 3 compares the utility of our simplified variant against DP baselines under privacy budget $\epsilon = 1$. It can be observed that our method consistently outperforms the baselines. Specifically, our simplified version achieves 2.2×, 1.7×, 1.2× and 1.7× improved utility on average than RanText, CusText, CusText+ and DP-Prompt methods, respectively.

	GPT-4o-mini	GPT-4o
RanText	0.381$\pm$0.002	0.390$\pm$0.006
CusText	0.511$\pm$0.008	0.473$\pm$0.005
CusText+	0.686$\pm$0.002	0.694$\pm$0.003
DP-Prompt	0.497$\pm$0.010	0.496$\pm$0.008
Ours	**0.817**$\pm$**0.001**	**0.848**$\pm$**0.003**

Table 3: F1 score of our approach (with prompt simplification across all users) and DP baselines with privacy budget $\epsilon = 1$ on MMLU-Biz.

7.3 Attack Results

While the service provider only observes a shuffled set of prompts, it is crucial that the original queries remain indistinguishable from the injected noisy prompts. To assess the indistinguishability, we simulate the following inference attack: service provider utilizes a pre-trained classifier to label each prompt as genuine or noisy, aiming to distinguish true user queries. Based on the classification result, the server can identify the original prompts.

In Figure 2, we perform experiments using two Qwen-based discriminators, i.e., Qwen2.5-0.5B

and Qwen2.5-1.5B (Qwen et al., 2025), to generate candidate combinations. For each discriminator, we train four attack classifiers of varying model sizes (Liu et al., 2019; Qwen et al., 2025; Dubey et al., 2024) to distinguish genuine prompts from noisy variants. Depending on the sampled candidate set size m, the number of plausible combinations ranges from 8.3 to 52.4, and each attribute has over 7 alternatives on average for $\alpha \geq 10$. Among the qualified attributes, the F1 score of the attack is within 63% for Qwen2.5-0.5B and 58% for Qwen2.5-1.5B discriminators. In contrast, when the server randomly samples fake prompts without filtration, the attack success rate is much higher, reaching around 80%. Our combination-filtering mechanism reduces attack success by an average of approximately 32.7%, substantially improving robustness against server inference attacks.

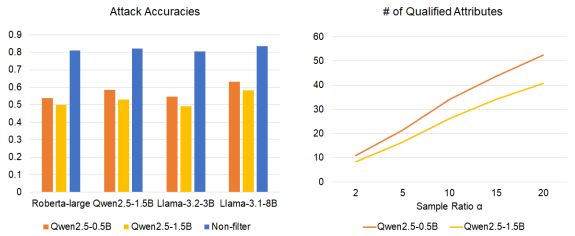

Figure 2: AUC of attack accuracies and size of qualified attributes on MMLU-Biz dataset using Qwen2.5-0.5B and Qwen2.5-1.5B discriminators. In the "non-filter" scenario, the server randomly samples fake prompts without applying combination filtering.

7.4 Query Cost

We compare monetary query cost of our protocol against a non-batched baseline, where noisy queries are not shared across users. To simulate real-world request distribution, we draw prompts from a Dirichlet distribution with concentration parameter β, where larger β reflects higher uniformity. As is shown in Figure 3, more concentrated distributions (smaller β) yield greater cost savings under our request sharing mechanism. At $\beta = 0.05$, our protocol reduces the monetary cost by up to $5.6\times$ compared to the non-batched setting. Prompt simplification further enhances this effect, since it increases collisions in generic instructions and amplifies batching efficiency.

7.5 Computation Cost for Attribute Filtering

Figure 4 reports the computation cost required for attribute combination filtering. Computation time increases roughly linearly with sample ratio α, staying below 2.1s for Qwen2.5-0.5B and 6.2s for Qwen2.5-1.5B when $\alpha \leq 20$. Overall, Qwen2.5-1.5B incurs about a $2.8\times$ overhead compared to Qwen2.5-0.5B, suggesting the trade-off between computation overhead and privacy protection. Prompt simplification further improves filtering efficiency, reducing runtime by an average of 40.8%, with larger efficiency gains at higher α.

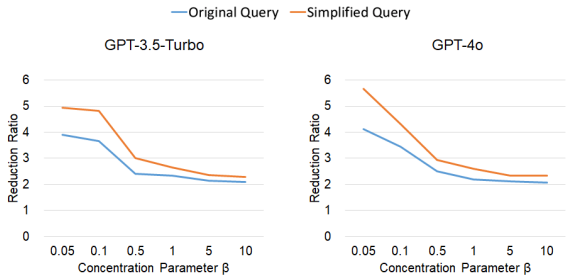

Figure 3: Query cost reduction under varying concentration parameters β for gpt models. Reduction ratio is measured as the ratio of non-batched baseline cost to that of SharedRequest.

Figure 4: Computation cost per generic instruction group (in seconds) for attribute filtering on MMLU-Biz using Qwen2.5-0.5B and Qwen2.5-1.5B discriminators.

8 Conclusion

This paper proposes SharedRequest, a privacy-preserving LLM inference paradigm that requires no architectural modifications to the model. Our multi-party protocol obscures user prompts by injecting noisy variants, where the additional inference costs are shared among a large batch of users. To alleviate user-side computation cost, we design a coordinated mechanism between users and noise sampler to sample qualified noisy queries. Theoretical analysis reveals that SharedRequest achieves greater query-cost reduction when request distributions are uneven, such as long-tailed. The empirical evaluation demonstrates that: (1) SharedRequest incurs minor utility degradation compared to non-private baseline, and outperforms DP baselines by over 20%. (2) Under concentrated distributions, SharedRequest reduces query costs by up to $5.6\times$ versus non-batched baselines.

Limitation

Our work has the following limitations and practical considerations.

Assumption of non-colluding parties. Our protocol assumes that the noise sampler can not collude with the service provider to ensure user privacy. In practice, the noise sampler can be: (1) a third party who provides the privacy-related service; or (2) an organization such as finance company that protects the confidentiality of employees' or clients' prompts. While two non-colluding parties is a common assumption in SMPC protocols (Corrigan-Gibbs and Boneh, 2017; Mohassel and Zhang, 2017), the security could be enhanced by increasing the colluding threshold. In Appendix C.1, we extend our protocol to multi-server setting, where an adversary needs to collude with all m noise samplers and the service provider to compromise privacy.

Private attributes and alternatives. We assume that users are responsible for defining their own privacy, i.e., identifying private attributes and generating alternatives for each attribute. This user-defined privacy paradigm is increasingly recognized in academic literature and implemented in practice (Van Alsenoy et al., 2014; Busch, 2019). Appendix F.1 discusses methods for the self-definition process, along with empirical results. In practice, users can add an anonymization layer prior to SharedRequest, removing information that is irrelevant to the task yet increases privacy risk. For example, they might might pattern-match and redact phone numbers or replace user names with generic placeholders like [NAME]. Then SharedRequest can be implemented to protect attributes that are sensitive while relevant for LLM inference.

Ethical Considerations

We emphasize that all datasets used in this work are publicly available benchmarks widely adopted in the NLP community for evaluation, such as standard language understanding and task benchmarks. These datasets are intended for research and model comparison and do not contain privately collected or user-generated content with personally identifying information. We also confirm that the data used in our experiments do not include offensive or sensitive content linked to identifiable individuals.

Acknowledgements

The authors utilized ChatGPT for polishing writings of the manuscript. All technical content, experimental design, analysis, and conclusions were developed and verified by the authors.

References

Christoph Busch. 2019. Implementing personalized law. *The University of Chicago Law Review*, 86(2):309–332.

Junying Chen, Zhenyang Cai, Ke Ji, Xidong Wang, Wanlong Liu, Rongsheng Wang, Jianye Hou, and Benyou Wang. 2024. Huatuogpt-o1, towards medical complex reasoning with llms. *Preprint*, arXiv:2412.18925.

Sai Chen, Fengran Mo, Yanhao Wang, Cen Chen, Jian-Yun Nie, Chengyu Wang, and Jamie Cui. 2023. A customized text sanitization mechanism with differential privacy. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 5747–5758.

Henry Corrigan-Gibbs and Dan Boneh. 2017. Prio: Private, robust, and scalable computation of aggregate statistics. In *14th USENIX symposium on networked systems design and implementation (NSDI 17)*, pages 259–282.

Minxin Du, Xiang Yue, Sherman SM Chow, Tianhao Wang, Chenyu Huang, and Huan Sun. 2023. Dp-forward: Fine-tuning and inference on language models with differential privacy in forward pass. *arXiv preprint arXiv:2309.06746*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Cynthia Dwork. 2006. Differential privacy. In *International colloquium on automata, languages, and programming*, pages 1–12. Springer.

dzunggg. 2023. legal-qa-v1: A collection of legal q&a pairs. Hugging Face Dataset. https://huggingface.co/datasets/dzunggg/legal-qa-v1.

Maud Ehrmann, Ahmed Hamdi, Elvys Linhares Pontes, Matteo Romanello, and Antoine Doucet. 2023. Named entity recognition and classification in historical documents: A survey. *ACM Computing Surveys*, 56(2):1–47.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, and 1 others. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231.

Meng Hao, Hongwei Li, Hanxiao Chen, Pengzhi Xing, Guowen Xu, and Tianwei Zhang. 2022. Iron: Private inference on transformers. *Advances in Neural Information Processing Systems*, 35:15718–15731.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, and 1 others. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Jonathan Katz. 2010. *Digital signatures*, volume 1. Springer.

Ravneet Kaur and Amandeep Kaur. 2012. Digital signature. In *2012 International Conference on Computing Sciences*, pages 295–301. IEEE.

Brenna Kelly and Luke Harsel. 2025. Investigating chatgpt search: Insights from 80 million clickstream records.

Tomasz J Kozubowski and Krzysztof Podgórski. 2000. A multivariate and asymmetric generalization of laplace distribution. *Computational Statistics*, 15(4):531–540.

Dacheng Li, Hongyi Wang, Rulin Shao, Han Guo, Eric Xing, and Hao Zhang. 2023. Mpcformer: Fast, performant and private transformer inference with mpc. In *The Eleventh International Conference on Learning Representations*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *ICLR (Poster)*.

Jinglong Luo, Yehong Zhang, Zhuo Zhang, Jiaqi Zhang, Xin Mu, Hui Wang, Yue Yu, and Zenglin Xu. 2024. Secformer: Fast and accurate privacy-preserving inference for transformer models via smpc. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 13333–13348.

Lingjuan Lyu, Xuanli He, and Yitong Li. 2020a. Differentially private representation for nlp: Formal guarantee and an empirical study on privacy and fairness. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2355–2365.

Lingjuan Lyu, Yitong Li, Xuanli He, and Tong Xiao. 2020b. Towards differentially private text representations. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1813–1816.

Peihua Mai, Ran Yan, Zhe Huang, Youjia Yang, and Yan Pang. 2024. Split-and-denoise: Protect large language model inference with local differential privacy. In *International Conference on Machine Learning*, pages 34281–34302. PMLR.

Justus Mattern, Benjamin Weggenmann, and Florian Kerschbaum. 2022. The limits of word level differential privacy. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 867–881.

Payman Mohassel and Yupeng Zhang. 2017. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE symposium on security and privacy (SP)*, pages 19–38. IEEE.

OpenAI. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Harshvardhan J Pandit, Beatriz Esteves, Georg Philip Krog, Paul Ryan, Delaram Golpayegani, and Julian Flake. 2024. Data privacy vocabulary (dpv)-version 2.0. In *ISWC (3)*.

Qi Pang, Jinhao Zhu, Helen Möllering, Wenting Zheng, and Thomas Schneider. 2024. Bolt: Privacy-preserving, accurate and efficient inference for transformers. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 4753–4771. IEEE.

Dan Pelleg and Andrew Moore. 1999. Accelerating exact k-means algorithms with geometric reasoning. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 277–281.

Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, and 24 others. 2025. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, page 3982. Association for Computational Linguistics.

Ronald L Rivest, Adi Shamir, and Leonard Adleman. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126.

Yu Rozanov. 2012. *Probability theory, random processes and mathematical statistics*, volume 344. Springer Science & Business Media.

Shubham Singh. 2025. Chatgpt statistics 2025.

Baris Sumengen, Anand Rajagopalan, Gui Citovsky, David Simcha, Olivier Bachem, Pradipta Mitra, Sam Blasiak, Mason Liang, and Sanjiv Kumar. 2021. Scaling hierarchical agglomerative clustering to billion-sized datasets. *arXiv preprint arXiv:2105.11653*.

Shun Takagi, Fumiyuki Kato, Yang Cao, and Masatoshi Yoshikawa. 2022. Asymmetric differential privacy. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 1576–1581. IEEE.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, and 1 others. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Meng Tong, Kejiang Chen, Jie Zhang, Yuang Qi, Weiming Zhang, Nenghai Yu, Tianwei Zhang, and Zhikun Zhang. 2025. Inferdpt: Privacy-preserving inference for black-box large language models. *IEEE Transactions on Dependable and Secure Computing*.

Saiteja Utpala, Sara Hooker, and Pin-Yu Chen. 2023. Locally differentially private document generation using zero shot prompting. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 8442–8457.

Brendan Van Alsenoy, Eleni Kosta, and Jos Dumortier. 2014. Privacy notices versus informational self-determination: Minding the gap. *International Review of Law, Computers & Technology*, 28(2):185–203.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19.

Andrew C Yao. 1982. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pages 160–164. IEEE.

Xiang Yue, Minxin Du, Tianhao Wang, Yaliang Li, Huan Sun, and Sherman SM Chow. 2021. Differential privacy for text analytics via natural text sanitization. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3853–3866. Association for Computational Linguistics (ACL).

Jiawen Zhang, Xinpeng Yang, Lipeng He, Kejia Chen, Wen-jie Lu, Yinghao Wang, Xiaoyang Hou, Jian Liu, Kui Ren, and Xiaohu Yang. 2024. Secure transformer inference made non-interactive. *Cryptology ePrint Archive*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623.

A Algorithm

Algorithm 1 and 2 outline our SharedRequest, consisting of forward and backward transmission.

B Sampling Size

Denote k_i as the number of candidates for the ith attribute, and μ is the number of private attributes within a group. The probability that an alternative attribute is not covered in a sample is:

$$(1 - 1/k_i)^m \,, \tag{6}$$

for attribute i.

Using union bound, the probability that each of the μ attributes has all candidates is bounded by:

$$\sum_{i}^{\mu} k_i \left(1 - 1/k_i\right)^m \leq \mu k \left(1 - 1/k\right)^m \leq 1 - p, \tag{7}$$

where $k = \max\{k_1, k_2, ..., k_\mu\}$. Taking the logarithm of both side, we have:

$$m \geq \frac{\log(1-p) - \log(\mu k)}{\log(1 - 1/k)}. \tag{8}$$

C Privacy Analysis

C.1 Resistant to Stronger Adversaries

Figure 5 presents the overview of our extended version.

C.1.1 Collusion between Servers

Our protocol assumes that the noise sampler can not collaborate with service provider. To relax the assumption, we extend our SharedRequest to a multi-server system with m noise samplers. In particular, the following steps in SharedRequest are modified:

- **Onion encryption:** User i onion encrypts the private attribute and seed with the servers public key:

$$OEnc_0^i = Enc(pk_n^1, Enc(...Enc(pk_s, s||A_{q_i}))), \tag{9}$$

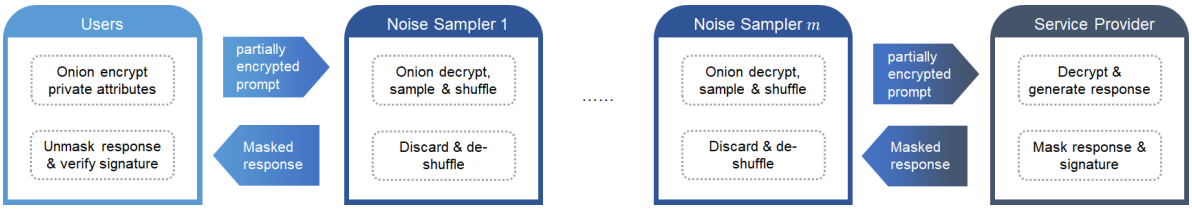

Figure 5: Overview of SharedRequest's extension. The requests are transmitted through m noise samplers. On receiving response, the user unmask the message and verify the signature generated by the service provider.

where pk_n^j is the public key for the jth noise sampler. Noise sampler j onion encrypts the private attribute within noisy prompt:

$$OEnc_j^i = Enc(pk_n^{j+1}, Enc(...Enc(pk_s, s||A_{q_i}))). \tag{10}$$

- **Onion decryption:** Noise sampler j onion decrypts the private attribute and seed with its private key:

$$OEnc_j^i = Dec(sk_n^j, OEnc_{j-1}^i), \tag{11}$$

where sk_n^j is the secret key for the jth noise sampler. The service provider decrypts with its private key sk_s:

$$(s, A_{q_i}) = Dec(sk_s, OEnc_m^i). \tag{12}$$

- **Noisy query sampling.** Each noise sampler generates qualified attribute combinations, and samples from the attributes added to the mix.

C.1.2 Malicious Noise Samplers

We consider the case where some noise samplers might temper with the transmitted message and return incorrect result to the user, including user queries and service provider's response. To ensure the integrity of message, the service provider creates signature for each query:

$$\sigma_i = Sign(sk_s^d, q_i||r_i), \tag{13}$$

where sk_s^d is the private signing key. Then the user can verify the correctness of the response by asserting that

$$Verify(pk_s^d, q_i||r_i, \sigma_i) = 1. \tag{14}$$

C.1.3 Security Analysis

We provide a high-level security analysis of the extended protocol. For secrecy, the views of service provider and noise sampler are similar to that in SharedRequest. The private attributes are onion encrypted through the public keys of m noise samplers and finally the service provider. An adversary would need to compromise all m samplers and the service provider to decrypt the attributes. If at least one noise sampler remains honest, the privacy of user attributes is preserved. Furthermore, each sampler independently shuffles and injects noisy variants to the batch, ensuring that the service provider's view is a mix of noisy and original prompts.

For integrity, the UF-CMA secure signature scheme (Kaur and Kaur, 2012; Katz, 2010) prevents the noise samplers from modifying user queries and service provider's responses. Consequently, users are able to identify that the output is incorrect and reject it.

C.2 Proof of Theorem 3

Proof. In our algorithm, modifying the private attributes within a prompt does not alter the batching outcome; it only affects the server's observed frequency of each attribute combination within a batch. Therefore, we can prove theorem 3 by showing that the frequency of the private attributes, H, satisfies $(\mathcal{A}^n, \epsilon)$-indistinguishability.

Changing Q to Q', i.e., changing the private attribute in one prompt, corresponds to decreasing one element in H by 1 and increasing another element in H by 1. Let $f : \mathcal{Q} \to \mathcal{H}$ be the function that maps the group of prompts to the frequency before injecting noisy query, and $M' : \mathcal{Q} \to \mathcal{H}$ be the mechanism that maps to the frequency after adding λ. Then we have:

Algorithm 1 SharedRequest: Forward Transmission

Input: User queries $Q = \{q_1, q_2, ..., q_N\}$.
Output: Responses $R = \{r_1, r_2, ..., r_S\}$, where response r_i corresponds to original or noisy query q_i.
User $i \in [N]$
• *Sample noisy attributes*: The user identifies private attributes A_{q_i}, and samples alternatives for each individual attributes $\mathcal{A}^i = \{\mathcal{A}^i_1, \mathcal{A}^i_2, ..., \mathcal{A}^i_{|A(q)|}\}$.
• *Encrypt attribute and seed*: The user encrypts the private attributes A_{q_i} and sampled seed s with service provider's public key, such that $Enc^i_s = Enc(pk_s, s||A_{q_i})$.
• *Encrypt and submit message*: The user encrypts the message with the noise sampler's public keys: $Enc^i_n = Enc(pk_n, T_{q_i}||\mathcal{A}^i||Enc^i_s)$, and sends the encrypted message Enc^i_n to noise sampler.

Noise Sampler n
• *Decrypt messages*: For every received cipher Enc^i_n, the sampler decrypts the cipher with its secret key sk_n, producing T_{q_i}, $\mathcal{A}^i$, and Enc^i_s.
• *Batch prompts*: The sampler groups prompts based on their generic instruction via a batching module $\mathcal{B}$, producing K groups $[G_1, G_2, ..., G_K]$ and their corresponding generic instruction $[T_{G_1}, T_{G_2}, ..., T_{G_k}]$.
• *Sample noisy query*: For each group G_k, the sampler: (i) unions all alternatives for each single attribute; (ii) samples and filters attribute combinations via a local discriminator, producing $\mathcal{A}^n(T_{G_k})$; (iii) randomly samples $\mathcal{N}[v]$ attribute combinations.
• *Encrypt attribute and seed*: Given the total count of dummy queries $S = \sum \mathcal{N}$, the sampler randomly generates S seeds. Then it encrypts the noisy attributes and seeds with the service provider's public key.
• *Shuffle and forward message*: The sampler shuffles all messages and sends them to the service provider s.

Service Provider s
• *Decrypt message*: For every received message Enc^i_s, the service provider decrypts the cipher with its secret key sk_n, producing s^i and A_{p^i}.
• *Generate response*: The service provider recovers each prompt by inserting the private attributes into the generic instruction $T_{G_k}, k \in [K]$. Then it groups the same prompts and generates the response for each group at once.

Algorithm 2 SharedRequest: Backward Transmission

Input: Responses for a set of queries (including original and noisy ones) $R = \{r_1, r_2, ..., r_S\}$.
Output: Responses $R = \{r_1, r_2, ..., r_N\}$, where response r_i corresponds to original query q_i.
Service Provider s
• *Handle and forward responses*: For each response r_i, the service provider generates random number from the corresponding seed $e^s_i = PRG(s_i)$, computes $r^i_s = r_i + e^s_i$, and sends them to noise sampler n.

Noise Sampler n
• *Handle responses*: On receiving the response from service provider s, the sampler de-shuffles them using the inverse permutation, and discards the response corresponding to the noisy queries.

• *Forward responses*: The sampler sends all responses r^s_i to the corresponding user i.

User $i \in [N]$
• *Reconstruct response*: The user receives response r^s_i from noise sampler n, generates random number from their seed $e^s_i = PRG(s_i)$, and reconstructs $r^i = r^i_i - e^s_i$.

$$\frac{p(M'(Q) = z)}{p(M'(Q') = z)} = \frac{\prod_i \exp\left(-\epsilon(z_i - f(Q)_i)/2\right)}{\prod_i \exp\left(-\epsilon(z_i - f(Q')_i)/2\right)} \leq \exp\left(\sum_i \epsilon|f(Q')_i - f(Q)_i|/2\right) \leq \exp(\epsilon). \tag{15}$$

Therefore, for any subset $S \in \mathcal{H}$, it holds that:

$$P(f(Q) \in S) \leq \exp(\epsilon) P(f(Q') \in S). \tag{16}$$

Taking the floor of λ is equivalent to taking the floor of frequency after adding λ. Then we have:

$$\begin{aligned} P(\lfloor f(Q) \rfloor = y) &= P\left(f(Q) \in floor^{-1}(z)\right) \\ &\leq \exp(\epsilon) P\left(\left(f(Q') \in floor^{-1}(z)\right)\right) \\ &= \exp(\epsilon) P(\lfloor f(Q') \rfloor = y), \end{aligned} \tag{17}$$

where $floor^{-1}$ denotes the inverse of floor function. Hence we complete the proof.

□

C.3 Preservation under Post-processing and Composition

Proposition 7 and 8 provides the privacy preservation under post-processing and composition.

Proposition 7. *Let $M : \mathcal{Q} \to \mathcal{Y}$ be a $(\mathcal{A}^n, \epsilon)$-indistinguishability mechanism. Let $f : \mathcal{Y} \to \mathcal{Z}$ be an arbitrary (possibly randomized) mapping. Then $f \circ M\mathcal{Q} \to \mathcal{Z}$ is $(\mathcal{A}^n, \epsilon)$-indistinguishability.*

Proof. Fix pair of neighboring input Q, $Q' \in N_{\mathcal{A}^n}(Q)$, and fix a a measurable set $S \subset \mathcal{Z}$. By the definition of f, we have:

$$\begin{aligned} &\Pr[f(M(Q)) \in S] \\ &= \int_y \Pr[f(y) \in S|y] \cdot \Pr[M(Q) = y]dy \\ &\leq \int_y \Pr[f(y) \in S|y] \cdot e^{\epsilon} \Pr[M(Q') = y]dy \\ &= e^{\epsilon} \Pr[f(M(Q')) \in S]. \end{aligned} \tag{18}$$

□

Proposition 8. *Let $M_i : \mathcal{Q} \to \mathcal{Y}_i$ be a $(\mathcal{A}^n, \epsilon_i)$-indistinguishability mechanism. Then if $\underline{M}$ is defined by:*

$$\underline{M}(Q) = (M_1(Q), M_2(Q), \cdots, M_k(Q)), \tag{19}$$

$\underline{M}$ satisfies $(\mathcal{A}^n, \epsilon)$-indistinguishability.

Proof. We start with the proof for $k = 2$. Let Q, $Q' \in N_{\mathcal{A}^n}(Q)$ be the $\mathcal{A}^n$-neighbors, and let $S = S_1 \times S_2$ be any measurable set in the product output space. As M_1, M_1 have independent randomness, we have:

$$\begin{aligned} &\Pr[(M_1(Q), M_2(Q)) \in S] \\ &= \Pr[M_1(Q) \in S_1] \cdot \Pr[M_2(Q) \in S_2] \\ &\leq e^{\epsilon_1} \Pr[M_1(Q') \in S_1] \cdot e^{\epsilon_2} \Pr[M_2(Q') \in S_2] \\ &= e^{\epsilon_1 + \epsilon_2} \Pr[(M_1(Q'), M_2(Q')) \in S]. \end{aligned} \tag{20}$$

The result can extend inductively to any $k > 2$. □

D Query Cost Analysis

D.1 Proof of Theorem 5

Proof. For each instruction group, it appears in the batch for at least one time with probability $1 - (1 - p_i)^B$. Then the expected per-query cost is as follows:

$$E(C) = \frac{1}{B} \sum_{i=1}^{|\mathcal{T}|} \left(1 - (1 - p_i)^B\right) c_i. \tag{21}$$

Thus we focus on the proof for the remaining part under $c_i = c$ for all generic instructions. To prove the relationship between $E(C)$ and $H(T)$, we start with a basic case: p_i increases with δ, and p_j decreases with δ for any pair p_i, p_j.

For $E(C)$, its derivative with regards to δ is given by:

$$\begin{aligned} &\frac{\nabla E(C)}{\nabla \delta} = \frac{\nabla}{\nabla \delta} \\ &\left(\frac{c}{B}\left(\left(1 - (1 - p_i - \delta)^B\right) + \left(1 - (1 - p_i + \delta)^B\right)\right)\right) \\ &= c\left((1 - p_i - \delta)^{B-1} - (1 - p_j + \delta)^{B-1}\right). \end{aligned} \tag{22}$$

Then $E(C)$ increases in δ if $\delta \leq (p_j - p_i)/2$, and decreases in δ otherwise.

For $H(T)$, it derivative is as follows:

$$\begin{aligned} &\frac{\nabla H(T)}{\nabla \delta} = \frac{\nabla}{\nabla \delta} \\ &((p_i + \delta)\log(p_i + \delta) + (p_j - \delta)\log(p_j - \delta)) \\ &= \log\left(\frac{p_i + \delta}{p_j - \delta}\right). \end{aligned} \tag{23}$$

$H(T)$ also increases in δ if $\delta \leq (p_j - p_i)/2$, and decreases in δ otherwise. Therefore, the change in δ leads to the same direction of change for $E(C)$ and $H(T)$.

We can generalize the above case to any change from $\mathbf{p} = (p_1, ..., p_{|\mathcal{T}|})$ to $\mathbf{p}' = (p'_1, ..., p'_{|\mathcal{T}|})$, which can be achieved through $|\mathcal{T}| - 1$ steps of the following change (for the ith steps):

- Add $\delta = (p'_i - p_i + p'_{i-1} - p_{i-1} + \cdots + p'_1 - p_1)$ to p_i.
- Subtract the same δ from p_{i+1}.

For each step, the direction of change for $E(C)$ and $H(T)$ is the same, and thus $E(C)$ and $H(T)$ both increase or decrease with regards to the instruction distribution $\mathbf{p}$.

Given that $E(C)$ is positively association with $H(T)$, it maximize at the uniform distribution:

$$E(C) = \frac{1}{B} \sum_{i=1}^{|\mathcal{T}|} \left(1 - (1 - 1/|\mathcal{T}|)^B\right) c. \tag{24}$$

Taking the limit:

$$\lim_{|\mathcal{T}| \to \infty} E(C) = \frac{|\mathcal{T}|}{B}(1 - e^{-B/|\mathcal{T}|})c. \tag{25}$$

□

D.2 Comparison with Non-batched Scenario

For non-batched scenario, we consider the simple construction where each user sends the noisy and original prompts to the server individually, without the noise sampler for aggregated sampling and shuffling. Denote c'_j and p'_j as, respectively, the average query cost and sampling probability for query $q_j, j \in [|\mathcal{Q}|]$. The cost reduction ratio can be formulated as:

$$\frac{E(C)}{E(C')} = \frac{\sum_{i=1}^{|\mathcal{T}|} \left(1 - (1 - p_i)^B\right) c_i}{B \sum_{j=1}^{|\mathcal{Q}|} p'_j c'_j}, \tag{26}$$

where $E(C')$ is the expected per-query cost for non-batched scenario.

Noted that c_i and c'_j is determined by the token length and number of qualified alternatives for each query or instruction group. For simplicity, we assume that $c_i = c'_j = c$ for all $i \in [|\mathcal{T}|], j \in [|\mathcal{Q}|]$. This is reasonable since the per-instruction overhead in our SharedRequest is similar to the average per-query overhead of the non-batched baseline. Then we have:

$$\frac{E(C)}{E(C')} = \frac{\sum_{i=1}^{|\mathcal{T}|} \left(1 - (1 - p_i)^B\right)}{B} = \frac{E(C)}{c}. \tag{27}$$

Therefore, the reduction ratio increases with $E(C)$ and $H(T)$, maximizing at the uniform distribution:

$$\lim_{|\mathcal{T}| \to \infty} \frac{E(C)}{E(C')} = \frac{|\mathcal{T}|}{B}(1 - e^{-B/|\mathcal{T}|}). \tag{28}$$

E Experiment

E.1 Experiment Setting

For prompt grouping, we use paraphrase-distilroberta-base-v1 model (Reimers and Gurevych, 2019) to convert generic instruction into embeddings, and leverage RAC (Sumengen et al., 2021) to cluster these embeddings. We set sampling ratio $\alpha = 10$, privacy parameter $\epsilon = 1$, and batch size $B = 5000$ unless specified. We employ RSA cryptosystem (Rivest et al., 1978) for asymmetric encryption. The experiments are conducted on a 96-core Ubuntu Linux 20.04 server with 128GB RAM and 2 A100 driver. Each reported experiment result is an average of 3 experiments.

E.1.1 Prompt Simplification

To train simplification model, We finetune Qwen3-4B (Yang et al., 2025) for 5 epochs with a learning rate of 0.00001 using AdamW optimizer (Loshchilov and Hutter, 2019), using demonstrations generated by GPT-4o. Specifically, we ask GPT-4o to simplify a question from the three datasets following the rules:

- Remove all unnecessary or redundant information.
- Keep all important information necessary to answer the question.
- Do not use abbreviations or emojis.
- Compress the origin as short as you can.

The initial version of simplification demonstrations may lose important information. To ensure quality, we iteratively refine the dataset. If the simplified version yields lower response quality compared to the original, we prompt GPT-4o with the original prompt, initial simplification, and response evaluation to generate an improved version. The refinement refinement loop runs for 5 rounds.

For simplified version of SharedRequest, we finetune the simplification model on 80% of the constructed dataset, and implement the simplification on the remaining 20% for evaluation.

E.1.2 Cluster Algorithm

RAC (Sumengen et al., 2021) is employed to cluster the sentence embeddings during prompt grouping. Clustering is based on Euclidean distance between embeddings, with a merging threshold of 0.3, and executed using 8 parallel threads. After clustering, a single instruction is randomly sampled from each cluster to serve as the shared generic instruction for that group.

E.1.3 Output Arrangement

To prevent the noise sampler or any external observer from inferring private information via side channels in response lengths and timings, the service provider pads all outputs within a cluster to the maximum length observed in that cluster using masked values. Masked responses from each cluster are then delivered to the noise sampler at a constant rate of 20 tokens/s.

E.1.4 Discrimination Model

We tested two Qwen-based discriminators, i.e., Qwen2.5-0.5B and Qwen2.5-1.5B (Qwen et al., 2025). We input the noisy and original queries, as well as their binary labels, to train the discriminator for 5 epochs with a learning rate of 0.00001

using AdamW optimizer, with balanced batches containing noise and original queries in a 1:1 ratio. We maintain strict separation between the datasets used for training the discriminator and those used for attack models to prevent any data leakage.

E.1.5 Utility Evaluation Metrics

MMLU-Biz, which consists of multiple-choice questions, is evaluated using F1 score. Medical-QA and Legal-QA, which comprise of open-ended questions, are scored from 1 to 10 using GPT-4o as an automatic judge, following (Zheng et al., 2023).

E.1.6 Latency Analysis

We focus on the active processing time for latency analysis. We assume the noise sampler parallelizes API calls and cryptographic operations across each prompt, and parallelizes attribute combination filtering across each generic instruction. As we cannot directly deploy proprietary GPT models, we estimate the LLM inference time using the average query time per prompt without privacy protection. We use Qwen2.5-0.5B for combination filtering.

E.2 Data Processing

E.2.1 Dataset Construction

The MMLU-Biz dataset is constructed by filtering questions from the business or finance-related categories: business ethics, econometrics, marketing, high school macroeconomics, high school microeconomics, management, and marketing. The final MMLU-Biz dataset contains 1534 samples. To ensure equal representation across domains (business, legal, and medical), we then randomly sample 2,000 items each from the Legal-QA and Medical-QA datasets. The final evaluation set therefore consists of balanced sample sizes across the three domains. In experiment, we sample the items from Dirichlet distribution with concentration parameter $\beta = 1$.

E.2.2 Private Attributes

The private attributes for each sample are labeled with GPT-4o using the prompt in Figure 6. The GPT-extracted attributes are then manually checked and corrected to ensure quality.

E.2.3 Alternative Attributes

To sample alternatives for each single attributes, we prompt GPT-4o to generate candidates using the instruction in Figure 7. Each attribute has around 10 candidates on average.

E.3 Comparison with DP Baselines

To provide similar privacy guarantee with the relax DP version of SharedRequest, we adapt CusText+ by replacing only the private attributes from the user-defined candidate lists using exponential sampling mechanism. Then the algorithm satisfies ϵ-DP with similar neighborhood of SharedRequest. The full results under ϵ from 1 to 5 is presented in Table 4. Even under the highest privacy budget, our method consistently outperforms the baselines.

Method	ϵ	GPT-4o-mini	GPT-4o
RanText	1	$0.381_{\pm 0.002}$	$0.390_{\pm 0.006}$
	3	$0.388_{\pm 0.006}$	$0.405_{\pm 0.008}$
	5	$0.411_{\pm 0.005}$	$0.435_{\pm 0.006}$
CusText	1	$0.511_{\pm 0.008}$	$0.473_{\pm 0.005}$
	3	$0.553_{\pm 0.002}$	$0.603_{\pm 0.003}$
	5	$0.631_{\pm 0.006}$	$0.649_{\pm 0.005}$
CusText+	1	$0.686_{\pm 0.002}$	$0.694_{\pm 0.003}$
	3	$0.695_{\pm 0.004}$	$0.758_{\pm 0.006}$
	5	$0.743_{\pm 0.005}$	$0.754_{\pm 0.005}$
DP-Prompt	1	$0.497_{\pm 0.010}$	$0.496_{\pm 0.008}$
	3	$0.522_{\pm 0.009}$	$0.533_{\pm 0.006}$
	5	$0.529_{\pm 0.003}$	$0.542_{\pm 0.003}$
Ours	1	$\mathbf{0.817}_{\pm 0.001}$	$\mathbf{0.848}_{\pm 0.003}$

Table 4: F1 score of our approach (with prompt simplification across all users) and DP baselines across privacy budget $\epsilon = 1$ to 5 on MMLU-Biz.

E.4 Computation Cost of Combination Filtering

Figure 8 presents the computation time for attribute combination filtering across datasets. Legal-QA incurs the highest computation cost due to its longer questions and more private attributes, under 6s for Qwen2.5-0.5B and 15s for Qwen2.5-1.5B for $\alpha \leq 20$. Under $\alpha \leq 10$, the computation cost stays within 3s for Qwen2.5-0.5B and 6s for Qwen2.5-1.5B. Prompt simplification consistently reduces the computation cost by an average of 32.7%, 63.1%, and 53.3% for MMLU-Biz, Legal-QA, and Medical-QA, respectively.

E.5 Statistics Analysis of Private Attributes

We summarize the statistics of private attributes and their noisy alternatives in Table 5. We can observe that Legal-QA and Medical-QA have more private attributes within each prompt, leading to higher number of noisy combinations. Furthermore, our filtering algorithm ensures that each single attribute has over 8 alternatives on average.

User Prompt:
Please extract all words or phrases in the question that indicate sensitive attributes.
The sensitive attributes include: Personal identifiable information, job, race, religion, ethnicity, religion, beliefs, age, contact information, geolocation data, residency and citizenship status, date, festival, sexual orientation or practices, gender identity, third-person pronoun, disability, employment and income details, company, physical health and mental health issues, clinical signs including medical conditions and physical examination data, behavioral data, name of medicine, including specific drugs, drug classes, and general medication types (e.g., nasal decongestants, antihistamines, beta-blockers, etc), financial data, vulnerable financial status, account details, loan information, tax records, contract, trading algorithm, confidential business information, educational records, voting status, membership in a trade union, social media and digital footprint, legal consultation and case status, Law enforcement jurisdiction and procedures, consumer rights, ownership details, criminal record or name, biometric data, genetic information, legal proceedings and disputes, surveillance and monitoring data, military and security clearance.
Your task is to strictly extract phrases that refer to these sensitive attributes. Even if a term indirectly refers to a sensitive attribute (e.g., a drug category instead of a specific medicine name), it should be included in the list. The list of sensitive attributes provided is not exhaustive — if you encounter any word or phrase that could reasonably be considered sensitive information under privacy laws or common data protection standards (like GDPR, HIPAA, or similar frameworks), include it in the list. Prioritize any data that could identify a person, describe their personal circumstances, or reveal confidential, medical, financial, or legal information.
Please preserve the original words from the user's question to form the attribute list and do not convert full names to abbreviations or abbreviations to full names, and do not add any extra words. Retain all duplicates, regardless of form variations (e.g., burglarize, burglarized, burglarizing).
Please strictly return a list of phrases in the format of ["attribute 1", "attribute 2", ..., "attribute n"]. If there is no sensitive attribute, return an empty list []. Try to identify as much as phrases as possible.
Question: {*question*}
Sensitive attributes:

Figure 6: The GPT-4o instructions used for private attribute extraction.

	# of Private Attributes	# of Noisy Combinations	# of Alternatives for Single Attribute
		MMLU-Biz	
Original	3.4	49.4	9.1
Simplified	2.3	34.2	8.6
		Legal-QA	
Original	10.7	118.7	11.1
Simplified	5.6	100.2	11.8
		Medical-QA	
Original	7.9	62.4	9.4
Simplified	5.4	52.4	8.7

Table 5: Statistics of private attribute for each dataset. The values are summarized as the average over all samples, using Qwen2.5-0.5B discriminator under sample ratio $\alpha = 10$.

E.6 Query Cost

We examine the query cost under privacy parameter ϵ ranging from 0.01 to 10 in Figure 9. While the cost decreases as ϵ increases (i.e., privacy is weakened), the query cost and thus reduction ratio retain stable under $\epsilon \leq 0.1$. This happens because the service provider aggregates and de-duplicates identical prompt variants for LLM inference, and thus the query cost is upper bounded by the size of qualified attribute combinations.

Figure 10 presents the query cost under batch size B ranging from 100 to 5000. Smaller batch size leads to lower level of reduction ratio, since the cost is distributed among fewer queries. At $B = 5000$, the reduction ratio ranges from $2\times$ to $5.6\times$, whereas at $B = 100$, the ratio drops to around $1.6\times$ to $2.1\times$. Prompt simplification consistently reduces the query cost across all batch configurations.

E.7 Latency Analysis

The latency can be decomposed into two parts: the idle waiting period used to gather prompts from multiple clients, and the active processing time required to handle the collected batch. The former component depends on the configured window time. We focus on the latter component, which comprises: (i) LLM inference time on the service provider; (ii) attribute combination filtering time for the noise sampler; (iii) prompt grouping time for the noise sampler; (iv) optional user-side query simplification time; (v) remaining overhead, mainly encryption and decryption operations. Components (ii)–(iv) represent the additional latency introduced by SharedRequest's privacy-preserving protocol.

Figure 11 breaks down active batch processing latency under parallelized implementation. It can be observed that attribute combination filtering adds the largest overhead beyond the server-side LLM inference. While prompt simplification significantly reduces downstream processing time, it introduces an average 1.4s of user-side overhead, illustrating a trade-off between local computation

User Prompt:
You are an AI designed to generate accurate, contextually relevant, and trustworthy fake sensitive attributes. Your task is to generate one list of fake sensitive attributes for each original attribute provided in filtered private attributes based on the context of the 'question'. The generated attributes must meet the following criteria:
1. Numerical Handling:
- If the original attribute is primarily numerical (e.g., percentages, durations, counts), generate fake numerical attributes that have broader variance to reduce guessability.
- Avoid clustering numbers closely around the original value.
- Avoid adding unnecessary descriptive context (e.g., avoid turning "50%" into "55% of alternative components"). Instead, generate standalone numerical fakes like "20%", "90%", or "10%".
- Do not introduce numerical values if the original attribute is non-numerical in any form. (e.g. aviod turning "internship period" into "3 months")
2. Length Consistency: Ensure that the generated fake attributes have a similar length (in words or characters) to the original attribute to maintain fluency and reduce guessability.
3. Sound Sensitive and Relaiable:
- The fake attributes should resemble private, confidential, or sensitive concepts, particularly in legal, medical, or financial domains."
- Generated fake attributes actually exist in real-world literature.
4. No Rephrasings or Simple Synonyms: Avoid generating superficial rephrasings or synonyms (e.g., do not turn "80 years" into "eight decades"):
- Create attributes that keep contextual depth and believability.
- Do not generate fakes by slightly modifying words.
- The genearted word should under the same category of the original attribute but with strictly distinct meanings.
5. Diversity and Independence:
- Generate independent fake attributes for each prompt and do not reference or rely on fake attributes generated for the other prompt. Avoid generating fake attributes that are too similar to each other.
- Ensure that the fake attributes are diverse and do not share common themes or patterns.
- if the input private attributes are dependent based on the context, generate fake attributes that are dependent on the context as well.
6. Contextual Fluency:
- Make sure that the fake attributes fit naturally and coherently within the sentence structure for each context.
The list must be structured as follows:
- The next five attributes must be unique, fake attributes.
- Output format for each list is: [["fake attribute 1_1", ..., "fake attribute 1_n"],...]
The input is structured as follows:
Private Attributes: {*private attributes*}
Question: {*question*}
Fake attributes for question:

Figure 7: The GPT-4o instructions used for alternative attribute generation.

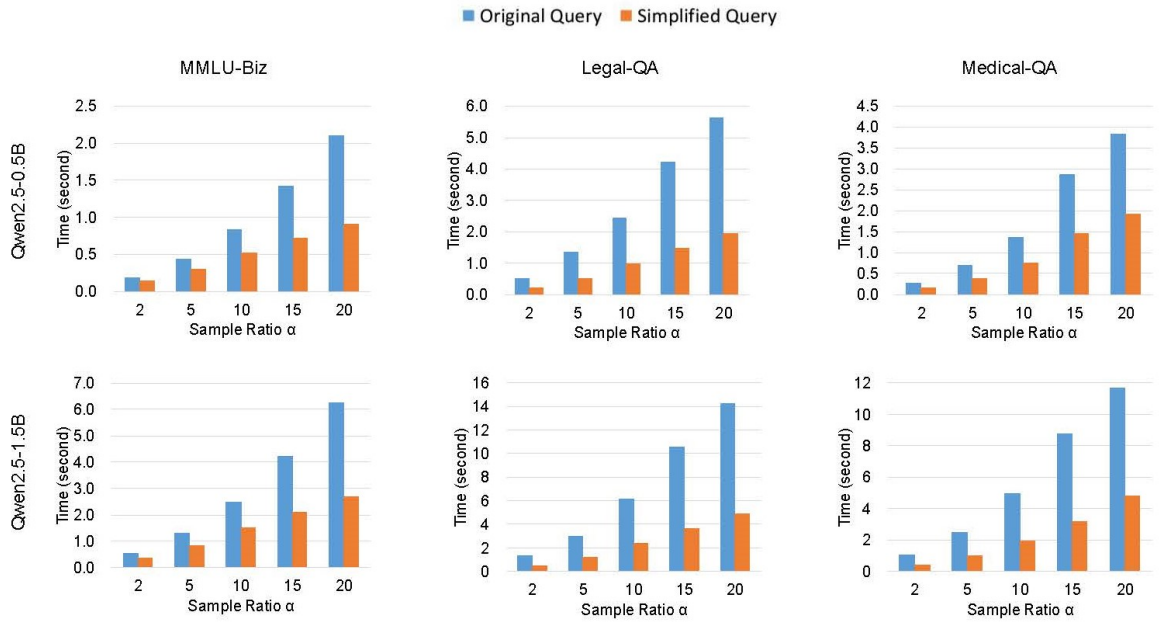

Figure 8: Computation cost per generic instruction group (in seconds) for attribute filtering on three datasets using Qwen2.5-0.5B and Qwen2.5-1.5B discriminators.

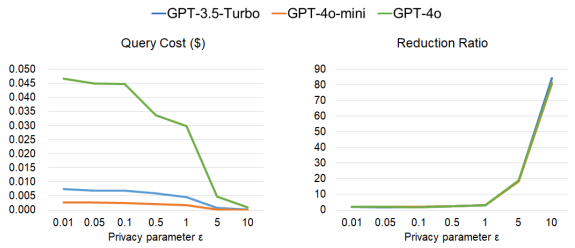

Figure 9: Per-request query cost (in dollars) and reduction ratio under varying privacy parameter ϵ.

and subsequent latency. In contrast, the overheads for cryptographic operations and prompt grouping are minimal, accounting for only 0.6% and 2% of total latency on average, respectively.

E.8 Impact of Clustering Parameter

We study the impact of merging threshold on query costs and accuracies in Figure 12 and 13. As the merging threshold increases, more generic instructions are grouped into fewer clusters, which reduces both accuracy and query cost. The optimal threshold should be chosen based on the desired trade-off between utility and overhead.

E.9 Overhead Comparison

We compare the computation and communication overhead of SharedRequest with existing LLM inference frameworks. We evaluate overhead against two method types: (1) SMPC-based methods (Iron, MPCFormer, BOLT, and NEXUS) and (2) DP-based methods (RanText, CusText and DP-Prompt).

Table 6 summarizes the resource requirements and overhead of these inference methods. SMPC-based approaches, due to their multiparty computation protocols, incur significantly higher communication overhead than both SharedRequest and DP-based methods. Moreover, despite running on smaller models (i.e., Bert-base), SMPC-based methods exhibit higher inference time compared to SharedRequest, which operates on GPT-4o. To our best knowledge, existing SMPC-based techniques have only been evaluated on models up to 13B parameters, and their scalability to larger models (e.g., 70B+) remains an open challenge. While SharedRequest has a higher runtime than perturbation-based methods, its substantial utility gains highlight its advantage.

F Discussions

F.1 Private Attributes

Users could identify private attributes and their alternative through:

- **Named-entity recognition (NER).** Deploy standard NER tools (Ehrmann et al., 2023) to identify private information such as dates and locations.

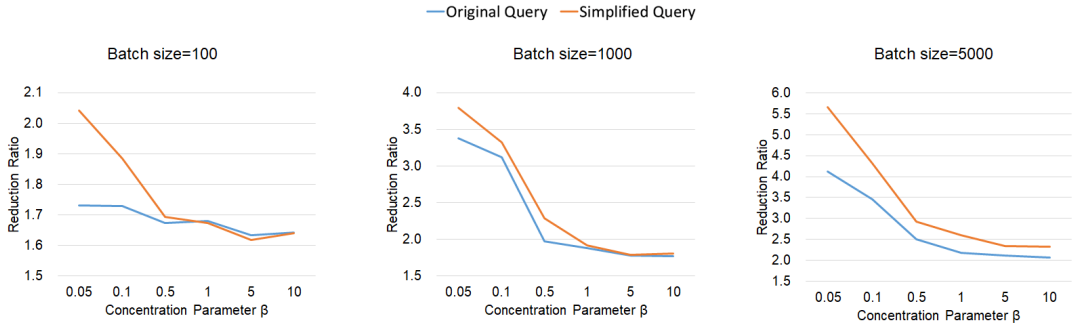

Figure 10: Query cost reduction ratio under varying batch sizes B.

	Framework	Model	I./O. size	Communication cost	Communication setting	Runtime
SMPC	Iron	Bert-base	(128, 1)	280 GB	(3 Gbps, 0.8 ms)	~475 s
	MPCFormer	Bert-base	(128, 1)	12 GB	(5 Gbps, 1 ms)	~55 s
	BOLT	Bert-base	(128, 1)	25 GB	(3 Gbps, 0.8 ms)	~185 s
	NEXUS	Bert-base	(128, 1)	0.16 GB	(100 Mbps, 80 ms)	~55 s
DP	RanText	GPT-4o	(512, $*$)	$<$ 0.1 MB	(100 Mbps, 80 ms)	~7 s
	CusText	GPT-4o	(512, $*$)	$<$ 0.1 MB	(100 Mbps, 80 ms)	~6.5 s
	DP-Prompt	GPT-4o	(512, $*$)	$<$ 0.1 MB	(100 Mbps, 80 ms)	~7 s
Ours	**SharedRequest**	GPT-4o	(512, $*$)	$<$ 0.1 MB	(100 Mbps, 80 ms)	~10 s

Table 6: Resource requirement, communication cost, and runtime of privacy-preserving LLM inference frameworks. Values in I./O. size refers to the lengths of input and output tokens, where $*$ denotes an unfixed length. Communication setting specifies the network's bandwidth and latency parameters.

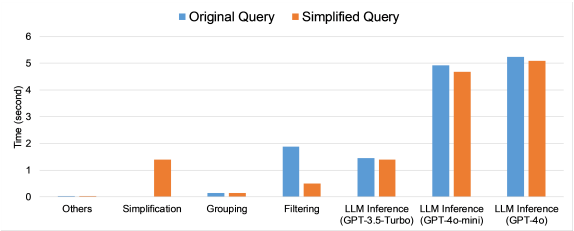

Figure 11: Computation time (in seconds) per query for each process. Simplification refers to local query simplification, grouping refers to prompt grouping, and filtering refers to filtering of qualified attribute combinations.

Figure 12: F1 score under varying merging threshold on MMLU-Biz.

Figure 13: Query cost reduction ratio under varying merging threshold on MMLU-Biz.

- **Pre-constructed attribute database.** Maintain a user-curated mapping from private categories to keywords (e.g. "job": ["cybersecurity engineer", "attorney",...]). Ontologies like DPV-PD offer structured category definitions and synonym support for this purpose (Pandit et al., 2024). The dictionary can be used to match private attributes in the keyword, and the alternatives can be chosen from keywords of each category.
- **Local classifier.** Train a token-level or span-level model on annotated datasets or token-annotation lexicons to predict whether individual tokens or phrases are sensitive and their privacy categories.
- **Mask-and-fill via online LLMs.** Replace

sensitive spans in the prompt with #MASK and request an online LLM to propose plausible substitutes. The validated outputs can be added to the alternative set for A_{q_i}.

F.2 Collision of Generic Instruction

In some cases, user prompts are highly specific or unique, resulting in very small clusters and limited queries for cost sharing. However, the cost for such extreme cases can be smoothed out by other common instructions on average. Both our theoretical analysis and empirical results show that the overall reduction is positively correlated with the entropy of the request distribution. In other words, the more skewed (long-tailed) the distribution, the greater the expected amortization benefit, because a larger proportion of prompts have semantically equivalent generic instruction.

To reduce per-prompt cost even in those extreme cases, we can adaptively adjust the batch window per generic instruction, waiting for sufficient queries before forwarding to the service provider. This dynamic batching introduces a natural latency–cost trade-off: longer waiting times enable higher reduction ratios at the expense of increased delay.

Furthermore, the noise sampler can support users proactively by estimating the likely cluster size for a given generic instruction based on historical data. The system can warn users when the expected "collision" count is low. The user can then refine or generalize their instruction based on the sampler's suggestions to increase cluster size.

F.3 Query Deduplication

Our framework's cost analysis implicitly assumes that the service provider can deduplicate identical prompts at inference time and charge only once per unique evaluation. While such deduplication is technically feasible and implementable within a provider's internal inference stack, current commercial LLM APIs might not generally perform this kind of deduplication by default.

To address the gap between our cost model and real-world API billing, we propose incorporating an additional non-colluding deduplicator between the noise sampler and service provider:

- **Encrypted Prompt Deduplication:** users encrypt their private attributes using the deduplicator's public key. The deduplicator decrypts the private attributes to identify identical prompt for deduplication.
- **Service Provider Inference:** The deduplicated prompt set is sent to the service provider, which processes each unique prompt once and returns the corresponding outputs.
- **Response Expansion and Masking:** Upon receiving the responses, the deduplicator applies masking and then expands the responses back to match each user's original query set. These masked responses are then forwarded to the noise sampler while preserving privacy.

Under this design, the deduplicator's view consists of a mixed set of original and noisy prompts that satisfies $(\mathcal{A}^n, \epsilon)$-indistinguishabilit, and by the post-processing property (see Proposition 7), the service provider's view achieves the same privacy guarantee. According to our empirical evaluation, incorporating the deduplicator incurs within 5% additional computation cost under our experimental setting, where deduplication is parallelized across batches. This allows the billing benefits of deduplication to be realized even under pricing models that do not natively support per-prompt deduplication, while still maintaining the privacy guarantees of our framework.