# FastQuery: Communication-efficient Embedding Table Query for Private LLMs inference

Chenqi Lin[1†], Tianshi Xu[1†], Zebin Yang[1†], Runsheng Wang[134], Ru Huang[134]and Meng Li[213*]
[1]School of Integrated Circuits & [2]Institute for Artificial Intelligence, Peking University, China
[3]Beijing Advanced Innovation Center for Integrated Circuits, Beijing, China
[4]Institute of Electronic Design Automation, Peking University, Wuxi, China
[†]Equal contribution, [*]Corresponding author: meng.li@pku.edu.cn

## ABSTRACT

With the fast evolution of large language models (LLMs), privacy concerns with user queries arise as they may contain sensitive information. Private inference based on homomorphic encryption (HE) has been proposed to protect user query privacy. However, private embedding table query has to be formulated as a HE-based matrix-vector multiplication problem and suffers from enormous computation and communication overhead. We observe the overhead mainly comes from the neglect of 1) the one-hot nature of user queries and 2) the robustness of the embedding table to low bit-width quantization noise. Hence, in this paper, we propose a private embedding table query optimization framework, dubbed FastQuery. FastQuery features a communication-aware embedding table quantization algorithm and a one-hot-aware dense packing algorithm to simultaneously reduce both the computation and communication costs. Compared to prior-art HE-based frameworks, e.g., Cheetah, Iron, and Bumblebee, FastQuery achieves more than 4.3×, 2.7×, 1.3× latency reduction, respectively and more than 75.7×, 60.2×, 20.2× communication reduction, respectively, on both LLAMA-7B and LLAMA-30B.

## 1 INTRODUCTION

Large language models (LLMs), such as ChatGPT [1] and LLAMA [20], have demonstrated state-of-the-art performance in a wide range of tasks and get increasingly adopted in sensitive and private applications, such as personal assistants, medical diagnosis, etc. However, to leverage LLMs on the cloud, users are required to disclose their queries directly to the service provider, which may contain sensitive information. Privacy has thus emerged as a major concern [18].

To address these privacy concerns, secure two-party computation (2PC) based on homomorphic encryption (HE) has been proposed recently as a promising solution [5, 7]. Besides enabling accurate LLM inference, it also protects the privacy of both user data and LLM model parameters with a formal privacy guarantee [5, 6, 11, 18].

However, the privacy protection of HE-based 2PC is achieved at the cost of high communication overhead. This is due to the transfer of a large number of high bit-width homomorphically encrypted ciphertexts (e.g., 109-bit ciphertext for Cheetah [7]) between the server and the client. Previous works, such as Iron [5], CipherGPT [6], and Bumblebee [11], etc, have focused on optimizing the HE-based Transformer computation. However, they overlook the private embedding table query, which we have found to be more time-consuming and communication-intensive compared to a transformer block. As depicted in Figure 1, while the cost of the embedding table query is negligible in plaintext, it incurs significantly higher latency in ciphertext compared to the QKV projection. This is because to protect the user query privacy, embedding table query has to be realized with a HE-based matrix-vector multiplication [11]. Considering the vocabulary size of LLM is very large, e.g., 32000 for LLAMA-7B, the communication overhead is massive and thus the latency becomes substantial.

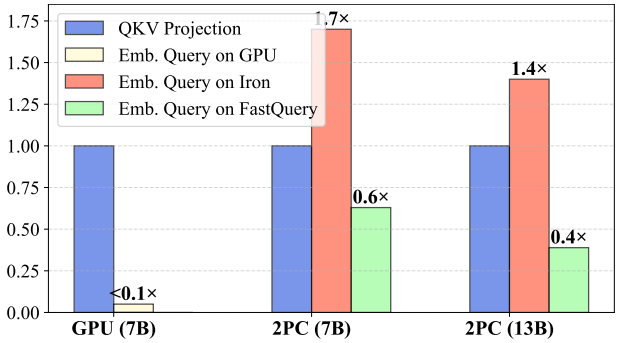


**Figure 1: Compare the latency of QKV projection with embedding table query on GPU and prior-art 2PC framework Iron as well as our 2PC framework FastQuery on LLAMA-7B and 13B. The data is normalized where the QKV projection is** 1.0**.**

To enable efficient private embedding table query, we observe previous works ignore the key characteristics of embedding table. On the one hand, the query is a one-hot vector since each time only one token from the vocabulary is picked. On the other hand, the embedding table is robust against low bit-width quantization noise. For example, we empirically find directly quantizing the embedding table to 6 bits introduces negligible perplexity degradation (e.g., 7.404 vs 7.340 for LLAMA-7B). However, all existing works assume high-precision plaintext for the embedding table [5, 7, 11], leading to a significant waste of communication. To this end, we propose FastQuery, a communication-efficient embedding table query protocol. By leveraging the one-hot nature of user queries and a low bit-width quantized embedding table, FastQuery significantly reduces the communication overhead. The contributions of this work can be summarized as follows:

- We identify that existing HE-based 2PC frameworks overlook the optimization opportunities for embedding table queries and propose FastQuery, a communication-efficient protocol customized for private embedding table query.
- We introduce a communication-aware embedding table quantization algorithm to reduce the bit-widths of ciphertexts. Additionally, we develop a novel HE-based query protocol that utilizes the one-hot nature of user queries, further reducing communication overhead.
- FastQuery outperforms prior-art HE-based 2PC frameworks, including Cheetah, Iron, and Bumblebee. On LLAMA-7B and LLAMA-13B models, FastQuery achieves more than 75.7×, 60.2×,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*DAC '24, June 23–27, 2024, San Francisco, CA, USA*
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0601-1/24/06
https://doi.org/10.1145/3649329.3657374

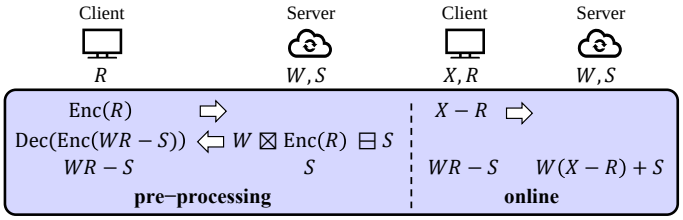


**Figure 2: Flow of secure embedding table query.**

**Table 1: Notations used in the paper.**

| Notations | Meanings |
|---|---|
| $\lceil\cdot\rceil, \lfloor\cdot\rfloor, \lfloor\cdot\rceil$ | Ceiling, flooring, and rounding operations |
| $\mathrm{Enc}(\cdot), \mathrm{Dec}(\cdot)$ | Homomorphic encryption and decryption |
| $\boxplus, \boxminus, \boxtimes$ | Homomorphic addition, subtraction and multiplication |
| $N, p, q$ | Polynomial degree, bit-width of plaintext and ciphertext |
| $m, n$ | Vocabulary size and embedding dimension |

20.2× communication reduction respectively with negligible model performance degradation.

## 2 PRELIMINARY

### 2.1 Threat Model

FastQuery focuses on efficient privacy-preserving DNN inference involving two parties, i.e., server and client. The server holds the model with private weights and the client holds private inputs. The model architecture, including the number of layers as well as the types, dimensions, and bit-widths for each layer, are public to both the server and the client [5, 7, 11]. After executing the protocol, the client learns the inference results without revealing any information to the server. Consistent with previous works [5, 7, 11], we adopt an honest-but-curious security model in which both parties follow the specifications of the protocol but also try to learn more from the information than allowed.

### 2.2 Flow of Secure Embedding Table Query

We summarize the notations used in the paper in Table 1. Figure 2 shows the flow of secure embedding table query. Previous work [14] considers embedding table query as a matrix-vector multiplication, e.g., $WX$, which has two stages: the pre-processing stage that generates the input-independent helper data and the online stage to process the client's real query. During the pre-processing stage, the client encrypts a randomly sampled $R$ and sends it to the server. Since HE supports homomorphic addition and multiplication without the need for decryption, the server can compute $\mathrm{Enc}(WR)$ and blind it by subtracting a randomly sampled $S$. Finally, the server sends the result to the client for decryption. During the online stage, the client blinds a one-hot query $X$ by subtracting $R$, i.e., $X - R$, and sends it to the server. Then, the server just needs to compute $W(X - R) + s$ and each party gets a secret share of the results $WX$, i.e., $WR - S$ by the client and $W(X - R) + S$ by the server.

### 2.3 Communication Complexity of Embedding Table Query

The main communication cost of embedding table query comes from the pre-processing stage while the online stage is usually lightweight since it only transfers $X - R$ in plaintext. The communication cost primarily comes from two parts in the pre-processing stage. As shown in Figure 2, the first part is the transfer of input ciphertext $\mathrm{Enc}(R)$ from the client to the server, which equals the product of the number of polynomials and the communication of each polynomial. The number of polynomials is $\lceil \frac{m}{N} \rceil$, and the communication of each polynomial scales with both the polynomial degree and the bit-width of each coefficient. Hence, the input communication complexity is $O(\lceil \frac{m}{N} \rceil \times Nq)$. The second part is the transfer of output ciphertext from the server to the client. Following the optimization of Cheetah, the output transfer communication is $O(n/\lceil \frac{N}{m} \rceil \times Nq + nq)$. Since the size of embedding table $m$ is quite large, e.g. 32000 for LLAMA-7B, $\lceil \frac{N}{m} \rceil = 1$ and thus the output communication can be simplified as $O(n \times Nq + nq)$.

**HE parameters** In HE, the bit-width of plaintext $p$ is decided by the accumulation bit-width of a matrix-vector multiplication to avoid overflow. Given $p$, $q - p$ determines the noise budget of the HE-based computation, which is related to the multiplication depth. Given $q$, $N$ can be determined according to the required security level.

**PackLWEs optimization** To improve the packing density of output polynomials, [11] proposes PackLWEs optimization which extracts the useful elements in each polynomial and packs them into a new polynomial. With the PackLWEs optimization, the number of output ciphertexts is reduced from $n$ to $\lceil \frac{n}{N} \rceil$.

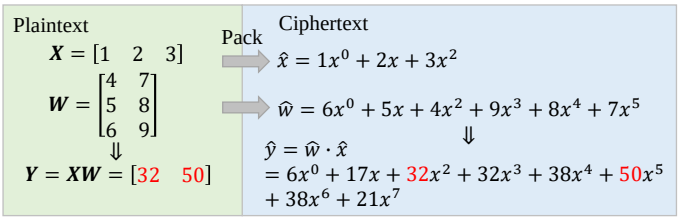


**Figure 3: A matrix-vector multiplication example of coefficient packing.**

### 2.4 Coefficient Packing of HE

HE encrypts plaintexts into ciphertext polynomials, namely, 1-dimensional vectors. However, DNNs operate on high-dimensional tensors. To bridge this gap, packing is necessary to encode tensors into polynomials. Cheetah [7] recognizes that polynomial multiplications inherently perform matrix-vector multiplication and introduces a HE coefficient packing method. In this approach, tensor elements are encoded into coefficients, and the correct result can be extracted from the outcome of polynomial multiplication. Figure 3 provides a toy example of this process. Iron further optimizes the algorithm of Cheetah for matrix-matrix multiplication. CHAM [18] adopts a different weight matrix partitioning method to maximize the number of output ciphertexts, which enables fully utilizing PackLWEs to reduce communication overhead.

### 2.5 Transformer Model Quantization

Quantization is a frequently used method in model compression. There are two approaches for language model quantization, quantization-aware training (QAT) [10] and post-training quantization (PTQ) [8]. For LLM quantization, PTQ is adopted by most of the works, due to the lack of access to training data and the high retraining cost of QAT [8]. However, there is little work focusing on LLM embedding quantization. This is because most of the parameters in LLMs are distributed in the encoder layers rather than the embedding table. In this paper, however, we propose a communication-aware post-training embedding table quantization method to reduce embedding table query latency.

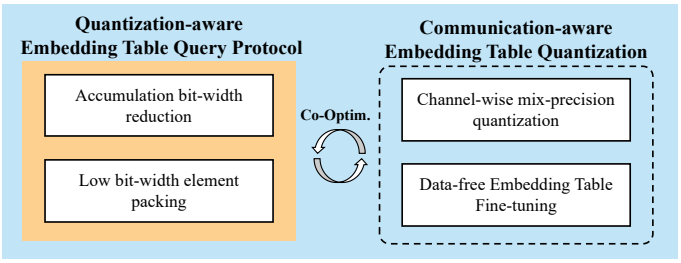


**Figure 4: Overview of FastQuery Framework.**

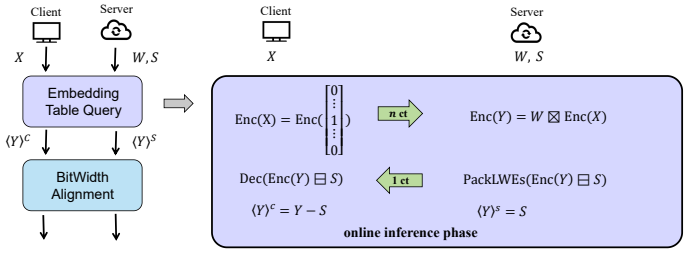


**Figure 5: The proposed private embedding table query protocol.**

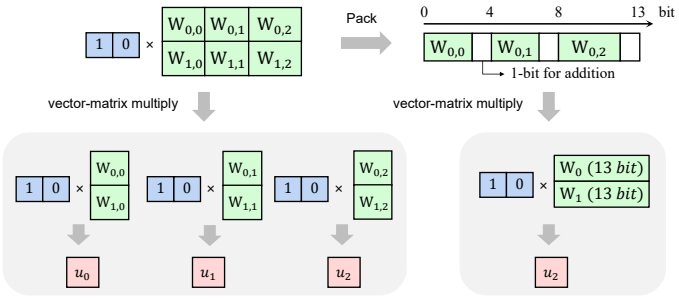


**Figure 6: Toy example of how we pack a matrix with three columns into one column. The bit-widths for each element in the matrix's channels are 3, 3, and 4 respectively and every 13-bit plaintext polynomial coefficient is fully packed. By employing the method we propose, we can merge the three channels into a single channel.**

## 3 MOTIVATION

In this section, we analyze the origin of the communication cost associated with embedding table query to motivate our work. As discussed in Section 2.3, the communication cost for input and output ciphertexts is given by $O(\lceil \frac{m}{N} \rceil \times Nq)$ and $O(n/\lceil \frac{N}{m} \rceil \times Nq + nq)$, respectively. We have made several observations to reduce the communication further.

**Observation 1: Both HE parameters $p$ and $q$ can be reduced with low bit-width quantized embedding query, bringing in improved communication efficiency.** As discussed in Section 2.3, the bit-width of plaintext $p$ and ciphertext $q$ is determined by the accumulation bit-width of a matrix-vector multiplication. Low bit-width quantization directly reduces the required accumulation bit precision, leading to efficient communication. Furthermore, the robustness of DNNs to quantization error suggests a promising direction for efficiency improvement.

**Observation 2: Although the embedding table can be quantized to low precision, e.g., sub-4-bit quantization, further reduction of $p$ and $q$ is not feasible.** Previous research has demonstrated the feasibility of quantizing LLM weights to low bit-widths, such as 3 bits [3, 8]. However, we have empirically found that reducing $p$ below 13 bits compromises the correctness of the decryption. Hence, the communication reduction saturates once $p$ is reduced below 13 bits [19, 22]. Therefore, to fully leverage LLMs' robustness to low bit-width quantization, how to reduce communication for sub-13-bit plaintext becomes important.

**Observation 3: Existing approaches move the HE computation to the pre-processing stage and thus, cannot leverage the one-hot nature of user queries.** Previous works [5–7] move the HE computation to the pre-processing stage, where we compute $WR$ instead of $WX$. However, since the query $X$ is a one-hot vector with only one non-zero element, there are possibilities to further reduce the communication cost. Furthermore, recent studies suggest that for DLaaS, pre-processing communication cannot be ignored and often incurs online latency due to insufficient server downtime to hide the communication costs [4]. Therefore, computing $WX$ directly in the online stage is a promising direction for further reducing the total communication cost.

## 4 FASTQUERY

### 4.1 Overview

Based on the observations above, we propose FastQuery, as shown in Figure 4. FastQuery features both network and protocol optimizations. For protocol optimization, we leverage the one-hot nature of user queries and the low bit-width embedding table to reduce the accumulation bit-width, and thus, the plaintext and ciphertext bit-widths. To further reduce the communication for sub-13-bit quantization, we propose a novel element packing algorithm to squeeze the embedding table dimension. For network optimization, we propose a channel-wise mix-precision quantization method to reduce the bit-width of the embedding table, which also considers the proposed element packing protocol. After quantization, we adopt a data-free fine-tuning to further recover model performance. With the network and protocol co-optimization, FastQuery demonstrates over 10× communication reduction compared to baseline protocols.

### 4.2 Quantization-aware Embedding Table Query Protocol

As mentioned in Section 3, offline pre-processing communication not only can potentially waste power for both the server and the client, but also can slow down the online stage. Therefore, we choose to perform the matrix-vector multiplication solely in the online phase. Moreover, we propose several optimizations to fully utilize the characteristics of one-hot encoding to improve communication efficiency. In Figure 5, we illustrate the proposed protocol for private embedding table query.

**Accumulation bit-width reduction** The core computation in the embedding table query is the matrix-vector multiplication $WX - S$. When querying the embedding table, we need to compute $WX - S$ homomorphically. Let $b_x$ and $b_w$ denote the bit-widths of $X$ and $W$, respectively. Then, a naive estimation of the bit-width for output vector $u$ is $b_u = b_x + b_w + b_{acc} + 1$, where $b_{acc} = \log_2 m$ is the bit-width required for the accumulation of the matrix-vector multiplication (recall $m$ is the vocabulary size) and +1 is to ensure the randomly sampled $S$ can be corrected subtracted [7]. To guarantee HE computation correctness, we need to have $p \leq b_u$. However, we find this is not necessary. Because the query is a one-hot vector, it is sufficient for $p = b_u = b_w + 1$ since there is only one non-zero value, which equals 1. Therefore, we can significantly reduce the plaintext modulus and the communication cost.

**Low bit-width aware element packing** While the one-hot nature of user queries enables us to reduce the accumulation bit-width to $b_w + 1$, the plaintext bit-width $p$ cannot be reduced to below 13 bits

**Table 2: Comparison on different granularities of INT3 embedding table quantization, evaluated on WikiText-103. Per-token quantization is forbidden as it can leak the query information.**

| Quantization Granularities | LLaMA-7B | LLaMA-13B | LLaMA-30B |
|---|---|---|---|
| Per-tensor | 15236.57 | 11210.18 | 1772.56 |
| Per-token (**forbidden**) | 10.949 | 9.576 | 6.000 |
| Per-channel | 10.950 | 8.268 | 5.659 |

to ensure correct decryption as introduced in Section 2. Hence, it is important to propose further optimizations for communication reduction. Specifically, we can pack multiple low bit-width elements into an element with a higher precision, making full use of the high-precision plaintext coefficients. In FastQuery, we fix the plaintext bit-width to 13. Depending on the embedding table precision, we always try to pack as many elements as possible. With the proposed packing algorithm, each dot product between the input vector and a column is equivalent to performing a dot product with multiple embedding table columns, which reduces both the computation and number of output polynomials for better communication efficiency. In Figure 6, we illustrate the packing process and provide an example to demonstrate the benefits of matrix-vector multiplication.

**Bit-width alignment** After computing the HE-based matrix-vector multiplication, we need to align the bit-width of each output element before the computation of the next layer. We extend the bit-width of all elements to the required bit by leveraging the bit-width extension protocol proposed by SiRNN [16]. The bit-width alignment step enables us to support a mixed-precision quantized embedding table, which boost the performance of low bit-width quantized networks with negligible communication cost.

## 4.3 Communication-aware Embedding Table Quantization

We now aim to reduce the bit-width of the embedding table while preserving the performance of LLMs. While per-tensor quantization is easy to implement, we find it can hurt model performance, as shown in Table 2. Since our proposed protocol can support mixed bit-widths, we study more fine-grained quantization strategies, including per-token quantization and per-channel quantization. As shown in Table 2, by increasing the quantization flexibility and the quantization parameters, both per-token and per-channel quantization achieve significantly better performance. However, per-token quantization can leak the query information. Since the quantization parameters, including the scale and bit-width, are publicly known, an adversary can infer the query based on which bit-width or scale is selected after the private table query. Hence, we propose a per-channel quantization strategy that consists of the following three steps.

**Number of channels and bit-width combination in each coefficient** As mentioned before, the bit-width of each plaintext coefficient is fixed to 13. In addition, to generate secret shares, the output needs to be subtracted by a random number $S$. So we need to reserve an additional 1 bit for each channel in each coefficient. Based on this, now we determine how many channels we should pack in one coefficient. If we set this number too low, e.g., two channels, the total number of coefficients will be high, leading to a higher embedding table query latency. If we set the number too high, e.g., four channels, more than half of the channels will be quantized to 2-bit or lower. We find that this will badly hurt model performance, shown in Table 4 and Table 5. So for query efficiency and model performance, we set the number of channels in each coefficient to 3. The bit-widths of the three channels

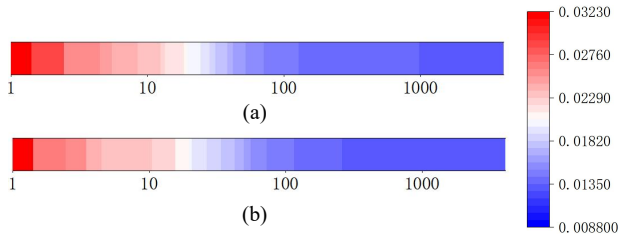


**Figure 7: Mean absolute values of different embedding channels in (a) LLaMA-7B and (b) LLaMA-13B.**

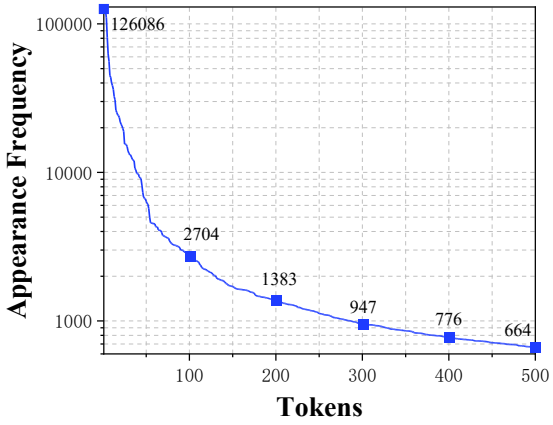


**Figure 8: Appearance frequencies of the most frequent 500 tokens in the WikiText-103 dataset.**

are set to 4-bit, 3-bit, and 3-bit (represented by 4, 3, 3) to prevent 2-bit quantization, which may hurt model performance.

**Precision of each packing channel** Now we need to determine which channel we should quantize to 4-bit in each coefficient. Considering outliers (weights with high absolute values) cause huge approximation errors in quantization, we observe the weight distribution in the embedding table, and find the mean absolute value of each channel varies greatly, as shown in Figure 7. Based on this, we quantize the salient channels whose weights have higher mean absolute values to 4-bit. There are other strategies such as considering gradient value [9] or Hessian value [3, 8]. While our method achieves the best performance in this bit-width combination and is easy to implement, as shown in Table 4 and Table 5.

In practice, we just quantize 1/3 of the embedding channels that have the highest mean absolute value into 4-bit and the other 2/3 into 3-bit. Then we change the order of embedding channels and the order of QKV projection weight rows in the first MHA block together, keeping the multiplication result unchanged. At last, the bit-widths of three adjacent channels will be 4-bit, 3-bit, and 3-bit, which can be packed in a coefficient. This adjustment can unify the bit-width combination order in each coefficient and will not introduce any calculation error.

**Data-free Embedding Table Fine-tuning** To further improve the model performance, we need to conduct fine-tuning on the quantized embedding table. A simple idea is to train the model on the downstream task, while this will lead to huge training costs and overfitting problems [8]. So we need a data-free post-training algorithm to conduct fine-tuning. At the same time, we find that different tokens in vocabulary usually have different importance because their appearance frequencies vary greatly, as shown in Figure 8. Based on these, we design a data-free embedding table fine-tuning algorithm to further reduce the impact of

quantization error. In this period, we want to minimize

$$L = \|I_{token}(Q(W'_{Emb})W_{QKV} - W_{Emb}W_{QKV})\|_2,$$

where $W_{QKV}$ is the concatenation of Q, K, and V projection weight in the first MHA block. $W_{Emb}$ and $Q(W'_{Emb})$ is the embedding table before and after quantization. $Q()$ is a function representing our per-channel mix-precision quantization method. $I_{token}$ is a diagonal matrix and the elements on the diagonal represent the appearance frequency of each token. This can better protect embedding vectors for important tokens that have a higher frequency. In practice, We set a threshold value to lower the elements in $I_{token}$ which is bigger than this value to this value. This can prevent the fine-tuning algorithm from paying too much attention to only a few tokens. We freeze $I_{token}$, $W_{Emb}$, and $W_{QKV}$, only changing $Q(W'_{Emb})$ with gradient descent and straight-through estimator (STE) . This further improves model performance without introducing much training time. It is worth noting that this fine-tuning algorithm is completely data-free, preventing the problem of over-fitting on downstream tasks faced by existing LLM quantization methods.

# 5 EXPERIMENT RESULTS

## 5.1 Experimental Setup

FastQuery is built on top of SEAL library [19] and OpenCheetah [7] in C++. We use EZPC [2] library to evaluate CrypTFlow2 [17] and SIRNN [16] and use SPU library [12] to simulate Iron [5] and Bumblebee [11]. All experiments were conducted on a server with a 2.4GHz Intel Xeon CPU and 125GB RAM. Following the Cheetah experiment protocol, we set $N$ to 4096 for most of our experiments. We quantize and evaluate LLaMA-2 family [21] on WikiText [13] and C4 [15], using a symmetric uniform quantizer. We round each scale factor to the power of 2 to be friendly to 2PC protocols [16].

## 5.2 Benchmark for Embedding Table Quantization

**Perplexity and Query Latency Comparison** As we are the first to focus on LLM embedding table quantization, we compare FastQuery with Round to Nearest per-channel quantization (RTN) [24]. As shown in Table 3, RTN INT4 quantization shows similar perplexity with the full precision model, while RTN INT3 can cause an obvious model performance drop. Compared with RTN INT4, FastQuery can pack more channels in one coefficient, which leads to a smaller number of coefficients with only a small performance drop. This can bring 1.39 ∼ 1.52× query latency reduction compared to RTN INT4 and 2.95 ∼ 3.40× compared to full precision models. Compared with RTN INT3, with the same query latency, FastQuery achieves much lower perplexity on each model size. After data-free fine-tuning, the performance of the quantized model can be further improved. On WikiText-103, our quantized LLaMA-13B and LLaMA-30B model even achieves lower perplexity than RTN INT4.

**Different bit-width combinations and salient channel judgment strategies** In our method, we use the mean absolute value to select the salient channels. In each 13-bit coefficient, we use the bit-width combination of 4-bit, 3-bit, and 3-bit (represented by 4, 3, 3). We also evaluate other bit-width combinations and salient channel judgment strategies, such as mean gradient value, mean Hessian value, etc. We find our strategy achieves the best performance on various datasets, as shown in Table 4 and Table 5. We also find that 2-bit quantization should be avoided since even 1/3 of the channels quantized to 2-bit can cause notable model performance reduction.

**Table 3: Comparison on WikiText-103 and C4 perplexity (↓) and embedding table query latency.**

| Method | # channels per-coefficient | Metrics | LLaMA-7B | | LLaMA-13B | | LLaMA-30B | |
|---|---|---|---|---|---|---|---|---|
| | | | Wiki | C4 | Wiki | C4 | Wiki | C4 |
| FP16 | / | PPL (↓) | 7.340 | 8.220 | 7.313 | 7.281 | 5.576 | 6.704 |
| | | Latency (s) | 11.39 | 11.39 | 14.27 | 14.27 | 17.45 | 17.45 |
| RTN (INT4) | 2 | PPL (↓) | 7.403 | 8.236 | 7.326 | 7.284 | 5.615 | 6.706 |
| | | Latency (s) | 5.11 | 5.11 | 6.39 | 6.39 | 8.23 | 8.23 |
| RTN (INT3) | 3 | PPL (↓) | 10.950 | 17.738 | 8.268 | 12.390 | 5.659 | 6.881 |
| | | Latency (s) | 3.35 | 3.35 | 4.33 | 4.33 | 5.91 | 5.91 |
| FastQuery | 3 | PPL (↓) | 7.679 | 8.409 | 7.327 | 7.312 | 5.618 | 6.713 |
| | | Latency (s) | 3.35 | 3.35 | 4.33 | 4.33 | 5.91 | 5.91 |
| FastQuery +Fine-tuning | 3 | PPL (↓) | 7.430 | 8.285 | 7.317 | 7.303 | 5.600 | 6.713 |
| | | Latency (s) | 3.35 | 3.35 | 4.33 | 4.33 | 5.91 | 5.91 |

**Table 4: Comparison of different bit-width combinations and salient channel judgment, evaluated on WikiText-103 perplexity (↓) of LLaMA-7B.**

| Criterion | 6, 2, 2 | 5, 3, 2 | 4, 3, 3 |
|---|---|---|---|
| Random | 17015.25 | 132.992 | 12.554 |
| Gradient | 2249.00 | 36.349 | 8.243 |
| Hessian value | 2670.43 | 55.715 | 8.478 |
| Gradient*Absolute Value | 1317.64 | 24.819 | 8.411 |
| Absolute Value | 750.80 | 15.313 | **7.679** |

**Table 5: Comparison of different bit-width combinations and salient channel judgment, evaluated on C4 perplexity (↓) of LLaMA-7B.**

| Criterion | 6, 2, 2 | 5, 3, 2 | 4, 3, 3 |
|---|---|---|---|
| Random | 21308.77 | 90.453 | 11.214 |
| Gradient | 1008.85 | 11.450 | 8.439 |
| Hessian value | 1197.39 | 12.468 | 8.441 |
| Gradient*Absolute Value | 591.41 | 10.060 | 8.432 |
| Absolute Value | 577.16 | 11.785 | **8.409** |

## 5.3 Benchmark for Embedding Table Query Protocol

**Different dimensions evaluation** We compare the communication of FastQuery with other methods across different embedding table dimensions. The experimental settings use embedding table dimensions from LLaMA-7B, LLaMA-13B, and LLaMA-30B. As shown in Figure 9, compared to Cheetah, FastQuery achieves a reduction in communication overhead of 75.2 ∼ 95.1×, as well as a reduction in latency of 4.3 ∼ 4.5×. FastQuery achieves a latency reduction of 2.7 ∼ 3.7× compared to Iron. Compared to Bumblebee, FastQuery also achieves a latency reduction of 1.3 ∼ 1.5× and a communication reduction of 20.4 ∼ 33.4×. Additionally, it can be observed that CryptFlow2 incurs significant latency overhead but relatively lower communication overhead. This is primarily due to its adoption of SIMD encoding, which reduces communication overhead through time-consuming rotations. In comparison, our method achieves a latency reduction of 164.2 ∼ 205.2× compared to CryptFlow2 and exhibits even lower communication overhead.

**Different bandwidth evaluation** We also compare the latency of FastQuery against CryptFlow2, Cheetah, and Iron under different bandwidths: 384Mb/s, 44Mb/s, and 9Mb/s, which are selected from the

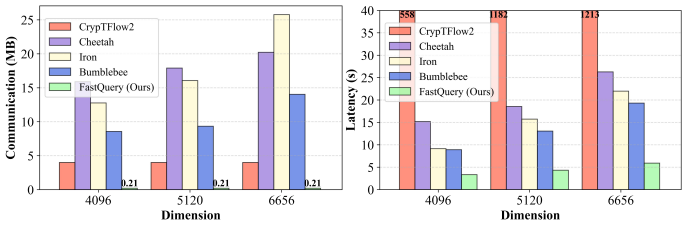


**Figure 9: Communication (MB) and Latency (s) comparison under different embedding table dimensions.**

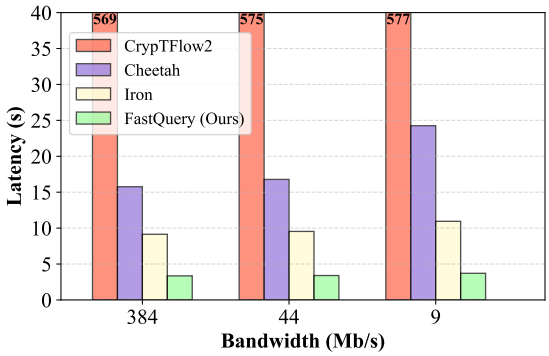


**Figure 10: Lantency (s) comparison under different bandwidths.**

**Table 6: ablation study of our proposed method in FastQuery with a size of (32000, 4096).**

| METHOD | Latency (s) | Communication (MB) |
|---|---|---|
| Baseline Cheetah | 15.19 | 15.933 |
| +Quant | 15.19 | 15.933 |
| +Accumulation bit-width reduction | 10.908 | 8.492 |
| +Low bit-width aware element packing | 3.492 | 4.911 |
| +PackLWEs | 3.353 | 0.212 |

experiments of Cheetah[7] and Falcon[23] respectively. As shown in Figure 10, the latency changes of FastQuery and CryptFlow2 are not significant, while those of Cheetah and Iron are more evident. The experimental results are consistent with the fact that smaller communication overhead leads to less sensitivity to bandwidth. We also find that our method reduces the latency by 4.7× to 6.5× compared to Cheetah.

## 5.4 Ablation study

To better understand the improvements of different methods on communication efficiency and latency, we incrementally integrate our proposed optimization techniques into the baseline Cheetah and we present the results in Table 6. According to the results, we can learn that 1) directly applying quantization does not provide significant benefits for accelerating embedding table query; 2) after quantization, reducing the bit-width of plaintext modulus and ciphertext modulus and packing low bit-width elements has a significant impact on reducing communication overhead and latency; 3) although the PackLWEs method incurs heavy computation costs, it reduces the communication significantly. These experimental results demonstrate that the proposed methods are highly effective in reducing both latency and communication overhead.

## 6 CONCLUSION

In this paper, we propose FastQuery, a private embedding table query framework, which leverages the one-hot nature of queries and robustness to low bit-width quantization noise, to improve communication efficiency. FastQuery features a communication-aware embedding table quantization algorithm and a one-hot-aware dense packing algorithm. Compared to prior-art HE-based frameworks, e.g., Cheetah, Iron, and Bumblebee, FastQuery achieves more than 4.3×, 2.7×, 1.3× latency reduction, respectively, and more than 75.7×, 60.2×, 20.2× communication reduction, respectively, on both LLAMA-7B and LLAMA-30B.

## ACKNOWLEDGE

This work was supported in part by the NSFC (62125401) and the 111 Project (B18001). We also thank Prof. Yier Jin and Dr. Jiafeng Hua from Huawei Co. Ltd. for the useful discussion and support.

## REFERENCES

[1] Tom Brown, Mann, et al. Language models are few-shot learners. *NeurIPs*, 2020.
[2] Nishanth Chandran, Divya Gupta, et al. Ezpc: Programmable, efficient, and scalable secure two-party computation for machine learning. *Cryptology ePrint Archive*, 2017.
[3] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.
[4] Karthik Garimella, Zahra Ghodsi, Nandan Kumar Jha, Siddharth Garg, and Brandon Reagen. Characterizing and optimizing end-to-end systems for private inference. In *ACM ASPLOS*, pages 89–104, 2023.
[5] Meng Hao, Hongwei Li, Hanxiao Chen, Pengzhi Xing, Guowen Xu, and Tianwei Zhang. Iron: Private inference on transformers. *NeurIPS*, 35:15718–15731, 2022.
[6] Xiaoyang Hou, Jian Liu, Jingyu Li, Yuhan Li, Wen-jie Lu, Cheng Hong, and Kui Ren. Ciphergpt: Secure two-party gpt inference. *Cryptology ePrint Archive*, 2023.
[7] Zhicong Huang, Wen-jie Lu, Cheng Hong, et al. Cheetah: Lean and fast secure two-party deep neural network inference. In *USENIX Security*, pages 809–826, 2022.
[8] Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W Mahoney, and Kurt Keutzer. Squeezellm: Dense-and-sparse quantization. *arXiv preprint arXiv:2306.07629*, 2023.
[9] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv preprint arXiv:2306.00978*, 2023.
[10] Zechun Liu, Barlas Oguz, Changsheng Zhao, et al. Llm-qat: Data-free quantization aware training for large language models. *arXiv preprint arXiv:2305.17888*, 2023.
[11] Wen-jie Lu, Zhicong Huang, Zhen Gu, et al. Bumblebee: Secure two-party inference framework for large transformers. *Cryptology ePrint Archive*, 2023.
[12] Junming Ma, Yancheng Zheng, Jun Feng, et al. SecretFlow-SPU: A performant and User-Friendly framework for Privacy-Preserving machine learning. In *USENIX ATC*, pages 17–33, Boston, MA, July 2023. USENIX Association.
[13] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.
[14] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and RalucaAda Popa. Delphi: A cryptographic inference service for neural networks, Jan 2020.
[15] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.
[16] Deevashwer Rathee, Mayank Rathee, Rahul Kranti Kiran Goli, Divya Gupta, Rahul Sharma, Nishanth Chandran, and Aseem Rastogi. Sirnn: A math library for secure rnn inference. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1003–1020. IEEE, 2021.
[17] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, et al. Cryptflow2: Practical 2-party secure inference. In *ACM SIGSAC CCS*, 2020.
[18] Xuanle Ren, Zhaohui Chen, Zhen Gu, et al. Cham: A customized homomorphic encryption accelerator for fast matrix-vector product. In *DAC*. IEEE, 2023.
[19] Microsoft SEAL (release 3.6). https://github.com/Microsoft/SEAL, November 2020. Microsoft Research, Redmond, WA.
[20] Hugo Touvron, Thibaut Lavril, Gautier Izacard, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.
[21] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
[22] Tianshi Xu, Meng Li, and Runsheng Wang. Hequant: Marrying homomorphic encryption and quantization for communication-efficient private inference. *arXiv preprint arXiv:2401.15970*, 2024.
[23] Tianshi Xu, Meng Li, Runsheng Wang, and Ru Huang. Falcon: Accelerating homomorphically encrypted convolutions for efficient private mobile network inference. *arXiv preprint arXiv:2308.13189*, 2023.
[24] Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *NeurIPS*, 35:27168–27183, 2022.