

Maverick: Private and Verifiable LLM Inference Made Practical via Matrix–Vector Multiplication Delegation

Ben Merbaum*
Yale University

Mohammad Amin Raeisi*
Yale University

Wenhao Wang*
Yale University, IC3

Charalampos Papamanthou
Yale University

Katerina Sotiraki
Yale University

Fan Zhang
Yale University, IC3

Abstract

Open-source large language models (LLMs) are increasingly competitive with closed-source models while offering transparency and the ability to run inference without exposing user inputs to a service provider. However, running large-scale models locally requires substantial computational resources. In practice, users may still resort to a third-party provider, giving rise to privacy and correctness concerns. Existing solutions that address these problems often impose substantial server overhead or introduce additional trust assumptions.

In this paper, we present *Maverick*, a novel approach to private and verifiable LLM inference based on a protocol for delegating matrix–vector multiplication, a dominant operation in LLMs. At its core, *Maverick* provides, to our knowledge, the first information-theoretically sound verification protocol for matrix–vector multiplication delegation with *transparent preprocessing*, *efficient (batch) verification*, and *virtually no server overhead*. We combine this verification primitive with LPN-based pseudorandom masking to provide input privacy.

We implement our matrix–vector delegation primitive and use it to build an end-to-end prototype of *Maverick*, which we evaluate on Qwen3–4B by measuring throughput in tokens per second. We evaluate client configurations with 1–8 threads. With one client thread and a CPU server using up to 128 threads, *Maverick* achieves throughput gains over local inference of up to $17\times$ when privacy masks are generated online, $45\times$ when they are precomputed, and $44\times$ when only verification is required. With four client threads, the corresponding gains are $13\times$, $18\times$, and $17\times$. When server computation is no longer the bottleneck, client-side microbenchmarks with simulated network delay show speedups of $12\times$ – $20\times$, $34\times$ – $135\times$, and $38\times$ – $157\times$.

*These authors contributed equally to this work and are ordered alphabetically.

[†]Emails: {ben.merbaum, amin.raeisi, wenhao.wang, charalampos.papamanthou, katerina.sotiraki, f.zhang}@yale.edu.

1 Introduction

Open-source models such as DeepSeek, Qwen, and Kimi are increasingly matching the performance of closed-source models [32, 56, 77]. For example, as of July 2026, Kimi K3 outperforms Claude models on frontend design tasks [33]. In addition, open-source models can be run locally, allowing users with sufficient hardware to keep their inputs private from service providers. Their publicly available weights also offer better transparency [27] and reproducibility [29], while reducing dependence on provider-controlled changes to pricing, model availability, and access policies. Despite various advantages, their substantial computational requirements make local deployment impractical or undesirable for most users (e.g., running Kimi K3 requires a cluster of high-end GPUs estimated to cost \$0.55–\$1 million to set up [35]). In practice, users resort to specialized service providers, reintroducing privacy and correctness concerns.

The problem of private and verifiable outsourcing of AI and LLM inference has been extensively studied. However, aside from recent systems based on trusted execution environments (TEEs) [5, 6, 74], existing cryptographic approaches typically incur significant overhead on the server side. For instance, proof systems [15, 36, 41, 42, 61, 62, 69, 70, 76, 85, 86, 88] and homomorphic encryption [43, 47–49, 51, 52, 59, 64, 66, 87, 92] have been used, either individually or in combination, to provide verifiability, input privacy, or both. However, generating proofs and performing fully homomorphic encryption (FHE) computations can be orders of magnitude more expensive than the original workload. Solutions based on TEEs offer practical performance, though their security guarantees may not always hold in practice [31].

In this paper, we present a lightweight protocol for private and verifiable outsourced LLM inference. Our protocol allows a *computationally constrained client*, such as a mobile device, to use an open-source model run by a powerful, untrusted server while hiding the client’s private input and supporting verification, all with *negligible* server overhead.

In particular, our technical contributions are as follows.

1) A protocol for fast verifiable matrix–vector multiplication delegation (vMVMD). To our knowledge, it is the first information-theoretically sound construction with transparent preprocessing and efficient verification runtime. 2) We combine vMVMD with pseudorandom masks using coding theory to hide client inputs without introducing server overhead, providing private and verifiable matrix–vector multiplication delegation (pvMVMD). 3) Building on pvMVMD, we present Maverick, a system for private and verifiable LLM inference.

Maverick outsources the linear operations of LLM inference to an untrusted server. Nonlinear operations [82] are inexpensive by comparison and are performed locally. We present an end-to-end prototype implementation and evaluation using the open-source model Qwen3-4B.

1.1 Verifiable Matrix–Vector Multiplication Delegation (vMVMD)

In a vMVMD protocol, a client with input $\mathbf{x} \in \mathbb{F}^n$ verifies that a server’s output $\mathbf{y} \in \mathbb{F}^m$ satisfies $\mathbf{y} = \mathbf{M}\mathbf{x}$ for a public matrix $\mathbf{M} \in \mathbb{F}^{m \times n}$. Since the client can always compute $\mathbf{y}$ locally in $O(mn)$ time, verification time must be less than $O(mn)$ for delegation to be meaningful. One idea is to apply Freivalds’ algorithm [40]. Given matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$, Freivalds’ algorithm checks whether $\mathbf{AB} \stackrel{?}{=} \mathbf{C}$ by sampling a random vector $\mathbf{r}$ and checking $(\mathbf{r}^\top \mathbf{A})\mathbf{B} \stackrel{?}{=} \mathbf{r}^\top \mathbf{C}$. However, when applied to the matrix–vector multiplication (MVM) $\mathbf{M}\mathbf{x} \stackrel{?}{=} \mathbf{y}$, it requires the client to compute $\mathbf{r}^\top \mathbf{M}$, which is as expensive as computing $\mathbf{y}$ locally.

Our idea is to use a *sparse* vector in Freivalds’ algorithm. Rather than sampling a uniform challenge $\mathbf{r}$, we sample a sparse vector $\mathbf{e}$ from $\mathbb{F}^m$ with t nonzero elements. To check $\mathbf{M}\mathbf{x} \stackrel{?}{=} \mathbf{y}$, the client computes $(\mathbf{e}^\top \mathbf{M})\mathbf{x}$ and $\mathbf{e}^\top \mathbf{y}$ in time $O(tn)$ and accepts if and only if they match. When $t = O(\kappa)$, verification takes $O_\kappa(m + n)$ time instead of $O(mn)$ time.

However, this strawman protocol is not sound. Consider an invalid claim $\mathbf{M}\mathbf{x} \neq \mathbf{y}$ for which $\mathbf{M}\mathbf{x}$ and $\mathbf{y}$ differ at only one index, say k . The above protocol only rejects if the k -th element of $\mathbf{e}$ happens to be nonzero, which happens with probability t/m , i.e., the soundness error is at least $1 - t/m$.

To reduce the soundness error, we use *error-correcting codes* to perform checks on encoded values. Let $\mathbf{G} \in \mathbb{F}^{m \times N}$ be the generator matrix of a linear code. The client samples a sparse challenge (of length N) and checks $(\mathbf{e}^\top \mathbf{G}^\top \mathbf{M})\mathbf{x} = \mathbf{e}^\top \mathbf{G}^\top \mathbf{y}$. Intuitively, the encoding spreads every nonzero error vector $\Delta := \mathbf{M}\mathbf{x} - \mathbf{y}$ into a codeword $\mathbf{G}^\top \Delta \in \mathbb{F}^N$ with enough nonzero entries that its support is likely to intersect the support of $\mathbf{e}$. This requires a code with high minimum Hamming weight (equivalently, high relative distance) and also fast encoding for fast client online time, properties satisfied by, for example, RAA codes [3].

Computing $\mathbf{G}^\top \mathbf{M}$ is relatively heavy, but $\mathbf{Q} = \mathbf{G}^\top \mathbf{M}$ is independent of client inputs and thus can be prepared in a

preprocessing phase. The preprocessing adds no extra trust assumption because both $\mathbf{G}$ and $\mathbf{M}$ are public and anyone can verify the correctness of $\mathbf{Q}$. With $\mathbf{Q}$ preprocessed, verification is efficient since $\mathbf{e}$ is sparse and $\mathbf{G}$ has linear-time encoding.

To further optimize client verification time, we develop a protocol for efficiently verifying a batch of MVMs involving different matrices and vectors. For each matrix $\mathbf{M}_b$, preprocessing produces $\mathbf{Q}_b = \mathbf{G}^\top \mathbf{M}_b$. The same sparse challenge $\mathbf{e}$ can be shared across all claims in a repetition, but the client would still need to compute the expensive vectors $\mathbf{Q}_b^\top \mathbf{e}$ separately for every matrix. We instead stack the computation of $\mathbf{Q}_b^\top \mathbf{e}$ into a single MVM and delegate it to the server. We then recursively apply vMVMD to verify this computation. Crucially, its input is now the sparse vector $\mathbf{e}$ itself, so the second check can be evaluated efficiently by the client.

1.2 Private and Verifiable Matrix–Vector Multiplication Delegation (pvMVMD)

We now show how to add privacy (i.e., hiding $\mathbf{x}$ from the server) on top of vMVMD without introducing server overhead. A naive construction is to have the client sample a uniformly random mask $\mathbf{x}'$ and send $\hat{\mathbf{x}} = \mathbf{x} + \mathbf{x}'$ to the server; upon receiving $\hat{\mathbf{y}}$, the client performs vMVMD to check $\mathbf{M}\hat{\mathbf{x}} \stackrel{?}{=} \hat{\mathbf{y}}$. To remove the mask from $\hat{\mathbf{y}} = \mathbf{M}(\mathbf{x} + \mathbf{x}')$, however, the client needs to compute $\mathbf{M}\mathbf{x}'$ and subtract it from $\hat{\mathbf{y}}$; computing $\mathbf{M}\mathbf{x}'$ takes $O(mn)$ work.

To avoid this overhead, we use another code $\mathbf{G}_x$ to generate pseudorandom privacy masks $\mathbf{x}' = \mathbf{G}_x \mathbf{e}_x$ from sparse random vectors $\mathbf{e}_x$ under standard cryptographic assumptions. To distinguish this code from the one used for verification, we denote it by $\mathbf{G}_x$, using the subscript x because privacy masks are added to the input $\mathbf{x}$. The resulting mask can be removed efficiently as follows. Because $\mathbf{M}\mathbf{G}_x$ is independent of the client’s input, it can be computed during preprocessing. With $\mathbf{P} := \mathbf{M}\mathbf{G}_x \in \mathbb{F}^{m \times N_x}$, we can write $\mathbf{M}\mathbf{x}' = (\mathbf{M}\mathbf{G}_x)\mathbf{e}_x = \mathbf{P}\mathbf{e}_x$. Since $\mathbf{e}_x$ is sparse, computing $\mathbf{P}\mathbf{e}_x$ requires combining only the columns of $\mathbf{P}$ corresponding to the nonzero entries of $\mathbf{e}_x$. In our construction, we use a code that satisfies the above requirement under the dual-LPN assumption [20].

Overall, the client time is $O_{\lambda, \kappa}(m + n)$, while the server performs the ordinary multiplication without any overhead.

1.3 Maverick: Private and Verifiable Outsourced LLM Inference

Maverick uses pvMVMD to outsource linear operations in LLM inference. Before nonlinear operations, the server returns the result of each linear operation to the client. The client evaluates nonlinear operations locally until it reaches the next linear operation, which it outsources to the server. In our evaluation, nonlinear operations account for less than 1% of total inference time, so this simple design outsources most of the computation. Moreover, our implementation uses

pipelining so the client can use its idle time to prepare for upcoming computation, such as generating the output masks used in decryption. To leverage batch verification, the client defers verification to the end of the inference execution. We call this strategy *optimistic verification*.

We also implement and evaluate an optimization in which the client precomputes privacy masks. This optimization is similar to how Beaver triples are pre-generated in multi-party computation (MPC) protocols to speed up multiplications [12]. We further present a verification-only variant of Maverick for scenarios where privacy is not a concern, e.g., when user prompts do not include sensitive information.

1.4 Implementation and Evaluation

We implement our pvMVMD protocol and an end-to-end prototype of Maverick. We evaluate pvMVMD on square matrices of dimension up to $n = 2^{14}$. The client time grows approximately linearly with n , while the preprocessing and native MVM grow quadratically. At $n = 2^{14}$, the client completes its online work in 9.33 ms, whereas performing the native matrix-vector multiplication locally would take 116.00 ms, yielding a $12.44\times$ client-side speedup. We compare the pvMVMD protocol against Dumas-Zucca [34], which is specialized for verifying matrix-vector products, and against Sum-Check instantiated with BaseFold [78, 91], which represents the general proof-based approach used by recent verifiable-ML systems, including DeepProve [41]. Across the evaluated dimensions up to $n = 2^{12}$, our implementation is up to $34.8\times$ faster than Dumas-Zucca and up to $194.8\times$ faster than Sum-Check+BaseFold, and our reported client time additionally includes input privacy.

We then evaluate Maverick using the open-source model Qwen3-4B [77]. We study the throughput gains achieved when a client outsources inference to a powerful server, where throughput is measured in tokens per second. We configure the client with 1–8 threads to model a computationally constrained device. For the server, we consider two regimes. First, we measure the throughput gain from outsourcing inference to our CPU server using 32–128 server threads, and we exclude network latency to isolate the computational benefit of outsourcing. Second, we measure the client-side throughput limit when server computation is sufficiently fast, as could be achieved with GPU acceleration, so that the client becomes the bottleneck. We evaluate this regime using client-side microbenchmarks that treat server computation as negligible. To capture the impact of communication in realistic deployments, this experiment incorporates simulated network RTTs to evaluate client throughput.

With one client thread and up to 128 server threads, standard, boost, and verification-only modes improve throughput over local inference by $17\times$, $45\times$, and $44\times$, respectively. With four client threads, the corresponding improvements are $13\times$, $18\times$, and $17\times$.

Using the results from the first experiment, we compare Maverick against DeepProve [41], which provides verification but not input privacy. Although the model sizes, number of input tokens, and hardware differ, our verification-only mode evaluates Qwen3-4B in 531.6 ms, including inference, with 45.1 ms of verification. DeepProve’s proof generation takes 34.2 s for GPT-2 and 81.6 s for Gemma 3, excluding native inference. Its corresponding verification times are 1.35 s and 1.91 s. Both models are at least $10\times$ smaller than Qwen3-4B.

In the second experiment, we measure steady-state client throughput under simulated network RTTs of 50 ms and 100 ms with sufficiently fast server computation. Across 1–8 client threads, standard mode improves throughput over local inference by $12\times$ – $20\times$, boost mode by $34\times$ – $135\times$, and verification-only mode by $38\times$ – $157\times$.

2 Related Work

HE- and MPC-based private-inference systems protect client inputs and, in some settings, model parameters, but replace native linear algebra with encrypted or secure computation [43, 44, 47–49, 51–53, 59, 64–66, 72, 92]. Specialized protocols exploit the structure of linear computation to reduce this overhead. EMVP [13], for instance, uses structured codes to efficiently delegate encrypted matrix-vector products while additionally hiding the stored matrix, but assumes a semi-honest server and cannot provide verifiability. Braverman and Newman [23] use trapdoored matrices for delegated linear algebra, including matrix multiplication and batched MVMs with efficiency in the amortized setting. Relatedly, Abbaszadeh et al. [1] use dual-LPN-friendly codes and transparent preprocessing to privately delegate multi-scalar multiplications. However, these works do not provide efficient verifiable delegation of a single MVM in our setting.

A large body of work verifies neural-network and LLM inference using interactive proofs and SNARKs [10, 36, 41, 42, 61, 62, 69, 70, 76]. These systems can certify substantially more of the inference computation, but require additional proof-generation work at the server. Since linear layers account for a substantial fraction of inference computation, efficient verification of matrix products is a central subproblem. A direct Freivalds-style check provides information-theoretic soundness, but the verification requires quadratic time [40]. Prior work has also combined Freivalds-style verification with coding theory to reduce its randomness complexity [14]; however, the resulting verification time remains quadratic. Slalom [80] performs quadratic verification work in a private preprocessing by computing a secret reusable Freivalds state; however, this state encounters leakage over a bounded number of queries. Further, Slalom uses fresh one-time masks to provide privacy for MVMs, but the corresponding demasking term is computed in preprocessing via an independent MVM, leaving quadratic preprocessing work per query. Slalom at the Carnival [25] reduces this per-query privacy-preprocessing

cost by deriving fresh masks from reusable public structure. However, we show in Appendix E that its vector-space instantiation can leak linear information about the client’s input, violating its claimed input-privacy guarantee.

Publicly verifiable schemes such as Fiore–Gennaro [37], Dumas–Zucca [34], and general sum-check or SNARK-based approaches [78] instead require authentication or proof generation in addition to the underlying computation, introducing considerable additional server-side work beyond the target MVM. LAMP [58] also combines Freivalds-style verification with error-correcting codes, but uses them inside a commit-and-prove SNARK for matrix multiplication.

Other systems that provide both privacy and verifiability include verifiable computation over encrypted data through FHE+ZKP and verifiable-FHE constructions [8, 26, 38, 39, 67, 83, 94], SNARGs specialized to matrix multiplication over encrypted data [79], plaintext-authentication approaches such as VERITAS [28], whose main encodings were subsequently cryptanalyzed in the fully homomorphic setting [30], VOLE-based proof systems [84–86, 88], DataSeal’s FHE-based redundancy checks [75], and schemes combining homomorphic matrix computation with Freivalds-style verification [60, 93]. These approaches require encrypted server computation, an auxiliary proof or authentication protocol, trusted hardware, or client-specific secret verification state. In contrast, Maverick achieves linear online client work with reusable transparent preprocessing and no secret verifier state; its base protocol requires neither trusted hardware nor heavy cryptographic machinery, and the server performs only one field MVM.

Concurrent work. We are aware of two concurrent papers, MOSAIC [90], and Gupta et al. [45], which have similarly employed matrix–vector multiplication delegation toward outsourcing LLM inference. MOSAIC provides privacy for the client input and model, but does not provide verifiability. Like Maverick, MOSAIC achieves asymptotically optimal online client time for an MVM; however, under the concrete parameterizations of the two protocols, Maverick has a substantially smaller constant, while its online client work additionally includes information-theoretic verification. Moreover, MOSAIC relies on both LWE and LPN and introduces Gaussian noise into the delegated computation, resulting in a nonzero protocol-error probability, whereas the MVM delegation of Maverick introduces no such error. Furthermore, Gupta et al. [45] provide both privacy and verifiability, but security relies on two non-colluding servers, whereas Maverick outsources inference to only a single server. Meanwhile, their verifiability mechanism requires a trusted setup party (as opposed to our transparent preprocessing) not to collude with the server performing outsourced computation; the verification protocol requires group rather than field operations, and soundness is computational rather than information-theoretic. Nevertheless, we note that both concurrent works require substantially less client-side storage for preprocessing state, which is a key bottleneck of our approach.

3 Preliminaries

Notation. We use $\mathbb{F}$ to denote a finite field, $\mathbb{F}^* = \mathbb{F} \setminus \{0\}$, and $\mathbb{F}_q$ for a prime power q to denote the field with q elements. Matrices are denoted by **bold** uppercase letters (e.g., $\mathbf{M}$) and vectors by **bold** lowercase letters (e.g., $\mathbf{x}$). For a vector $\mathbf{v}$, we write $\text{wt}(\mathbf{v})$ to denote its Hamming weight, the number of nonzero entries of $\mathbf{v}$. For a vector $\mathbf{v}$ of length n , we use $\text{supp}(\mathbf{v})$ to denote the support, the set of indices in $[n]$ where $\mathbf{v}$ is nonzero. We say that a vector $\mathbf{v}$ is t -sparse for a constant t if $\text{wt}(\mathbf{v}) = t$. We denote the set of t -sparse vectors in $\mathbb{F}^n$ by $\mathcal{E}_t^n$. We denote λ as a computational security parameter and κ as a statistical parameter. $\text{negl}(\cdot)$ denotes a negligible function and $X \approx_c Y$ denotes computational indistinguishability.

3.1 Freivalds’ Algorithm

Given matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$, Freivalds’ algorithm [40] allows a verifier to efficiently check whether $\mathbf{AB} \stackrel{?}{=} \mathbf{C}$ without explicitly computing the matrix product $\mathbf{AB}$. The verifier samples a uniformly random vector $\mathbf{r}$ of the appropriate dimension and checks whether $(\mathbf{r}^\top \mathbf{A})\mathbf{B} \stackrel{?}{=} \mathbf{r}^\top \mathbf{C}$. For square $n \times n$ matrices, this requires only matrix–vector multiplications, costing $O(n^2)$ field operations, whereas explicitly computing $\mathbf{AB}$ costs $O(n^3)$ operations using the standard algorithm. If $\mathbf{AB} = \mathbf{C}$, the check always accepts; otherwise, it accepts with probability at most $|\mathbb{F}|^{-1}$.

For verifying a matrix–vector multiplication $\mathbf{Ab} \stackrel{?}{=} \mathbf{c}$, Freivalds’ algorithm does not asymptotically improve the verifier’s work. Computing $(\mathbf{r}^\top \mathbf{A})\mathbf{b} \stackrel{?}{=} \mathbf{r}^\top \mathbf{c}$ requires $O(n^2)$ field operations, equivalent to directly computing $\mathbf{Ab}$.

3.2 Linear Error-Correcting Codes

An error-correcting code encodes messages as fixed-length vectors, called codewords, by adding redundancy that enables error detection or correction; it is *linear* when its codewords form a linear subspace. Mathematically, let $n < N$ be integers and let $\mathbf{G} \in \mathbb{F}^{n \times N}$ have linearly independent rows. The corresponding linear code is $\mathcal{C} = \mathcal{C}(\mathbf{G}) := \{\mathbf{G}^\top \mathbf{v} : \mathbf{v} \in \mathbb{F}^n\} \subseteq \mathbb{F}^N$, and $\mathbf{G}$ is its generator matrix. The parameters n and N are the dimension and block length, respectively, and $R(\mathcal{C}) = n/N$ is the rate of $\mathcal{C}$. The minimum distance d of a linear code $\mathcal{C}$ is the minimum Hamming weight of a nonzero codeword. The *relative distance* of $\mathcal{C}$ is $\delta = d/N$.

We say that a family of codes $\{\mathcal{C}_n\}_{n=1}^\infty$ over a field $\mathbb{F}$ is *asymptotically good* if both the code rate R and the relative distance δ are constant for large enough n (see, e.g. [46]). Such codes provide efficiency in our constructions as they enable maximal error detection under minimal redundancy.

3.3 Learning Parity with Noise

Let $\mathcal{G}$ be a distribution over code generators $\mathbf{G} \in \mathbb{F}^{n \times N}$ and let $\mathcal{E}$ be a noise distribution over sparse vectors $\mathbf{e} \in \mathbb{F}^N$. Informally, we say the $(\mathcal{G}, \mathcal{E}, \mathbb{F})$ -dual-LPN assumption holds for dimensions n and N if $(\mathbf{G}, \mathbf{Ge}) \approx_c (\mathbf{G}, \mathbf{u})$ where $\mathbf{G} \leftarrow \mathcal{G}$, $\mathbf{e} \leftarrow \mathcal{E}$, and $\mathbf{u} \leftarrow \mathbb{F}^n$. We say $\mathcal{G}$ is *dual-LPN-friendly* if this assumption is believed to hold for the exact noise distributions $\mathcal{E} = \mathcal{E}_t^N$ for a suitable sparsity parameter t . A formal definition of the LPN assumption and a discussion of plausibly dual-LPN-friendly code families is provided in Appendix A.

4 Model and Security Definitions

We study cryptographic primitives for private and verifiable delegation of matrix–vector products, and their applications for LLM inference. The core primitives, vMVMD and pvMVMD, involve a *client* holding private inputs $\mathbf{x}$, and an untrusted *server*. In both primitives, the client wishes to obtain $\mathbf{y} = \mathbf{M} \cdot \mathbf{x}$ from the server for a given public matrix $\mathbf{M}$. vMVMD allows the client to verify the correctness of $\mathbf{y}$, and pvMVMD additionally hides $\mathbf{x}$ from the server.

Before defining their security guarantees, we explain what it means for these primitives to be efficient and field-agnostic. For a matrix $\mathbf{M} \in \mathbb{F}^{m \times n}$ and vectors $\mathbf{x} \in \mathbb{F}^n$ and $\mathbf{y} \in \mathbb{F}^m$, computing $\mathbf{y} = \mathbf{M}\mathbf{x}$ locally takes $O(mn)$ time. We call a protocol *efficient* if it runs in time $O_{\lambda, \kappa}(mn)$ and *asymptotically time-optimal* if it runs in time $O_{\lambda, \kappa}(m+n)$, matching the time required to process the input and output vectors $\mathbf{x}$ and $\mathbf{y}$.

We call a protocol *field-agnostic* if it treats $\mathbb{F}$ as a black box, allowing field elements to have arbitrary labels and accessing them only through an oracle for field operations. As in [13], we assume that the oracle provides access to addition, subtraction, multiplication, inversion, unit, zero testing, and uniform sampling of field elements.

4.1 Verifiable Matrix–Vector Multiplication Delegation (vMVMD)

In verifiable matrix–vector multiplication delegation (vMVMD), a client would like to efficiently verify if a server’s claim $\mathbf{y} \stackrel{?}{=} \mathbf{M}\mathbf{x}$ is correct. We formalize this primitive in Definition 4.1.

A vMVMD protocol consists of two algorithms: PREPROCESS, which takes the matrix $\mathbf{M}$ and outputs public parameters pp, and VERIFY, which uses pp to verify input–output pairs $(\mathbf{x}, \mathbf{y})$.

Our soundness definition guarantees information-theoretic (statistical) security. In particular, it is not only computationally hard for an adversary to find a pair $(\mathbf{x}, \mathbf{y})$ that fools VERIFY, but furthermore, there does not exist any such pair $(\mathbf{x}, \mathbf{y})$ that fools VERIFY with non-negligible probability taken over the random coins within the VERIFY algorithm.

Definition 4.1 (Verifiable Matrix–Vector Multiplication Delegation). *A vMVMD protocol Π is a pair of PPT algorithms (PREPROCESS, VERIFY) defined by:*

- $\text{PREPROCESS}(1^\kappa, \mathbf{M}) \rightarrow \text{pp}$. *On input statistical parameter κ and matrix $\mathbf{M} \in \mathbb{F}^{m \times n}$, outputs public parameters pp.*
- $\text{VERIFY}(\text{pp}, \mathbf{x}, \mathbf{y}) \rightarrow \{0, 1\}$. *On input public parameters pp, input vector $\mathbf{x} \in \mathbb{F}^n$, and output vector $\mathbf{y} \in \mathbb{F}^m$, either accepts (outputs 1) or rejects (outputs 0).*

The protocol should satisfy the following properties:

- **Completeness:** Π is complete if for all $\mathbf{M} \in \mathbb{F}^{m \times n}$ and $\mathbf{x} \in \mathbb{F}^n$,

$$\Pr \left[b = 1 \mid \begin{array}{l} \text{pp} \leftarrow \text{PREPROCESS}(1^\kappa, \mathbf{M}) \\ \mathbf{y} = \mathbf{M}\mathbf{x} \\ b \leftarrow \text{VERIFY}(\text{pp}, \mathbf{x}, \mathbf{y}) \end{array} \right] = 1.$$

- **Soundness:** Π is sound if for all unbounded adversaries $\mathcal{A}$ there exists a negligible function $\text{negl}(\cdot)$ such that for all $\mathbf{M} \in \mathbb{F}^{m \times n}$ and $\mathbf{x} \in \mathbb{F}^n$ ¹,

$$\Pr \left[\mathbf{y} \neq \mathbf{M}\mathbf{x} \wedge b = 1 \mid \begin{array}{l} \text{pp} \leftarrow \text{PREPROCESS}(1^\kappa, \mathbf{M}) \\ \mathbf{y} \leftarrow \mathcal{A}(\text{pp}, \mathbf{M}, \mathbf{x}) \\ b \leftarrow \text{VERIFY}(\text{pp}, \mathbf{x}, \mathbf{y}) \end{array} \right] < \text{negl}(\kappa).$$

4.2 Private and Verifiable Matrix–Vector Multiplication Delegation (pvMVMD)

In private and verifiable matrix–vector multiplication delegation (pvMVMD), a client would like to outsource the computation of $\mathbf{M}\mathbf{x}$ to the server without revealing $\mathbf{x}$. The client encrypts $\mathbf{x}$ to obtain $\hat{\mathbf{x}}$, the server evaluates $\hat{\mathbf{y}} = \mathbf{M}\hat{\mathbf{x}}$, and the client verifies and decrypts the result. The server may deviate from the prescribed protocol, and a public verification algorithm provides soundness for the client against such deviations. We formalize this primitive in Definition 4.2.

Definition 4.2 (Private and Verifiable Matrix–Vector Multiplication Delegation). *A pvMVMD protocol Π is a tuple of PPT algorithms (PREPROCESS, ENCRYPT, VERIFY, DECRYPT) defined by:*

- $\text{PREPROCESS}(1^\lambda, 1^\kappa, \mathbf{M}) \rightarrow \text{pp}$. *On input security parameter λ , statistical parameter κ , and matrix $\mathbf{M} \in \mathbb{F}^{m \times n}$, outputs public parameters pp.*
- $\text{ENCRYPT}(\text{pp}, \mathbf{x}) \rightarrow (\hat{\mathbf{x}}, \text{st})$. *On input public parameters pp and plaintext vector $\mathbf{x} \in \mathbb{F}^n$, outputs the ciphertext $\hat{\mathbf{x}}$ and a private state st.*

¹The adversary is unbounded, providing information-theoretic security.

- $\text{VERIFY}(\text{pp}, \hat{\mathbf{x}}, \hat{\mathbf{y}}) \rightarrow \{0, 1\}$. On input public parameters pp , ciphertext $\hat{\mathbf{x}} \in \mathbb{F}^n$, and output vector $\hat{\mathbf{y}} \in \mathbb{F}^m$, either accepts (outputs 1) or rejects (outputs 0).
- $\text{DECRYPT}(\text{pp}, \text{st}, \hat{\mathbf{y}}) \rightarrow \mathbf{y}$. On input public parameters pp , private state st , and ciphertext $\hat{\mathbf{y}} \in \mathbb{F}^m$, outputs the plaintext vector $\mathbf{y}$ which is claimed to equal $\mathbf{M}\mathbf{x}$.

The protocol should satisfy the following properties:

- **Completeness:** Π is complete if, whenever Π is executed honestly by both the client and the server, then the client outputs $\mathbf{y} = \mathbf{M}\mathbf{x}$ on input $\mathbf{x}$ and public matrix $\mathbf{M}$. Formally, for any $\mathbf{M} \in \mathbb{F}^{m \times n}$ and $\mathbf{x} \in \mathbb{F}^n$,

$$\Pr \left[\begin{array}{l} \text{pp} \leftarrow \text{PREPROCESS}(1^\lambda, 1^\kappa, \mathbf{M}) \\ (\hat{\mathbf{x}}, \text{st}) \leftarrow \text{ENCRYPT}(\text{pp}, \mathbf{x}) \\ \hat{\mathbf{y}} = \mathbf{M}\hat{\mathbf{x}} \\ b \leftarrow \text{VERIFY}(\text{pp}, \hat{\mathbf{x}}, \hat{\mathbf{y}}) \\ \mathbf{y} \leftarrow \text{DECRYPT}(\text{pp}, \text{st}, \hat{\mathbf{y}}) \end{array} \right] = 1.$$

- **Soundness:** Π is sound if a computationally unbounded server $\mathcal{A}$ that returns a vector $\hat{\mathbf{y}} \neq \mathbf{M}\hat{\mathbf{x}}$ will be rejected by VERIFY with high probability. Formally, there exists some negligible function $\text{negl}(\cdot)$ such that for any $\mathbf{M} \in \mathbb{F}^{m \times n}$ and $\mathbf{x} \in \mathbb{F}^n$,

$$\Pr \left[\begin{array}{l} \text{pp} \leftarrow \text{PREPROCESS}(1^\lambda, 1^\kappa, \mathbf{M}) \\ (\hat{\mathbf{x}}, \text{st}) \leftarrow \text{ENCRYPT}(\text{pp}, \mathbf{x}) \\ \hat{\mathbf{y}} \leftarrow \mathcal{A}(\text{pp}, \mathbf{M}, \hat{\mathbf{x}}) \\ b \leftarrow \text{VERIFY}(\text{pp}, \hat{\mathbf{x}}, \hat{\mathbf{y}}) \\ \mathbf{y} \leftarrow \text{DECRYPT}(\text{pp}, \text{st}, \hat{\mathbf{y}}) \end{array} \right] < \text{negl}(\kappa).$$

- **Client privacy:** Π is private for the client if executing Π reveals no information about $\mathbf{x}$ to the server. Formally, there exists a PPT simulator Sim such that for any matrix $\mathbf{M} \in \mathbb{F}^{m \times n}$ and $\mathbf{x} \in \mathbb{F}^n$, the following distributions are computationally indistinguishable with respect to κ and λ :

$$\text{Sim}(1^\lambda, 1^\kappa, \mathbf{M}) \approx_c \text{View}_{\text{Server}}^\Pi(\mathbf{M}, \mathbf{x}),$$

where $\text{View}_{\text{Server}}^\Pi(\mathbf{M}, \mathbf{x}) := (\text{pp}, \mathbf{M}, \hat{\mathbf{x}})$ denotes the server's view, given by $\text{pp} \leftarrow \text{PREPROCESS}(1^\lambda, 1^\kappa, \mathbf{M})$ and $(\hat{\mathbf{x}}, \text{st}) \leftarrow \text{ENCRYPT}(\text{pp}, \mathbf{x})$.

Remark 4.3 (Multi-query privacy). Client privacy extends to multiple queries via a standard hybrid argument, and the multi-query distinguishing advantage for q queries is at most q times the single-query advantage.

Remark 4.4 (Composition of vMVMD with pvMVMD). We remark that any possibly unsound pvMVMD protocol can be made sound by using the VERIFY algorithm of a sound vMVMD protocol. A formal proof appears in Appendix D.

Π_{vMVMD}

Input parameters:

- Distribution $\mathcal{G}$ over generator matrices $\mathbf{G} \in \mathbb{F}^{m \times N}$ of codes with rate $R = \Theta(1)$, relative distance $\delta = \Theta(1)$, and linear encoding time.
- Sparsity parameter $t = t(\kappa)$ and repetition parameter $\ell = \ell(\kappa)$.
- Block length $N = O(m)$.
- Field oracle $\mathbb{F}$.

PREPROCESS($1^\kappa, \mathbf{M}$) $\rightarrow$ pp:

- Sample $\mathbf{G} \leftarrow \mathcal{G}$.
- Compute $\mathbf{Q} := \mathbf{G}^\top \mathbf{M} \in \mathbb{F}^{N \times n}$ and output $\text{pp} = (\mathbf{G}, \mathbf{Q}, t, \ell)$.

VERIFY(pp, $\mathbf{x}, \mathbf{y}$) $\rightarrow$ {0, 1}:

- Compute $\mathbf{G}^\top \mathbf{y} \in \mathbb{F}^N$.
- Repeat ℓ times:
 - Sample $\mathbf{e} \leftarrow \mathcal{E}_t^N$.
 - Compute $v_L := (\mathbf{e}^\top \mathbf{Q})\mathbf{x}$ and $v_R := \mathbf{e}^\top (\mathbf{G}^\top \mathbf{y})$.
 - If $v_L \neq v_R$, output 0 (reject) and halt.
- If all ℓ checks pass, output 1 (accept).

Protocol 1: Verifiable matrix–vector multiplication delegation protocol

4.3 Transparent preprocessing

We call the preprocessing *transparent* if its output contains no secret trapdoor or client-specific verification key and can be validated against the public matrix $\mathbf{M}$ by any party. The server or a third party may therefore generate and publish the preprocessing state once for all clients delegating the same matrix. A client that does not trust the publisher can independently validate the state at a cost comparable to preprocessing and then reuse it for all future queries involving $\mathbf{M}$. Alternatively, the publisher may attach a proof of correct preprocessing to reduce this per-client validation cost.

5 Verifiable Matrix–Vector Multiplication Delegation (vMVMD)

In this section, we present a novel asymptotically time-optimal vMVMD protocol using tools from coding theory. We present our construction in Section 5.1, demonstrate its security in Section 5.2, analyze its costs in Section 5.3, and extend it with a batch verification protocol in Section 5.4, tailored to the batched MVM claims arising in our LLM application.

5.1 Our Construction

We begin by modifying Freivalds' algorithm to enable faster client verification of MVMs. Recall that $\mathcal{E}_t^m$ denotes the set of t -sparse vectors of dimension m . Rather than sampling a Freivalds' challenge uniformly at random, we sample a t -

sparse vector $\mathbf{e} \in \mathcal{E}_t^m$. Given a verification claim $\mathbf{M}\mathbf{x} \stackrel{?}{=} \mathbf{y}$, the client can compute $(\mathbf{e}^\top \mathbf{M})\mathbf{x}$ and $\mathbf{e}^\top \mathbf{y}$ in time $O(tn)$ and accept if and only if they match. When $t = O(\kappa)$, we have a protocol that runs in $O_\kappa(m+n)$ to verify instead of $O(mn)$.

However, the soundness of this protocol is not ideal. Consider an invalid expression $\mathbf{M}\mathbf{x} \neq \mathbf{y}$, and define the *error vector* $\Delta := \mathbf{M}\mathbf{x} - \mathbf{y} \in \mathbb{F}^m$. The previous protocol accepts whenever $\mathbf{e}^\top \Delta = 0$. For instance, if there is exactly one nonzero entry in Δ (i.e., $\text{wt}(\Delta) = 1$), then the protocol rejects only if $\mathbf{e}$ is nonzero in the same coordinate as Δ , which occurs with probability t/m . This yields a soundness error of at least $1 - t/m$.

To reduce the soundness error, we use *linear codes* (see Section 3.2) for dimension m and a block length N with the following two properties: (1) the code has *high relative distance* δ when the rate $R = m/N$ is kept constant for security, and (2) the code has *fast encoding time* for efficiency. Intuitively, the encoding spreads every nonzero error vector Δ into a codeword $\mathbf{G}^\top \Delta \in \mathbb{F}^N$ with at least δN nonzero entries.

Let $\mathbf{G} \in \mathbb{F}^{m \times N}$ be the generator matrix of such a linear code. Rather than verifying the original claim directly, we verify its encoding under $\mathbf{G}^\top$. That is, the client samples a sparse challenge $\mathbf{e} \in \mathcal{E}_t^N$ and accepts if $(\mathbf{e}^\top \mathbf{G}^\top \mathbf{M})\mathbf{x} = \mathbf{e}^\top \mathbf{G}^\top \mathbf{y}$.

If $\Delta = \mathbf{M}\mathbf{x} - \mathbf{y} \neq 0$, the minimum distance of the code guarantees that $\mathbf{G}^\top \Delta \in \mathbb{F}^N$ has at least δN nonzero entries. Thus, the support of a uniformly random t -sparse vector $\mathbf{e}$ is likely to intersect the support of $\mathbf{G}^\top \Delta$.

Computing $\mathbf{G}^\top \mathbf{M}$ for every query incurs heavy work, but we recognize that the term $\mathbf{Q} = \mathbf{G}^\top \mathbf{M}$ is independent of client input, and thus can be prepared in a preprocessing phase. Anyone can verify the correctness of $\mathbf{Q}$ since both $\mathbf{G}$ and $\mathbf{M}$ are public; our preprocessing phase adds no extra trust assumption. The full protocol is formally presented in Protocol 1.

Meanwhile, $(\mathbf{e}^\top \mathbf{Q})\mathbf{x}$ and $\mathbf{e}^\top (\mathbf{G}^\top \mathbf{y})$ can be computed efficiently, since $\mathbf{e}^\top \mathbf{Q}$ can be computed in time proportional to the sparsity of $\mathbf{e}$ and the fast encoding time of $\mathbf{G}$ makes computing $\mathbf{G}^\top \mathbf{y}$ efficient.

5.2 Security

We show that Π_{VMVMD} satisfies soundness for suitable parameters t , ℓ , and δ . We first prove the following lemma.

Lemma 5.1. *Let $\mathbf{Q} := \mathbf{G}^\top \mathbf{M} \in \mathbb{F}^{N \times n}$, and let $\mathbf{e}$ be sampled uniformly from $\mathcal{E}_t^N$. If the code generated by $\mathbf{G}$ has relative distance δ , then for any $\mathbf{x} \in \mathbb{F}^n$ and $\mathbf{y} \in \mathbb{F}^m$ such that $\mathbf{M}\mathbf{x} \neq \mathbf{y}$, we have $\Pr[\mathbf{e}^\top (\mathbf{G}^\top \mathbf{y}) = (\mathbf{e}^\top \mathbf{Q})\mathbf{x}] \leq (1 - \delta)^t + \frac{1}{|\mathbb{F}| - 1}$.*

Proof. Let $\mathbf{u} := \mathbf{y} - \mathbf{M}\mathbf{x} \in \mathbb{F}^m$ be the nonzero error vector and $\mathbf{z} := \mathbf{G}^\top \mathbf{u} \in \mathbb{F}^N$ its encoding. The event $\mathbf{e}^\top (\mathbf{G}^\top \mathbf{y}) = (\mathbf{e}^\top \mathbf{Q})\mathbf{x}$ is equivalent to $\mathbf{e}^\top \mathbf{z} = 0$. Since $\mathbf{G}$ generates a code of relative distance δ , we have $\text{wt}(\mathbf{z}) \geq N\delta$.

Let $S_e := \text{supp}(\mathbf{e})$ and $S_z := \text{supp}(\mathbf{z})$. We bound $\Pr[\mathbf{e}^\top \mathbf{z} = 0]$ by splitting into two cases based on whether S_e and S_z

intersect and applying the law of total probability.

$$\begin{aligned} & \Pr[\mathbf{e}^\top \mathbf{z} = 0] \\ &= \Pr[\mathbf{e}^\top \mathbf{z} = 0 \mid S_e \cap S_z = \emptyset] \Pr[S_e \cap S_z = \emptyset] \\ &+ \Pr[\mathbf{e}^\top \mathbf{z} = 0 \mid S_e \cap S_z \neq \emptyset] \Pr[S_e \cap S_z \neq \emptyset] \\ &\leq \Pr[S_e \cap S_z = \emptyset] + \Pr[\mathbf{e}^\top \mathbf{z} = 0 \mid S_e \cap S_z \neq \emptyset] \end{aligned}$$

Term 1. By assumption, $|S_z| \geq N\delta$ and $|S_e| = t$. Since S_e is a uniformly random size- t subset of $[N]$ and S_z is fixed, then,

$$\Pr[S_e \cap S_z = \emptyset] \leq \frac{\binom{N-N\delta}{t}}{\binom{N}{t}} = \prod_{i=0}^{t-1} \frac{N-N\delta-i}{N-i} \leq \left(\frac{N-N\delta}{N}\right)^t = (1 - \delta)^t.$$

Term 2. Define the multivariate polynomial $f_z(\mathbf{e}) = \mathbf{e}^\top \mathbf{z}$. Conditioned on $S_e \cap S_z \neq \emptyset$, f_z is nonzero and has total degree 1 in the variables $\{e_j\}_{j \in S_e}$. These variables are sampled uniformly at random from the set $\mathbb{F} \setminus \{0\}$. By the Schwartz-Zippel lemma, the probability that this polynomial evaluates to 0 is at most $\Pr[f_z(\mathbf{e}) = 0 \mid S_e \cap S_z \neq \emptyset] \leq \frac{1}{|\mathbb{F}| - 1}$.

Combining both cases yields the final probability. $\square$

The soundness bound of Lemma 5.1 can be amplified using independent repetitions. We choose t so that the two terms in the single-iteration bound are comparable, and repeat ℓ times to obtain soundness error at most $2^{-\kappa}$. We show that Π_{VMVMD} satisfies Definition 4.1 with the following theorem.

Theorem 5.2. *If the code generated by $\mathbf{G}$ has relative distance δ , then the protocol Π_{VMVMD} satisfies completeness and soundness as defined in Definition 4.1 for $|\mathbb{F}| \geq 4$, $\ell = \left\lceil \frac{\kappa}{\log_2(|\mathbb{F}| - 1) - 1} \right\rceil$, and $t = \left\lceil \frac{\max(\log_2(|\mathbb{F}| - 1), \kappa + 1)}{-\log_2(1 - \delta)} \right\rceil$.*

Proof. Completeness is immediate. For any $\mathbf{M} \in \mathbb{F}^{m \times n}$ and $\mathbf{x} \in \mathbb{F}^n$, let $\mathbf{y} = \mathbf{M}\mathbf{x}$. Then, for every t -sparse vector $\mathbf{e} \in \mathbb{F}^N$,

$$(\mathbf{e}^\top \mathbf{Q})\mathbf{x} = \mathbf{e}^\top \mathbf{G}^\top \mathbf{M}\mathbf{x} = \mathbf{e}^\top \mathbf{G}^\top \mathbf{y}.$$

For soundness, fix any unbounded adversary $\mathcal{A}$, matrix $\mathbf{M} \in \mathbb{F}^{m \times n}$, and vector $\mathbf{x} \in \mathbb{F}^n$. Condition on any public parameters pp output by PREPROCESS for which the code generated by $\mathbf{G}$ has relative distance δ , and on any output $\mathbf{y} \leftarrow \mathcal{A}(\text{pp}, \mathbf{M}, \mathbf{x})$ such that $\mathbf{M}\mathbf{x} \neq \mathbf{y}$. By Lemma 5.1, each iteration of VERIFY accepts with probability at most $(1 - \delta)^t + \frac{1}{|\mathbb{F}| - 1}$. The protocol accepts only if all ℓ iterations pass. Since the iterations sample their challenges $\mathbf{e}$ independently, the total acceptance probability is at most $\left((1 - \delta)^t + \frac{1}{|\mathbb{F}| - 1}\right)^\ell$.

We now consider the two parameter spaces.

Case 1: Suppose $\frac{1}{|\mathbb{F}| - 1} \leq 2^{-(\kappa+1)}$. The choice of t gives $(1 - \delta)^t \leq 2^{-(\kappa+1)}$. Since $\ell = 1$, the soundness error is at most

$$(1 - \delta)^t + \frac{1}{|\mathbb{F}| - 1} \leq 2^{-(\kappa+1)} + 2^{-(\kappa+1)} = 2^{-\kappa}.$$

Case 2: Suppose $\frac{1}{|\mathbb{F}| - 1} > 2^{-(\kappa+1)}$. The choice of t gives $(1 - \delta)^t \leq \frac{1}{|\mathbb{F}| - 1}$. Therefore, the soundness error is at

most $\left(\frac{2}{|\mathbb{F}|-1}\right)^\ell = 2^{-\ell(\log_2(|\mathbb{F}|-1)-1)}$. By the choice of ℓ , then $\ell(\log_2(|\mathbb{F}|-1)-1) \geq \kappa$, and hence $\left(\frac{2}{|\mathbb{F}|-1}\right)^\ell \leq 2^{-\kappa}$.

The bound holds for every public parameter pp satisfying the relative distance condition and every incorrect output chosen by $\mathcal{A}$. We therefore obtain

$$\Pr \left[\begin{array}{l} \mathbf{y} \neq \mathbf{M}\mathbf{x} \wedge b = 1 \\ \text{pp} \leftarrow \text{PREPROCESS}(1^\kappa, \mathbf{M}) \\ \mathbf{y} \leftarrow \mathcal{A}(\text{pp}, \mathbf{M}, \mathbf{x}) \\ b \leftarrow \text{VERIFY}(\text{pp}, \mathbf{x}, \mathbf{y}) \end{array} \right] \leq 2^{-\kappa}.$$

Thus, Π_{vMVMd} satisfies Definition 4.1. $\square$

Remark 5.3 (Composing with code failure probability). *Suppose $\mathbf{G} \leftarrow \mathcal{G}$ is sampled from a code distribution and let $p_{\text{fail}} = p_{\text{fail}}(\mathcal{G}, N, \delta)$ be the probability that the sampled code has relative distance smaller than δ , then the overall soundness error of Π_{vMVMd} is at most $p_{\text{fail}} + 2^{-\kappa}$, which remains negligible when p_{fail} is negligible. Thus, for code families where p_{fail} can be made arbitrarily small under practical parameters, the protocol satisfies Definition 4.1.*

5.3 Efficiency

Operation	Cost
$\mathbf{G}^\top \mathbf{y}$	$O(N)$
$\mathbf{e}^\top (\mathbf{G}^\top \mathbf{y}) = \sum_{j \in \text{supp}(\mathbf{e})} e_j (\mathbf{G}^\top \mathbf{y})_j$	$2t - 1$
$\mathbf{e}^\top \mathbf{Q} = \sum_{j \in \text{supp}(\mathbf{e})} e_j \mathbf{Q}_{j,*}$	$(2t - 1)n$
$(\mathbf{e}^\top \mathbf{Q})\mathbf{x}$	$2n - 1$
ℓ repetitions total	$O(N + \ell tn)$

Table 1: Cost tally for the client online phase of Π_{vMVMd} .

For optimal efficiency, we instantiate Π_{vMVMd} with a code with linear encoding time, constant rate, and constant distance. Under the parameters for ℓ and t specified in Theorem 5.2, $\ell \cdot t = O(\kappa)$, so Π_{vMVMd} has asymptotic time $O_\kappa(m + n)$. The verification cost is summarized in Table 1. The cost is asymptotically time-optimal, since reading the input $\mathbf{x}$ and output $\mathbf{y}$ takes $O(m + n)$ time.

Preprocessing time. The preprocessing phase computes $\mathbf{Q} = \mathbf{G}^\top \mathbf{M} \in \mathbb{F}^{N \times n}$, i.e., n encodings of the columns of $\mathbf{M}$. Since the encoding time is linear with linear-encodable codes, PREPROCESS has running time $O(mn)$.

5.4 Batch Verification

In our LLM application, a single inference produces many MVM claims that must eventually be verified. The same setting arises more generally whenever a client verifies computations under a fixed collection of matrices. Suppose the client wishes to verify $\mathbf{M}_b \mathbf{x}_b \stackrel{?}{=} \mathbf{y}_b$ for $b \in [B]$, where $\mathbf{M}_b \in \mathbb{F}^{m \times n}$,

claims with different dimensions can be grouped by shape and verified in separate batches.

Recall that preprocessing computes $\mathbf{Q}_b := \mathbf{G}^\top \mathbf{M}_b$. Within each verification repetition, the same sparse challenge $\mathbf{e}$ can be shared across all B claims without increasing the soundness error. However, the client must still compute $\mathbf{Q}_b^\top \mathbf{e}$ for every claim and every repetition, costing $O(B\ell tn)$ field operations, which is usually the bottleneck. Our first observation is that these challenge-dependent projections can themselves be delegated to the server.

For one challenge $\mathbf{e}$, define $\mathbf{u}_b := \mathbf{Q}_b^\top \mathbf{e}$ and stack all the B computations into the single relation $\tilde{\mathbf{u}} = \tilde{\mathbf{Q}}\mathbf{e}$ where $\tilde{\mathbf{Q}} := [\mathbf{Q}_1 \mid \mathbf{Q}_2 \mid \dots \mid \mathbf{Q}_B]^\top$. We ask the server to compute $\tilde{\mathbf{u}}$ and verify this auxiliary relation using a second code $\mathbf{G}' \in \mathbb{F}^{Bn \times N'}$. Preprocessing computes $\tilde{\mathbf{Q}}^{\text{aux}} := \tilde{\mathbf{Q}}^\top \mathbf{G}'$. After the server fixes its auxiliary response, the client samples a fresh t' -sparse challenge $\mathbf{e}'$, computes $\mathbf{r} := \mathbf{G}'\mathbf{e}'$, and checks $\mathbf{e}^\top \tilde{\mathbf{Q}}^{\text{aux}} \mathbf{e}' \stackrel{?}{=} \mathbf{r}^\top \tilde{\mathbf{u}}$. The full protocol repeats this primary check ℓ times and uses ℓ' independent secondary repetitions.

Several observations make this second verification much more efficient. Unlike the original challenge projection, this auxiliary relation can be checked directly by the client: because both $\mathbf{e}$ and $\mathbf{e}'$ are sparse, the left-hand side accesses only a $t \times t'$ submatrix of $\tilde{\mathbf{Q}}^{\text{aux}}$, while the right-hand side is a single inner product of length Bn . Thus, no further delegation is needed. Moreover, the full protocol runs ℓ independent primary repetitions and ℓ' independent secondary repetitions. Within each secondary repetition, the same challenge $\mathbf{e}'$ can be used to check all ℓ auxiliary responses, since all of them are fixed before $\mathbf{e}'$ is sampled. Consequently, $\mathbf{G}'\mathbf{e}'$ is computed only once per secondary repetition, although the resulting scalar check is performed against each primary response.

The resulting protocol adds one challenge-response round and moves the expensive projection work from the client to the server, reducing the client's online time from $O(BN + B\ell tn)$ naively to $O(BN + B\ell(n + t) + \ell' t' Bn + \ell \ell' (Bn + t t'))$. We give the complete protocol in Protocol 2. In the following, we analyze this protocol.

We first prove the security of $\Pi_{\text{Batch-vMVMd}}$. The proof follows the ideas in the primary Π_{vMVMd} analysis.

Theorem 5.4 (Security of batch verification). *Suppose $\mathbf{G}$ and $\mathbf{G}'$ generate codes of relative distances at least δ and δ' , respectively. Let $\epsilon := (1 - \delta)^t + \frac{1}{|\mathbb{F}|-1}$ and $\epsilon' := (1 - \delta')^{t'} + \frac{1}{|\mathbb{F}|-1}$. Then $\Pi_{\text{Batch-vMVMd}}$ has perfect completeness and soundness error at most*

$$\epsilon^\ell + (\epsilon')^{\ell'}.$$

In particular, the parameters can be chosen so that the soundness error is at most $2^{-\kappa}$.

Proof. Completeness follows directly from $\mathbf{u}_b^{(q)} = \mathbf{Q}_b^\top \mathbf{e}^{(q)}$ and $\tilde{\mathbf{u}}^{(q)} = \tilde{\mathbf{Q}}\mathbf{e}^{(q)}$.

For soundness, first suppose every auxiliary response is correct. Fix any incorrect original claim b^* and let $\Delta_{b^*} :=$

$\mathbf{y}_{b^*} - \mathbf{M}_{b^*} \mathbf{x}_{b^*} \neq \mathbf{0}$. Acceptance in primary repetition q requires $(\mathbf{e}^{(q)})^\top \mathbf{G}^\top \Delta_{b^*} = 0$. By Lemma 5.1, this occurs with probability at most ε per repetition, and hence with probability at most ε^ℓ over all ℓ independent repetitions. Sharing $\mathbf{e}^{(q)}$ across the B claims introduces no factor B , since batch acceptance requires this fixed incorrect claim to accept.

Otherwise, fix any incorrect auxiliary response $q^* \in [\ell]$ and define $\mathbf{\Gamma}^{(q^*)} := \tilde{\mathbf{u}}^{(q^*)} - \tilde{\mathbf{Q}}\mathbf{e}^{(q^*)} \neq \mathbf{0}$. This vector is fixed before the secondary challenges are sampled. Acceptance in secondary repetition j requires $(\mathbf{e}'^{(j)})^\top \mathbf{G}'^\top \mathbf{\Gamma}^{(q^*)} = 0$, which again occurs with probability at most ε' by Lemma 5.1. Thus all ℓ' secondary repetitions accept with probability at most $(\varepsilon')^{\ell'}$. Sharing each secondary challenge across the ℓ auxiliary responses introduces no factor ℓ , since batch acceptance requires the fixed incorrect response q^* to accept.

Combining the two cases gives the claimed bound. $\square$

If either code is sampled from a distribution that may have relative distance below its target value, the corresponding code-distance failure probabilities are added to the above bound, as in Remark 5.3.

Client online time. The client first computes $\mathbf{z}_b := \mathbf{G}^\top \mathbf{y}_b$ for all $b \in [B]$, costing $O(BN)$ using the primary linear-time encoder. The original checks then cost $O(B\ell(n+t))$.

For each secondary repetition j , since $\mathbf{e}'^{(j)}$ is t' -sparse, $\mathbf{G}'\mathbf{e}'^{(j)} = \sum_{k \in \text{supp}(\mathbf{e}'^{(j)})} e'_{k,*} \mathbf{G}'_{*,k}$ and can therefore be computed in $O(t'Bn)$ field operations. Because the same secondary challenge is shared across all ℓ primary responses, this cost is incurred only ℓ' times.

For every pair (q, j) , computing $(\mathbf{e}^{(q)})^\top \tilde{\mathbf{Q}}^{\text{aux}} \mathbf{e}'^{(j)}$ costs $O(t't')$, while $(\mathbf{r}^{(j)})^\top \tilde{\mathbf{u}}^{(q)}$ costs $O(Bn)$. Hence the total online client time is

$$O(BN + B\ell(n+t) + \ell't'Bn + \ell\ell'(Bn + t't')).$$

Independent verification instead pays $O(B\ell n)$ for computing the challenge-dependent projections $\mathbf{Q}_b^\top \mathbf{e}^{(q)}$. Thus, batch verification moves this work to the server. For constant-distance codes, t, ℓ, t' , and ℓ' depend only on κ , so the client time is $O_\kappa(B(m+n))$.

The optimization adds one challenge–response round. The server performs $O(B\ell n)$ additional field operations and returns $B\ell n$ field elements.

Preprocessing and state. Batch verification additionally preprocesses $\tilde{\mathbf{Q}}^{\text{aux}} := \tilde{\mathbf{Q}}^\top \mathbf{G}' \in \mathbb{F}^{N \times N'}$. If applying the secondary encoder to a vector in $\mathbb{F}^{Bn}$ costs $T_{G'}(Bn, N')$, this requires $O(NT_{G'}(Bn, N'))$ preprocessing time and NN' field elements of additional state. For a constant-rate, linear-time encodable secondary code, $N' = \Theta(Bn)$, so both quantities are $O(Bmn)$.

The storage can be reduced when $\mathbf{G}'$ is a systematic code. If $\mathbf{G}' = [\mathbf{I}_{Bn} \mid \mathbf{P}]$, then $\tilde{\mathbf{Q}}^{\text{aux}} = [\tilde{\mathbf{Q}}^\top \mid \tilde{\mathbf{Q}}^\top \mathbf{P}]$. The first block is already represented by $\mathbf{Q}_1, \dots, \mathbf{Q}_B$, so only $\tilde{\mathbf{Q}}^\top \mathbf{P}$ needs additional storage. This optimization reduces storage but does

not inherently reduce preprocessing time, which depends on the cost of applying the secondary encoder.

The auxiliary preprocessing depends only on the fixed matrices in the batch and can therefore be reused across future batches involving the same matrices.

$\Pi_{\text{Batch-vMVM}}^{\text{D}}$

Input parameters:

- Distribution $\mathcal{G}$ over generator matrices $\mathbf{G} \in \mathbb{F}^{m \times N}$ of codes with rate $R = \Theta(1)$, relative distance $\delta = \Theta(1)$, and efficient encoding time.
- Distribution $\mathcal{G}'$ over generator matrices $\mathbf{G}' \in \mathbb{F}^{Bn \times N'}$ of codes with rate $R' := Bn/N'$, relative distance δ' , and efficient encoding time.
- Primary sparsity and repetition parameters t and ℓ .
- Secondary sparsity and repetition parameters t' and ℓ' .
- Primary block length $N = O(m)$ and secondary block length N' .
- Field oracle $\mathbb{F}$.

PREPROCESS($1^\kappa, \{\mathbf{M}_b\}_{b \in [B]}$) $\rightarrow$ pp:

- Sample $\mathbf{G} \leftarrow \mathcal{G}$ and $\mathbf{G}' \leftarrow \mathcal{G}'$.
- For every $b \in [B]$, compute $\mathbf{Q}_b := \mathbf{G}^\top \mathbf{M}_b \in \mathbb{F}^{N \times n}$.
- Form $\tilde{\mathbf{Q}} := (\mathbf{Q}_1, \dots, \mathbf{Q}_B)^\top \in \mathbb{F}^{Bn \times N}$.
- Compute $\tilde{\mathbf{Q}}^{\text{aux}} := \tilde{\mathbf{Q}}^\top \mathbf{G}' \in \mathbb{F}^{N \times N'}$.
- Output $\text{pp} := (\mathbf{G}, \mathbf{G}', \{\mathbf{Q}_b\}_{b \in [B]}, \tilde{\mathbf{Q}}^{\text{aux}}, t, \ell, t', \ell')$.
- Make the public matrices $\{\mathbf{Q}_b\}_{b \in [B]}$ available to the server.

VERIFY(pp, $\{(\mathbf{x}_b, \mathbf{y}_b)\}_{b \in [B]}$) $\rightarrow \{0, 1\}$:

- **Client:** Require that all alleged outputs $\mathbf{y}_1, \dots, \mathbf{y}_B$ are fixed before sampling any verification challenge.
- **Client:** For every $b \in [B]$, compute $\mathbf{z}_b := \mathbf{G}^\top \mathbf{y}_b \in \mathbb{F}^N$.
- **Client:** Sample independent primary challenges $\mathbf{e}^{(1)}, \dots, \mathbf{e}^{(\ell)} \leftarrow \mathcal{E}_t^N$ and send $\{\mathbf{e}^{(q)}\}_{q \in [\ell]}$ to the server. Within each repetition q , the same challenge $\mathbf{e}^{(q)}$ is used for all B claims.
- **Server:** For every $q \in [\ell]$, compute $\tilde{\mathbf{u}}^{(q)} := \tilde{\mathbf{Q}}\mathbf{e}^{(q)} \in \mathbb{F}^{Bn}$ and return $\{\tilde{\mathbf{u}}^{(q)}\}_{q \in [\ell]}$ to the client.
- **Client:** After all server responses are fixed, sample independent secondary challenges $\mathbf{e}'^{(1)}, \dots, \mathbf{e}'^{(\ell')} \leftarrow \mathcal{E}_{t'}^{N'}$.
- **Client:** For every $j \in [\ell']$, compute $\mathbf{r}^{(j)} := \mathbf{G}'\mathbf{e}'^{(j)} \in \mathbb{F}^{Bn}$.
- **Client:** For every $q \in [\ell]$ and $j \in [\ell']$, check $(\mathbf{e}^{(q)})^\top \tilde{\mathbf{Q}}^{\text{aux}} \mathbf{e}'^{(j)} \stackrel{?}{=} (\mathbf{r}^{(j)})^\top \tilde{\mathbf{u}}^{(q)}$. If any secondary check fails, output 0 (reject) and halt.
- **Client:** For every $q \in [\ell]$, parse $\tilde{\mathbf{u}}^{(q)} := ((\mathbf{u}_1^{(q)})^\top, \dots, (\mathbf{u}_B^{(q)})^\top)^\top$, where $\mathbf{u}_b^{(q)} \in \mathbb{F}^n$.
- **Client:** For every $b \in [B]$ and $q \in [\ell]$, check $(\mathbf{u}_b^{(q)})^\top \mathbf{x}_b \stackrel{?}{=} (\mathbf{e}^{(q)})^\top \mathbf{z}_b$. If any original check fails, output 0 (reject) and halt. If all checks pass, output 1 (accept).

Protocol 2: Batch verification for multiple matrices and vectors. The client delegates the stacked matrix-dependent challenge projection to the server and verifies the returned vector using a second coded check.

6 Private and Verifiable Matrix–Vector Multiplication Delegation (pvMVMD)

In this section, we present a construction for pvMVMD by combining Π_{VMVMD} in Section 5 for verification with dual-LPN-friendly linear codes for efficient privacy. We introduce our construction in Section 6.1, analyze its security in Section 6.2, and its costs in Section 6.3.

6.1 Our Construction

To encrypt a vector $\mathbf{x} \in \mathbb{F}^n$, the client masks $\mathbf{x}$ with a vector $\mathbf{x}'$ to form the ciphertext $\hat{\mathbf{x}} = \mathbf{x} + \mathbf{x}' \in \mathbb{F}^n$. The client sends $\hat{\mathbf{x}}$ to the server, which returns $\hat{\mathbf{y}} = \mathbf{M}\hat{\mathbf{x}}$. The client verifies the correctness of $\hat{\mathbf{y}}$ by running a vMVMD protocol over the ciphertexts. The client decrypts $\hat{\mathbf{y}}$ by computing $\mathbf{y} = \hat{\mathbf{y}} - \mathbf{M}\mathbf{x}' \in \mathbb{F}^m$. However, if $\mathbf{x}'$ is sampled uniformly at random, the time required for the client to compute $\mathbf{M}\mathbf{x}'$ matches the time for the client to locally compute $\mathbf{y} = \mathbf{M}\mathbf{x}$, rendering it useless to outsource this computation. To solve the efficiency problem, we prepare $\mathbf{x}'$ pseudorandomly such that $\mathbf{M}\mathbf{x}'$ can be computed in time $O(m+n)$. We make use of coding theory to construct such pseudorandom vectors.

Our construction for privacy follows the idea of prior works [1, 13, 23]. We use a separate distribution of linear codes $\mathcal{G}_x$ for privacy, distinct from the code used by Π_{VMVMD} in Section 5. Let $\mathbf{G}_x \in \mathbb{F}^{n \times N_x}$ denote a sampled generator matrix, where N_x is the block length of the privacy code. Under the $(\mathcal{G}_x, \mathcal{E}_{t_x}^{N_x}, \mathbb{F})$ -dual-LPN assumption, if $\mathbf{e}_x \in \mathbb{F}^{N_x}$ is t_x -sparse then $\mathbf{x}' := \mathbf{G}_x \mathbf{e}_x$ is a pseudorandom vector. We mask $\mathbf{x}$ by computing $\hat{\mathbf{x}} = \mathbf{x} + \mathbf{x}'$. By first computing $\mathbf{P} := \mathbf{M}\mathbf{G}_x \in \mathbb{F}^{m \times N_x}$ in a preprocessing phase, we can remove the mask by computing $\mathbf{M}\mathbf{x}' = \mathbf{M}(\mathbf{G}_x \mathbf{e}_x) = (\mathbf{M}\mathbf{G}_x) \mathbf{e}_x = \mathbf{P}\mathbf{e}_x$. Since $\mathbf{e}_x$ is sparse, computing $\mathbf{P}\mathbf{e}_x$ requires combining only the columns of $\mathbf{P}$ corresponding to the nonzero entries of $\mathbf{e}_x$.

The full protocol Π_{pvMVMD} is presented in Protocol 3.

The client privacy property can be strengthened to additionally hide a private matrix $\mathbf{M}$ owned by the client. While such a protocol enables greater privacy, our construction requires a private client-run preprocessing, rendering this most useful when the preprocessing can be amortized across many queries. For further details on our construction, see Appendix C.

6.2 Security

We now demonstrate that Π_{pvMVMD} satisfies completeness, soundness, and client privacy according to Definition 4.2. For verification, we use the same parameters as in Section 5. For privacy, the security analysis of prior works [20] demonstrates that a good minimum distance suffices for the dual-LPN assumption against known attacks under certain conditions (see Appendix A), and they provide the following

Π_{pvMVMD}

Input parameters:

- Distribution $\mathcal{G}_x$ over generator matrices $\mathbf{G}_x \in \mathbb{F}^{n \times N_x}$ of codes with rate $R_x = \Theta(1)$ and relative distance $\delta_x = \Theta(1)$, linear encoding time, and satisfying the dual-LPN assumption for error distribution $\mathcal{E}_{t_x}^{N_x}$.
- Distribution $\mathcal{G}_y$ over generator matrices $\mathbf{G}_y \in \mathbb{F}^{m \times N_y}$ of codes with rate $R_y = \Theta(1)$, relative distance $\delta_y = \Theta(1)$, and linear encoding time.
- Sparsity parameters $t_x = t_x(\lambda)$ and $t_y = t_y(\kappa)$ and repetition parameter $\ell = \ell(\kappa)$.
- Block lengths $N_x = O(n)$ and $N_y = O(m)$.
- Field oracle $\mathbb{F}$.

PREPROCESS($1^\lambda, 1^\kappa, \mathbf{M}$) $\rightarrow$ pp:

- Sample $\mathbf{G}_x \leftarrow \mathcal{G}_x$ and $\mathbf{G}_y \leftarrow \mathcal{G}_y$.
- Compute $\mathbf{P} := \mathbf{M}\mathbf{G}_x \in \mathbb{F}^{m \times N_x}$.
- Compute $\mathbf{Q} := \mathbf{G}_y^\top \mathbf{M} \in \mathbb{F}^{N_y \times n}$.
- Output pp = ($\mathbf{G}_x, \mathbf{G}_y, \mathbf{P}, \mathbf{Q}, t_x, t_y, \ell$).

ENCRYPT(pp, $\mathbf{x}$) $\rightarrow$ ($\hat{\mathbf{x}}, \text{st}$):

- Sample $\mathbf{e}_x \leftarrow \mathcal{E}_{t_x}^{N_x}$ and compute $\mathbf{x}' := \mathbf{G}_x \mathbf{e}_x \in \mathbb{F}^n$.
- Output $\hat{\mathbf{x}} := \mathbf{x} + \mathbf{x}'$ and $\text{st} := \mathbf{e}_x$.

VERIFY(pp, $\hat{\mathbf{x}}, \hat{\mathbf{y}}$) $\rightarrow$ {0, 1}:

- Compute $\mathbf{G}_y^\top \hat{\mathbf{y}} \in \mathbb{F}^{N_y}$.
- Repeat ℓ times:
 - Sample $\mathbf{e}_y \leftarrow \mathcal{E}_{t_y}^{N_y}$.
 - Compute $v_L := (\mathbf{e}_y^\top \mathbf{Q}) \hat{\mathbf{x}}$ and $v_R := \mathbf{e}_y^\top (\mathbf{G}_y^\top \hat{\mathbf{y}})$.
 - If $v_L \neq v_R$, output 0 (reject) and halt.
- If all ℓ checks pass, output 1 (accept).

DECRYPT(pp, st, $\hat{\mathbf{y}}$) $\rightarrow$ $\mathbf{y}$:

- Output $\mathbf{y} := \hat{\mathbf{y}} - \mathbf{P}\mathbf{e}_x \in \mathbb{F}^m$.

Protocol 3: Private and verifiable matrix–vector multiplication delegation protocol

heuristic for selecting t_x as a function of λ, N_x and δ_x :

$$t_x \geq \frac{(\ln 2)(\lambda - \log_2(N_x))}{\delta_x}. \quad (1)$$

We therefore choose an asymptotically good code for $\mathbf{G}_x$, with constant rate $R_x = \Theta(1)$, $N_x = n/R_x = O(n)$, and good relative distance $\delta_x = \Theta(1)$, guaranteeing $t_x = O_\lambda(1)$ under this heuristic. We show that Π_{pvMVMD} satisfies Definition 4.2 in Theorem 6.1.

Theorem 6.1. *Suppose that $\mathcal{G}_x$ is a distribution over codes with relative distance at least δ_x satisfying the $(\mathcal{G}_x, \mathcal{E}_{t_x}^{N_x}, \mathbb{F})$ -dual-LPN assumption for polynomially many independent samples, and that the code generated by $\mathbf{G}_y$ has relative distance δ_y . Then Π_{pvMVMD} satisfies completeness, soundness, and client privacy as defined in Definition 4.2 when instantiated with $|\mathbb{F}| \geq 4$, $\ell = \left\lceil \frac{\kappa}{\log_2(|\mathbb{F}|-1)-1} \right\rceil$, $t_y =$*

$$\left\lceil \frac{\max(\log_2(|\mathbb{F}|-1), \kappa+1)}{-\log_2(1-\delta_y)} \right\rceil.$$

Proof. Completeness is immediate. For any $\mathbf{M} \in \mathbb{F}^{m \times n}$ and $\mathbf{x} \in \mathbb{F}^n$, let $\mathbf{e}_x \leftarrow \mathcal{E}_{t_x}^{N_x}$, $\mathbf{x}' := \mathbf{G}_x \mathbf{e}_x$, and $\hat{\mathbf{x}} := \mathbf{x} + \mathbf{x}'$. If the server evaluates honestly, then $\hat{\mathbf{y}} = \mathbf{M}\hat{\mathbf{x}}$. By completeness of the verification protocol, VERIFY accepts with probability 1. Moreover, since $\mathbf{P} = \mathbf{M}\mathbf{G}_x$, decryption gives

$$\begin{aligned} \mathbf{y} &= \hat{\mathbf{y}} - \mathbf{P}\mathbf{e}_x = \mathbf{M}\hat{\mathbf{x}} - \mathbf{M}\mathbf{G}_x \mathbf{e}_x \\ &= \mathbf{M}(\mathbf{x} + \mathbf{G}_x \mathbf{e}_x) - \mathbf{M}\mathbf{G}_x \mathbf{e}_x = \mathbf{M}\mathbf{x}. \end{aligned}$$

For soundness, fix any unbounded adversary $\mathcal{A}$, matrix $\mathbf{M} \in \mathbb{F}^{m \times n}$, and input $\mathbf{x} \in \mathbb{F}^n$. Condition on any public parameters pp output by PREPROCESS for which the code generated by $\mathbf{G}_y$ has relative distance δ_y , and on any fresh privacy randomness $\mathbf{e}_x$ used to form $\hat{\mathbf{x}} = \mathbf{x} + \mathbf{G}_x \mathbf{e}_x$. Let $\hat{\mathbf{y}} \leftarrow \mathcal{A}(\text{pp}, \mathbf{M}, \hat{\mathbf{x}})$ be the server's response, and suppose that verification accepts. The decrypted output satisfies $\mathbf{y} = \hat{\mathbf{y}} - \mathbf{P}\mathbf{e}_x$. Using $\mathbf{P} = \mathbf{M}\mathbf{G}_x$, we have

$$\begin{aligned} \mathbf{y} - \mathbf{M}\mathbf{x} &= \hat{\mathbf{y}} - \mathbf{M}\mathbf{G}_x \mathbf{e}_x - \mathbf{M}\mathbf{x} \\ &= \hat{\mathbf{y}} - \mathbf{M}(\mathbf{x} + \mathbf{G}_x \mathbf{e}_x) \\ &= \hat{\mathbf{y}} - \mathbf{M}\hat{\mathbf{x}}. \end{aligned}$$

Therefore, if $\mathbf{y} \neq \mathbf{M}\mathbf{x}$, then $\hat{\mathbf{y}} \neq \mathbf{M}\hat{\mathbf{x}}$. By Theorem 5.2, under the stated choices of t_y and ℓ , VERIFY accepts such an incorrect response with probability at most $2^{-\kappa}$, proving soundness.

For client privacy, the simulator Sim on input $(1^\lambda, 1^\kappa, \mathbf{M})$ runs $\text{pp} \leftarrow \text{PREPROCESS}(1^\lambda, 1^\kappa, \mathbf{M})$, samples $\tilde{\mathbf{x}} \leftarrow \mathbb{F}^n$ uniformly at random, and outputs $(\text{pp}, \mathbf{M}, \tilde{\mathbf{x}})$. By the $(\mathcal{G}_x, \mathcal{E}_{t_x}^{N_x}, \mathbb{F})$ -dual-LPN assumption, $(\mathbf{G}_x, \mathbf{G}_x \mathbf{e}_x) \approx_c (\mathbf{G}_x, \mathbf{u})$, where $\mathbf{G}_x \leftarrow \mathcal{G}_x$, $\mathbf{e}_x \leftarrow \mathcal{E}_{t_x}^{N_x}$, and $\mathbf{u} \leftarrow \mathbb{F}^n$. Consequently, for any $\mathbf{x} \in \mathbb{F}^n$, $(\text{pp}, \mathbf{M}, \mathbf{x} + \mathbf{G}_x \mathbf{e}_x) \approx_c (\text{pp}, \mathbf{M}, \mathbf{x} + \mathbf{u})$. Because $\mathbf{u}$ is uniformly distributed over $\mathbb{F}^n$, the vector $\mathbf{x} + \mathbf{u}$ is also identically distributed to a uniform vector $\tilde{\mathbf{x}} \leftarrow \mathbb{F}^n$. Hence,

$$\begin{aligned} \text{View}_{\text{Server}}^\Pi(\mathbf{M}, \mathbf{x}) &= (\text{pp}, \mathbf{M}, \hat{\mathbf{x}}) \\ &\approx_c (\text{pp}, \mathbf{M}, \tilde{\mathbf{x}}) = \text{Sim}(1^\lambda, 1^\kappa, \mathbf{M}). \quad \square \end{aligned}$$

Remark 6.2 (Code failure probability in Π_{pvMVMD}). As in Remark 5.3, when the verification code $\mathbf{G}_y$ is sampled from a distribution with failure probability p_{fail} , the overall soundness error is at most $p_{\text{fail}} + 2^{-\kappa}$. Similarly, the privacy guarantee relies on the dual-LPN assumption, which requires that $\mathbf{G}_x$ generates a code with sufficient minimum distance. Both failure probabilities can be made negligible for suitable code families that we consider for instantiation.

6.3 Efficiency

We now analyze the online runtime of the client for Π_{pvMVMD} under the specified parameters above.

In Table 2, we break down the runtime of each operation performed by the client in ENCRYPT and DECRYPT.

Phase	Operation	Cost
ENCRYPT	$\mathbf{x}' := \mathbf{G}_x \mathbf{e}_x$ $\hat{\mathbf{x}} := \mathbf{x} + \mathbf{x}'$	$O(N_x)$ n
DECRYPT	$\mathbf{P}\mathbf{e}_x = \sum_{j \in \text{supp}(\mathbf{e}_x)} e_{x,j} \mathbf{P}_{*,j}$ $\mathbf{y} := \hat{\mathbf{y}} - \mathbf{P}\mathbf{e}_x$	$(2t_x - 1)m$ m
VERIFY	See Table 1	$O(N_y + \ell t_y n)$
Total	–	$O(N_x + N_y + n\ell t_y + mt_x)$

Table 2: Cost tally for the client online phase of Π_{pvMVMD} .

Since we use the VERIFY algorithm of Π_{vMVMD} (Section 5.1), VERIFY runs in time $O_\kappa(m+n)$. ENCRYPT and DECRYPT have running time $O_\lambda(m+n)$, which guarantees that the total client online time of Π_{pvMVMD} is $O_{\lambda,\kappa}(m+n)$.

Preprocessing time. The preprocessed matrix $\mathbf{P} = \mathbf{M}\mathbf{G}_x$ can be computed by realizing $\mathbf{P}^\top = \mathbf{G}_x^\top \mathbf{M}^\top$ as m encodings of the rows of $\mathbf{M}$. Similarly, $\mathbf{Q} = \mathbf{G}_y^\top \mathbf{M}$ can be realized as n encodings of the columns of $\mathbf{M}$. When $\mathbf{G}_x$ and $\mathbf{G}_y$ have linear encoding times, PREPROCESS has running time $O(mn)$.

7 Evaluations

We implement Π_{pvMVMD} and an end-to-end prototype of Maverick. In this section, we report on the implementation details and the evaluation results.

7.1 Evaluation of pvMVMD

Instantiation of codes. We implement Π_{pvMVMD} (Protocol 3) in Rust. We use the BabyBear field $\mathbb{F}_q$, where $q = 15 \cdot 2^{27} + 1$, and set the computational and statistical security parameters to $\lambda = 128$ and $\kappa = 40$, respectively.

We instantiate the privacy and verification codes using the RAA code construction of Akhiani et al. [3] (also see Appendix B), with rate $R_x = R_y = \frac{1}{8}$ and target relative distance $\delta_x = \delta_y = \frac{1}{2}$. We use RAA codes because they are linear-time encodable, field-agnostic, asymptotically good, and plausibly dual-LPN-friendly. For further discussion on these properties for RAA codes, see Appendix B.

We choose sparsity t_x according to (1): for $n \in \{2^{10}, 2^{11}, 2^{12}, 2^{13}, 2^{14}\}$ this gives $t_x \in \{160, 159, 157, 156, 154\}$, respectively. We choose sparsity parameter $t_y = 31$ and number of repetitions $\ell = 2$ per Theorem 6.1.

Experimental setup. We evaluate our pvMVMD implementation on an AWS c8i.4xlarge instance with 32 GB of memory, using a single thread. Unless otherwise noted, we measure computation time and exclude network communication. For simplicity, our experiments focus on square matrices where $m = n$.

n	Server	Client			Client speedup
		Total	Verification	Privacy	
2^8	0.14	0.19	0.09	0.09	$0.77\times$
2^9	0.30	0.32	0.18	0.13	$0.95\times$
2^{10}	0.39	0.79	0.50	0.29	$0.50\times$
2^{11}	1.59	1.55	1.09	0.46	$1.02\times$
2^{12}	9.54	2.33	1.47	0.85	$4.10\times$
2^{13}	34.68	4.56	2.51	2.05	$7.60\times$
2^{14}	116.00	9.33	4.65	4.68	$12.44\times$

Table 3: Online server and client runtime in Π_{pVMMD} , in milliseconds. The server runtime is the time required to perform the MVM on the masked vector. The client runtime is divided into verification-related and privacy-related. Client speedup is the server MVM time divided by the total client time.

Online Time. The *online runtime* refers to the runtime after preprocessing. For clients, this includes ENCRYPT, VERIFY, and DECRYPT. **Client (total)** reports the total online time, broken down into **Client (verification)** for Π_{pVMMD} .VERIFY, and **Client (privacy)** for Π_{pVMMD} .ENCRYPT and Π_{pVMMD} .DECRYPT. For servers, the online computation is the MVM on the masked input. The results are shown in Table 3.

For small matrices, the client online time is worse than the server time (i.e., outsourcing is not worthwhile). However, the advantage becomes significant as matrix dimension grows. At $n = 2^{13}$, the total client online time is 4.56 ms, consisting of 2.51 ms for verification and 2.05 ms for privacy, while the server multiplication takes 34.68 ms. Thus, the client’s online work is approximately $7.6\times$ smaller than the delegated multiplication. At $n = 2^{14}$, this gap increases to approximately $12.4\times$, consistent with linear client and quadratic server costs.

Preprocessing time. Table 4 presents the preprocessing overhead. Recall that preprocessing involves computing $\mathbf{P} := \mathbf{M}\mathbf{G}_x$ and $\mathbf{Q} := \mathbf{G}_y^\top \mathbf{M}$ where $\mathbf{M}$ is the public matrix and $\mathbf{G}_x, \mathbf{G}_y$ are sampled from code generator distributions. The computation overhead is essentially the same as encoding $\mathbf{M}$. As expected, preprocessing grows approximately quadratically with the matrix dimension because both $\mathbf{P}$ and $\mathbf{Q}$ require processing the complete matrix. At $n = 2^{14}$, preprocessing takes approximately 135.80 s, including 46.57 s for privacy-related preprocessing and 89.23 s for verification-related preprocessing.

Comparison with related work. We compare our protocol with Dumas–Zucca [34] and Sum-Check instantiated with the BaseFold polynomial commitment scheme [78, 91], where Dumas–Zucca is a specialized protocol for verifying matrix–vector products and Sum-Check+BaseFold represents the general proof-based approach used by recent verifiable-ML systems, including DeepProve [41]. Since both schemes provide verification only, we compare their runtime with our client verification time. To make a fair comparison, we benchmark over the scalar field of BLS12-381.

n	Preprocessing (total)	Preprocessing (verification)	Preprocessing (privacy)
2^8	18.76	10.72	8.04
2^9	69.04	39.08	29.97
2^{10}	329.54	194.84	134.69
2^{11}	1,499.06	945.06	554.00
2^{12}	6,030.76	3,928.78	2,101.98
2^{13}	25,063.59	16,245.40	8,818.19
2^{14}	135,800.06	89,233.69	46,566.37

Table 4: One-time preprocessing runtime for square matrices in milliseconds.

Across the evaluated matrix dimensions, our end-to-end time is between $2.5\times$ and $34.8\times$ faster than Dumas–Zucca and between $60.7\times$ and $194.8\times$ faster than Sum-Check+BaseFold. At $n = 4,096$, for example, our protocol completes in 421.943 ms, compared with 1.056 s for Dumas–Zucca and 82.196 s for Sum-Check+BaseFold. Our client verification time is also comparable to theirs. The performance numbers are detailed in Table 5.

7.2 Design and Evaluation of Maverick

In this section, we present the design of Maverick and the evaluation results using the open-source model Qwen3-4B.

7.2.1 Design of Maverick

We use the term inference execution to refer to the process of converting a sequence of input tokens into the logits for the output tokens. An inference execution involves both linear and nonlinear operations [82]. Clients in Maverick use pVMMD to outsource the MVMs in the linear operations while computing nonlinear operations locally.

As shown in Fig. 1, we denote an inference execution as a sequence of T operations $((\mathbf{M}_1, f_1), \dots, (\mathbf{M}_T, f_T))$, where $\mathbf{M}_j$ is the matrix for the linear operation, and f_j is the nonlinear function. In each step $j \in [T]$, the client outsources the computation of $\mathbf{y}_j = \mathbf{M}_j \mathbf{x}_j$ to the server, where $\mathbf{x}_j$ is a private vector computed from step $j - 1$ (with $\mathbf{x}_1$ being the client’s initial input); then the client runs the nonlinear function locally to compute $\mathbf{x}_{j+1} = f_j(\mathbf{y}_j)$, which is the input to the next step. $\mathbf{x}_{T+1}$ is the final output, the logits of the next token.

To leverage batch verification, we defer verification of all server response pairs $(\hat{\mathbf{x}}_j, \hat{\mathbf{y}}_j)$ to the end, a strategy we call *optimistic verification*. The client accepts the logits only if every check accepts; otherwise, it discards the execution.

Boost mode. Generating input masks and computing the corresponding output masks are a significant part of client overhead. Since these masks are independent of the model or the client input, they can be generated beforehand (e.g., when the client’s device is idle). This optimization is similar

n	Scheme	Preprocessing	Server multiplication	Server proof	Server total	Client online	Client verification
2^7	Ours	23.16	0.39	–	0.39	0.88	0.23
	Dumas–Zucca	70.24	0.35	20.82	21.17	23.14	23.14
	Sum-Check+BaseFold	101.80	0.36	74.92	75.29	1.99	1.99
2^8	Ours	85.22	1.51	–	1.51	1.81	0.49
	Dumas–Zucca	114.19	1.40	39.80	41.19	28.30	28.30
	Sum-Check+BaseFold	428.44	1.47	300.14	301.61	2.48	2.48
2^9	Ours	316.83	6.12	–	6.12	3.89	1.16
	Dumas–Zucca	195.50	5.59	75.82	81.41	35.88	35.88
	Sum-Check+BaseFold	1,843.15	6.58	1,229.57	1,236.15	3.29	3.29
2^{10}	Ours	1,275.83	24.08	–	24.08	7.14	1.64
	Dumas–Zucca	351.70	22.37	160.90	183.27	48.32	48.32
	Sum-Check+BaseFold	8,069.29	26.80	5,051.12	5,077.93	4.14	4.14
2^{11}	Ours	5,890.08	97.23	–	97.23	14.74	3.20
	Dumas–Zucca	685.06	89.44	308.67	398.11	60.67	60.67
	Sum-Check+BaseFold	34,915.39	107.84	20,521.54	20,629.39	5.07	5.07
2^{12}	Ours	27,339.81	392.26	–	392.26	29.68	6.76
	Dumas–Zucca	1,453.98	359.55	616.93	976.47	79.30	79.30
	Sum-Check+BaseFold	150,894.69	433.04	81,756.33	82,189.37	6.39	6.39

Table 5: Comparison with verifiable matrix–vector multiplication baselines. All times are in milliseconds. Our client online time includes both privacy and verification, where client verification reports only the verification component.

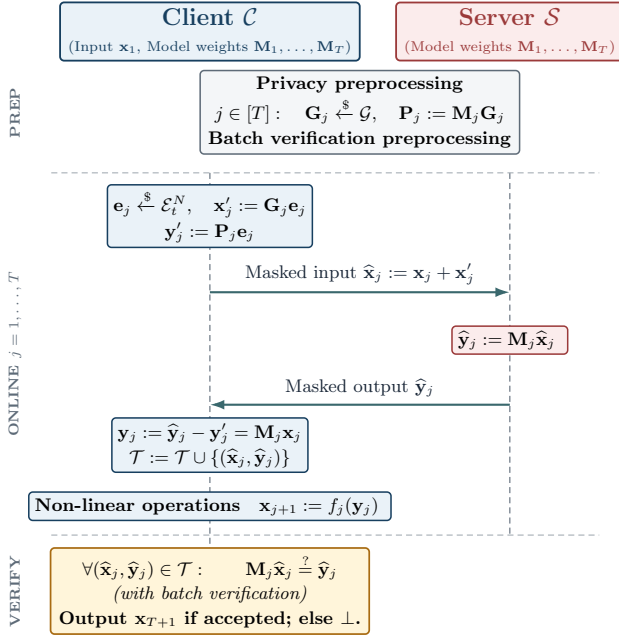


Figure 1: Private and verifiable LLM inference using Maverick. The client locally evaluates the nonlinear operations and delegates the linear operations $\mathbf{M}_1, \dots, \mathbf{M}_T$ to the server. The output is the logits of the next token.

to how MPC protocols generate Beaver Triples ahead of time to accelerate the online phase. We call this the *boost mode* of Maverick. We refer to the original workflow where the client generates these masks during inference as the *standard mode*; we implement *pipelining* so the masks for step $j + 1$ are generated while waiting for the server to complete step j .

Verification-only mode. Some users may not always require input privacy, for example when their prompts contain no sensitive information or when they are willing to share them with the service provider, but may still want assurance that the provider performs the inference correctly. In this case, the protocol can be simplified and, importantly, the interaction with the server can be removed. The client first sends the model input to the server. The server evaluates the complete model, including the nonlinear operations, and then sends the output vectors $\mathbf{y}_1, \dots, \mathbf{y}_T$ back to the client. Starting from the original input, the client replays the nonlinear operations locally to reconstruct each input $\mathbf{x}_j$ and verifies $\mathbf{M}_j \mathbf{x}_j \stackrel{?}{=} \mathbf{y}_j$ together. Thus, the inference itself requires only one round of interaction and batch verification. The protocol is illustrated in Fig. 2.

Model quantization. In Maverick, all arithmetic operations are done in the BabyBear field. To get a fair baseline, we run a quantized Qwen3–4B model over the BabyBear field $\mathbb{F}_q$ as follows. The model parameters are stored in FP8 format. Let W_{real} denote the weight matrix in FP8. We quantize it element-wise as $W_q := \text{round}(2^s W_{\text{real}})$, rounding to the nearest

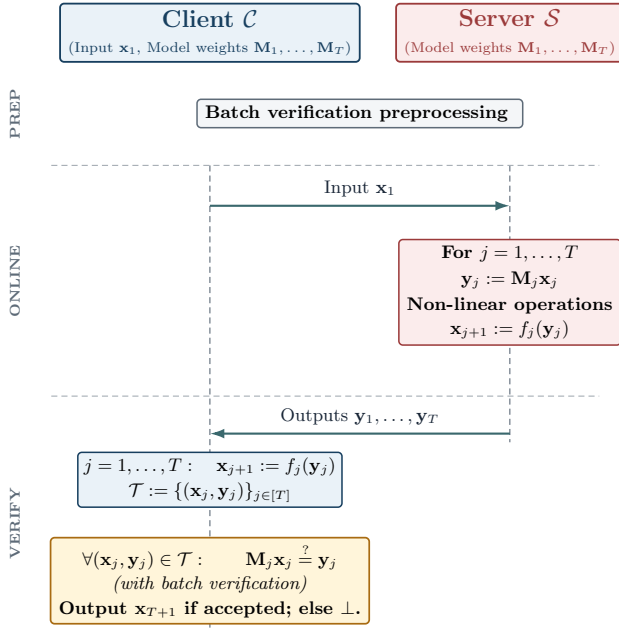


Figure 2: Maverick in the verification-only mode.

integer. We embed the resulting signed integers into $\mathbb{F}_q$ and use $s = 8$. After each delegated linear operation, the client demasks the result, interprets it as a centered integer, divides it by 2^{2s} , and converts it to BF16 before evaluating the next nonlinear operation. The protocol performs the MVM over $\mathbb{F}_q$, and we observe no modular overflow in our experiments. We omit these representation conversions from Fig. 1 for clarity. This quantization methodology is consistent with how related works execute models in finite fields, such as DeepProve [41] and Slalom [80].

7.2.2 Evaluation of Maverick

Experimental setup. We evaluate all three modes of Maverick using the open-source model Qwen3-4B [77]. The dimensions of the linear operations in each transformer block are shown in Table 6. Our inference execution takes an eight-token prompt as input and outputs the logits for the next token.

The client runs on an AWS c8i.4xlarge instance, while the server runs on an AWS c8i.32xlarge instance. Our baseline is the client running the inference locally.

We organize our evaluation in two stages. First, we measure the end-to-end throughput, excluding network latency, to quantify the client speedup relative to local inference. Second, we evaluate client throughput under simulated network latency, as in practice the client is the bottleneck. For this experiment, we assume a powerful enough server, allowing us to isolate the maximum throughput sustained by the client.

Linear Operation	Input dimension	Output dimension
q_proj	2560	4096
k_proj	2560	1024
v_proj	2560	1024
o_proj	4096	2560
gate_proj	2560	9728
up_proj	2560	9728
down_proj	9728	2560

Table 6: Dimensions of the linear operations in each Qwen transformer block.

End-to-end throughput. We measure the speedup of Maverick relative to running the inference entirely locally on the client. Figure 3 compares standard, boost, and verification-only modes across different client and server thread counts. Communication time is excluded.

All three modes provide substantial speedups. The benefit is largest for resource-constrained clients and decreases as more client threads become available, since local inference also benefits substantially from additional client-side parallelism and has better parallelism than our client protocol with more threads. Standard mode achieves between $8.08\times$ and $17.62\times$ speedup, while boost mode reaches up to $45.85\times$ by moving privacy mask generation time outside the online time. Verification-only mode behaves similarly to boost mode for clients with few threads, reaching $44.38\times$.

Increasing the server thread count from 32 to 64 generally improves the speedup, while increasing it further to 128 provides no consistent benefit. This suggests that the current implementation reaches its parallelization limit near 64 server threads. We therefore fix 64 server threads and 2 client threads, and report the end-to-end throughput and time breakdown in Table 7. Our end-to-end time measures the time from when the client processes the input tokens to when the client accepts the verification and outputs the next-token logits. We note that the client and server computation may overlap.

Metric	Standard	Boost	Verification-only
Throughput (tokens/s)	8.35	15.46	15.05
End-to-end time	958.3	517.6	531.6
Client time	856.6	227.4	206.8
Client verification	45.5	46.5	45.1
Client nonlinear operations	50.5	53.7	30.6
Server computation	353.5	341.6	324.8
Local client inference	14110.9	14110.9	14110.9

Table 7: Online time breakdown in milliseconds for one eight-token prompt, using 2 client threads and 64 server threads.

In standard mode, Maverick completes the inference in 958.3 ms and reaches a throughput of 8.35 tokens/s, giving a $14.7\times$ speedup over the 0.57 tokens/s local baseline. The client remains the main bottleneck because privacy masks are generated during inference. Boost mode achieves a $27.3\times$

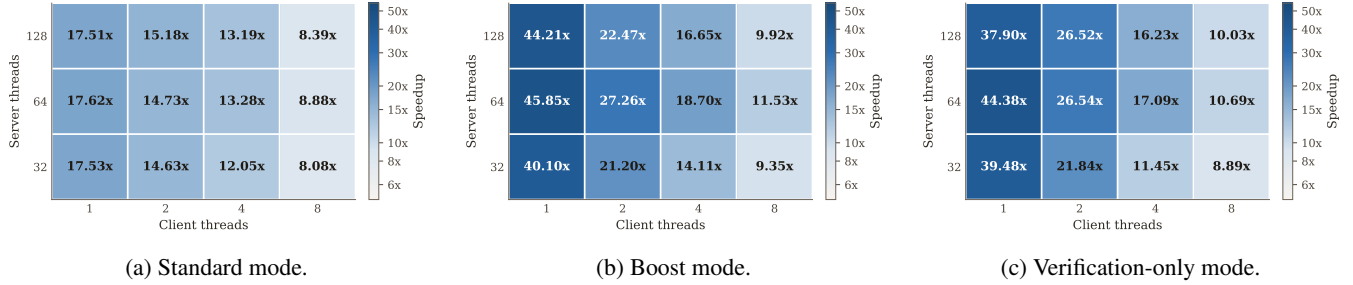


Figure 3: Client speedup of Maverick relative to local inference across different client and server thread numbers.

speedup over local inference, and verification-only mode has similar end-to-end performance as boost mode. In both boost and verification-only modes, the server computation becomes the bottleneck rather than the client. Across all three modes, client verification takes only approximately 45 ms. In Table 8, we also provide the end-to-end results of 64 fixed server threads and varying the number of client threads.

Communication. Standard and boost modes have identical communication. In both modes, the client sends 21.83 MB and receives 36.00 MB, for 57.83 MB in total over 182 rounds of communication. Verification-only mode sends only 81.92 kB and receives the 36.00 MB transcript, totaling 36.08 MB in a single round.

Comparison with DeepProve. DeepProve [41] is a recent proof-based system for verifiable LLM inference (though it does not protect input privacy). The authors evaluate 124M-parameter GPT-2 and 270M-parameter Gemma 3 models with 12-bit quantization at input lengths between 64 and 512. On its faster AMD Ryzen 9 7950X3D machine with 16 cores and 32 threads, its faster BaseFold instantiation at 64 input tokens requires 34.2 s to prove and 1.35 s to verify GPT-2, and 81.6 s to prove and 1.91 s to verify Gemma 3. Furthermore, DeepProve reports proof sizes between 9 and 43 MiB.

By comparison, our verification-only mode evaluates an eight-token prompt using the substantially larger Qwen3-4B model in only 531.6 ms, including the model computation, with 45.1 ms of client verification with only 36 MB of communication. Although these measurements are not apples-to-apples as DeepProve’s experiments use different hardware and LLM, we note that relative to DeepProve’s 64 input token results, our end-to-end time is about $64\times$ lower than that of GPT-2 and $153\times$ lower than that of Gemma 3, while our verification time is about $30\times$ and $42\times$ lower, respectively. These gaps are particularly notable since Maverick is evaluated on the substantially larger Qwen3-4B model.

Client throughput. The end-to-end results show that the performance bottleneck depends on the execution mode. Standard mode has the client as the bottleneck, while boost and verification-only modes shift more work to the server so the bottleneck becomes the server. In a deployed service, however, the provider can increase server capacity using better

hardware like GPUs, whereas the computational resources of the client are fixed. We therefore next measure the maximum throughput that one client can sustain, assuming a powerful server that responds in negligible time.

We use microbenchmarks to evaluate the standard, boost, and verification-only modes of Maverick. We measure throughput under simulated network RTTs to account for communication delays in realistic deployments. Steady-state *throughput* measures the aggregate number of prompt tokens processed per second. For each mode and client thread count, we use a pilot run to find the smallest number of input tokens whose client computation can cover one network RTT. Table 9 reports the corresponding number of input tokens and the sustained client throughput.

Doubling the RTT approximately doubles the number of input tokens required to hide network waiting, but changes throughput by less than 0.6%. Standard mode achieves an $11.73\times$ – $19.81\times$ throughput improvement over local inference. Moving mask generation offline increases the boost-mode improvement to $33.97\times$ – $135.33\times$, while verification-only mode achieves $37.59\times$ – $156.68\times$. The larger speedups with fewer client threads reflect that the client in our protocol is less parallelizable than local inference.

7.3 TTFT of Maverick

We measure TTFT (time to first token) using one eight-token input with 2 client threads and 64 server threads. Network latency is varied between 50 ms and 100 ms RTT, while Maverick is run in standard, boost, and verification-only mode.

The timer starts when the input enters the online protocol and ends when the next-token logits are available and the verification is accepted. The corresponding local execution takes 14.11 s. We report the detailed TTFT numbers in Table 10.

Communication dominates the TTFT in the standard mode and the boost mode because both require multiple rounds of communication. Standard and boost modes are respectively $1.42\times$ and $1.46\times$ faster than local inference at an RTT of 50 ms. At an RTT of 100 ms, however, communication makes both modes slower than local inference.

Verification-only mode requires one round of communi-

Client threads	Mode	Throughput (tokens/s)	End-to-end time (ms)	Client time (ms)	Client verification (ms)	Client nonlinear operations (ms)	Server computation (ms)	Local client inference (ms)
1	Standard	5.17	1547.7	1460.5	89.6	58.6	363.3	27278.6
	Boost	13.45	594.9	292.4	91.8	63.2	349.7	27278.6
	Verification-only	13.01	614.7	260.2	87.1	37.4	354.5	27278.6
2	Standard	8.35	958.3	856.6	45.5	50.5	353.5	14110.9
	Boost	15.46	517.6	227.4	46.5	53.7	341.6	14110.9
	Verification-only	15.05	531.6	206.8	45.1	30.6	324.8	14110.9
4	Standard	12.64	633.0	533.6	23.6	34.4	345.8	8407.0
	Boost	17.80	449.5	173.5	23.8	42.6	347.2	8407.0
	Verification-only	16.27	491.8	173.5	22.5	26.9	318.3	8407.0
8	Standard	14.55	550.0	433.6	17.7	22.4	345.1	4885.9
	Boost	18.88	423.7	159.2	15.6	34.8	346.4	4885.9
	Verification-only	17.51	456.9	142.5	12.8	17.4	314.4	4885.9

Table 8: Online computation breakdown for an eight-token prompt with 64 server threads.

Mode	Client threads	RTT = 50 ms		RTT = 100 ms		Local throughput (tokens/s)	Speedup over local (50/100 ms)
		Input tokens	Throughput (tokens/s)	Input tokens	Throughput (tokens/s)		
Standard	1	56	5.78	112	5.81	0.29	$19.71 \times / 19.81 \times$
	2	96	9.82	184	9.84	0.57	$17.32 \times / 17.36 \times$
	4	144	15.64	288	15.66	0.95	$16.44 \times / 16.46 \times$
	8	176	19.20	352	19.22	1.64	$11.73 \times / 11.74 \times$
Boost	1	368	39.49	728	39.69	0.29	$134.65 \times / 135.33 \times$
	2	408	43.99	808	44.10	0.57	$77.59 \times / 77.79 \times$
	4	488	53.28	976	53.35	0.95	$55.99 \times / 56.06 \times$
	8	512	55.63	1,016	55.68	1.64	$33.97 \times / 34.00 \times$
Verification-only	1	368	45.70	728	45.95	0.29	$155.82 \times / 156.68 \times$
	2	408	49.21	808	49.34	0.57	$86.80 \times / 87.03 \times$
	4	488	52.85	976	52.92	0.95	$55.54 \times / 55.61 \times$
	8	512	61.55	1,016	61.60	1.64	$37.59 \times / 37.62 \times$

Table 9: Client throughput under simulated network latency.

cation. Its TTFT is 0.66 s and 0.71 s at RTTs of 50 ms and 100 ms, respectively. This is $15.0\times$ and $26.8\times$ faster than standard mode, and $21.4\times$ and $19.9\times$ faster than local execution. Thus, this mode is less latency-sensitive.

Acknowledgements

Wenhao Wang is supported in part by a research grant from IC3.

8 Conclusions and Future Directions

We develop Maverick, a protocol for private and verifiable LLM inference that delegates the model’s linear operations using a new matrix–vector multiplication primitive with transparent preprocessing and efficient client verification. Our implementation on Qwen3–4B demonstrates substantial speedups. An important direction for future work is reducing the communication rounds for private inference.

RTT	Mode	TTFT (s)	Communication wait (s)	Client work (s)	Server computation (s)	Verification (s)
50 ms	Standard	9.92	9.13	0.86	0.35	0.05
	Boost	9.65	9.13	0.23	0.34	0.05
	Verification-only	0.66	0.05	0.21	0.40	0.05
100 ms	Standard	19.02	18.24	0.86	0.35	0.05
	Boost	18.75	18.23	0.23	0.34	0.05
	Verification-only	0.71	0.10	0.21	0.40	0.05

Table 10: Time to first token for one eight-token request under simulated network delay, using 2 client threads and 64 server threads.

References

- [1] Kasra Abbaszadeh, Hossein Hafezi, Jonathan Katz, and Sarah Meiklejohn. Single-server private outsourcing of zk-snarks. *Cryptology ePrint Archive*, 2025.
- [2] Carlos Aguilar-Melchor, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, and Gilles Zémor. Efficient encryption from random quasi-cyclic codes. *IEEE Transactions on Information Theory*, 64:3927–3943, 2016.
- [3] Pariya Akhiani and Yupeng Zhang. Distance of RAA codes over large finite fields (with applications in zk-SNARKs and PCGs). *Cryptology ePrint Archive*, Paper 2026/524, 2026.
- [4] Michael Alekhnovich. More on average case vs approximation complexity. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, FOCS '03*, page 298, USA, 2003. IEEE Computer Society.
- [5] Sarah Allen, Pranay Anchuri, James Austgen, Maryam Bahrani, Samuel Breckenridge, Aaron Buchwald, Christian Cachin, Andrés Fábrega, Jared Fernandez, James Hsin yu Chiang, Marwa Mouallem, Roi Bar-Zur, Neil DeSilva, Ittay Eyal, Giulia Fanti, Ari Juels, Andrew Miller, Christian Sillaber, Dani Vilardell, Pramod Viswanath, Wenhao Wang, Matt Weinberg, Sen Yang, Jianzhu Yao, and Fan Zhang. Crypto x AI, AI x crypto: A survey. Technical report, Initiative for CryptoCurrencies and Contracts (IC3), June 2026. Version 1.0.
- [6] Anthropic and Pattern Labs. Confidential inference systems: Design principles and security risks. White paper, Anthropic and Pattern Labs, June 2025. Version 1.0.
- [7] Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Santosh Ghosh, Shay Gueron, Tim Güneysu, et al. Bike: bit flipping key encapsulation, 2022.
- [8] Diego F. Aranha, Anamaria Costache, Antonio Guimarães, and Eduardo Soria-Vazquez. Heliopolis: Verifiable computation over homomorphically encrypted data from interactive oracle proofs is practical. In Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology – ASIACRYPT 2024*, pages 302–334, Singapore, 2025. Springer Nature Singapore.
- [9] Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In Luca Aceto, Monika Henzinger, and Jiří Sgall, editors, *Automata, Languages and Programming*, pages 403–415, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
- [10] David Balbás, Dario Fiore, Maria Isabel González Vasco, Damien Robissout, and Claudio Soriente. Modular sum-check proofs with applications to machine learning and image processing. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS '23*, page 1437–1451, New York, NY, USA, 2023. Association for Computing Machinery.
- [11] Louay Bazzi, Mohammad Mahdian, and Daniel A. Spielman. The minimum distance of turbo-like codes. *IEEE Transactions on Information Theory*, 55(1):6–15, 2009.
- [12] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 420–432, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.
- [13] Fabrice Benhamouda, Caicai Chen, Shai Halevi, Yuval Ishai, Hugo Krawczyk, Tamer Mour, Tal Rabin, and Alon Rosen. Encrypted matrix-vector products from secret dual codes. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security, CCS '25*, page 394–408, New York, NY, USA, 2025. Association for Computing Machinery.

- [14] Huck Bennett, Karthik Gajulapalli, Alexander Golovnev, and Evelyn Warton. Matrix Multiplication Verification Using Coding Theory. In Amit Kumar and Noga Ron-Zewi, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2024)*, volume 317 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 42:1–42:13, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- [15] Wyatt Benno, Alberto Centelles, Antoine Douchet, and Khalil Gibran. Jolt atlas: Verifiable inference via lookup arguments in zero knowledge, 2026.
- [16] Avrim Blum, Merrick Furst, Michael Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In Douglas R. Stinson, editor, *Advances in Cryptology — CRYPTO’ 93*, pages 278–291, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.
- [17] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM*, 50(4):506–519, July 2003.
- [18] J. L. Bordewijk. Inter-reciprocity applied to electrical networks. *Applied Scientific Research B: Electrophysics, Acoustics, Optics, Mathematical Methods*, 6:1–74, 1956. URL: <http://cr.ypt.to/bib/entries.html#1956/bordewijk>.
- [19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector ole. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS ’18*, page 896–912, New York, NY, USA, 2018. Association for Computing Machinery.
- [20] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. Correlated pseudorandomness from expand-accumulate codes. In *Advances in Cryptology – CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part II*, page 603–633, Berlin, Heidelberg, 2022. Springer-Verlag.
- [21] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent ot extension and more. In *Advances in Cryptology – CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III*, page 489–518, Berlin, Heidelberg, 2019. Springer-Verlag.
- [22] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudo-random correlation generators from ring-lpn. In *Advances in Cryptology – CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part II*, page 387–416, Berlin, Heidelberg, 2020. Springer-Verlag.
- [23] Mark Braverman and Stephen Newman. Practical secure delegated linear algebra with trapdoored matrices. In *Theory of Cryptography Conference*, pages 97–118. Springer, 2025.
- [24] Martijn Brehm, Binyi Chen, Ben Fisch, Nicolas Resch, Ron D. Rothblum, and Hadas Zeilberger. Blaze: Fast snarks from interleaved raa codes. In *Advances in Cryptology – EUROCRYPT 2025: 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4–8, 2025, Proceedings, Part IV*, page 123–152, Berlin, Heidelberg, 2025. Springer-Verlag.
- [25] Ida Bruhns, Sebastian Berndt, Jonas Sander, and Thomas Eisenbarth. Slalom at the carnival: Privacy-preserving inference with masks from public knowledge. *IACR Communications in Cryptology*, 1(3), 2024.
- [26] Ignacio Cascudo, Anamaria Costache, Daniele Cozzo, Dario Fiore, Antonio Guimarães, and Eduardo Soria-Vazquez. Verifiable computation for approximate homomorphic encryption schemes. In *Annual International Cryptology Conference*, pages 643–677. Springer, 2025.
- [27] Center for Research on Foundation Models. The foundation model transparency index.
- [28] Sylvain Chatel, Christian Knabenhans, Apostolos Pyrgelis, Carmela Troncoso, and Jean-Pierre Hubaux. Veritas: Plaintext encoders for practical verifiable homomorphic encryption. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS ’24*, page 2520–2534, New York, NY, USA, 2024. Association for Computing Machinery.
- [29] Lingjiao Chen, Matei Zaharia, and James Zou. How is chatgpt’s behavior changing over time? *arXiv preprint arXiv:2307.09009*, 2023.
- [30] Jung Hee Cheon and Daehyun Jang. Cryptanalysis on lightweight verifiable homomorphic encryption. In Goichiro Hanaoka and Bo-Yin Yang, editors, *Advances in Cryptology – ASIACRYPT 2025*, pages 366–397, Singapore, 2026. Springer Nature Singapore.
- [31] Jalen Chuang, Alex Seto, Nicolas Berrios, Stephan Van Schaik, Christina Garman, and Daniel Genkin. TEE.Fail:

- Breaking Trusted Execution Environments via DDR5 Memory Bus Interposition. In *2026 IEEE Symposium on Security and Privacy (SP)*, pages 4408–4426. IEEE Computer Society, May 2026.
- [32] DeepSeek-AI. DeepSeek-V3 technical report, 2024.
- [33] Design Arena. Kimi K3’s design secret may be in its thinking traces. The Intelligence Company, jul 2026.
- [34] Jean-Guillaume Dumas and Vincent Zucca. Prover efficient public verification of dense or sparse/structured matrix-vector multiplication. In Josef Pieprzyk and Suriadi Suriadi, editors, *Information Security and Privacy*, pages 115–134, Cham, 2017. Springer International Publishing.
- [35] duola15. Deploy guide & hardware requirements for Kimi-K3. Hugging Face discussion, 2026.
- [36] Boyuan Feng, Lianke Qin, Zhenfei Zhang, Yufei Ding, and Shumo Chu. ZEN: An optimizing compiler for verifiable, zero-knowledge neural network inferences. Cryptology ePrint Archive, Paper 2021/087, 2021.
- [37] Dario Fiore and Rosario Gennaro. Publicly verifiable delegation of large polynomials and matrix computations, with applications. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security, CCS ’12*, page 501–512, New York, NY, USA, 2012. Association for Computing Machinery.
- [38] Dario Fiore, Rosario Gennaro, and Valerio Pastro. Efficiently verifiable computation on encrypted data. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS ’14*, page 844–855, New York, NY, USA, 2014. Association for Computing Machinery.
- [39] Dario Fiore, Anca Nitulescu, and David Pointcheval. Boosting verifiable computation on encrypted data. In *Public-Key Cryptography – PKC 2020: 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4–7, 2020, Proceedings, Part II*, page 124–154, Berlin, Heidelberg, 2020. Springer-Verlag.
- [40] Rusins Freivalds. Probabilistic machines can use less running time. In *IFIP Congress*, 1977.
- [41] Nicolas Gailly, Ismael Hishon-Rezaizadeh, Tianyi Liu, Nicholas Mainardi, Dimitrios Papadopoulos, Charalampos Papamanthou, Christodoulos Pappas, Shravan Srinivasan, Zack Youell, and Yupeng Zhang. DeepProve: Verifiable end-to-end large language model inference. Cryptology ePrint Archive, Paper 2026/1112, 2026.
- [42] Zahra Ghodsi, Tianyu Gu, and Siddharth Garg. SafetyNets: Verifiable execution of deep neural networks on an untrusted cloud. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.
- [43] Ran Gilad-Bachrach, Nathan Dowlan, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 201–210, New York, New York, USA, 20–22 Jun 2016. PMLR.
- [44] Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, Ashish Panwar, and Rahul Sharma. SIGMA: Secure GPT inference with function secret sharing. Cryptology ePrint Archive, Paper 2023/1269, 2023.
- [45] Kanav Gupta, Jonathan Katz, and Ian Miers. Private and verifiable outsourcing of open-weight LLM inference. Cryptology ePrint Archive, Paper 2026/1849, 2026.
- [46] Venkatesan Guruswami, Atri Rudra, and Madhu Sudan. Essential coding theory, 2025.
- [47] Meng Hao, Hongwei Li, Hanxiao Chen, Pengzhi Xing, Guowen Xu, and Tianwei Zhang. Iron: Private inference on transformers. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 15718–15731, 2022.
- [48] Jiaxing He, Kang Yang, Guofeng Tang, Zhangjie Huang, Li Lin, Changzheng Wei, Ying Yan, and Wei Wang. Rhombus: Fast homomorphic matrix-vector multiplication for secure two-party inference. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS ’24*, page 2490–2504, New York, NY, USA, 2024. Association for Computing Machinery.
- [49] Zhicong Huang, Wen jie Lu, Cheng Hong, and Jian-sheng Ding. Cheetah: Lean and fast secure Two-Party deep neural network inference. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 809–826, Boston, MA, August 2022. USENIX Association.
- [50] Alexandre Graell i Amat and Eirik Rosnes. On the minimum distance properties of weighted nonbinary repeat multiple-accumulate codes. In *2011 Information Theory and Applications Workshop*, pages 1–5, 2011.
- [51] Wen jie Lu, Zhicong Huang, Zhen Gu, Jingyu Li, Jian Liu, Cheng Hong, Kui Ren, Tao Wei, and Wenguang Chen. BumbleBee: Secure two-party inference framework for large transformers. In *32nd Annual Network*

- and Distributed System Security Symposium (NDSS), 2025.
- [52] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In *27th USENIX Security Symposium (USENIX Security)*, pages 1651–1669, 2018.
 - [53] Andes Y. L. Kei and Sherman S. M. Chow. Shaft: Secure, handy, accurate and fast transformer inference. *IACR Cryptol. ePrint Arch.*, 2025:2324, 2025.
 - [54] Majid Khabbazzian. Sharp minimum-distance lower tails for RAA codes. *Cryptology ePrint Archive*, Paper 2026/1075, 2026.
 - [55] Yongsang Kim, Kyungwhoon Cheun, and Hyuntack Lim. Performance of weighted nonbinary repeat-accumulate codes over $gf(q)$ with q -ary orthogonal modulation. *IEEE Transactions on Communications*, 59(5):1208–1212, 2011.
 - [56] Kimi Team. Kimi K2: Open agentic intelligence, 2025.
 - [57] Jörg Kliewer, Kamil Sh. Zigangirov, Christian Koller, and Daniel J. Costello Jr. Coding theorems for repeat multiple accumulate codes. *CoRR*, abs/0810.3422, 2008.
 - [58] Kyeongtae Lee, Byeongkyu Han, Jihye Kim, and Hyunok Oh. LAMP: Linear verification of matrix multiplication via proximity testing. *Cryptology ePrint Archive*, Paper 2026/1571, 2026.
 - [59] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS '17*, page 619–631, New York, NY, USA, 2017. Association for Computing Machinery.
 - [60] Jing Liu and Liang Feng Zhang. Privacy-preserving and publicly verifiable matrix multiplication. *IEEE Transactions on Services Computing*, 16(3):2059–2071, 2023.
 - [61] Tianyi Liu, Xiang Xie, and Yupeng Zhang. zkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 2968–2985, 2021.
 - [62] Hidde Lycklama, Alexander Viand, Nikolay Avramov, Nicolas Küchler, and Anwar Hithnawi. Artemis: Efficient commit-and-prove SNARKs for zkML. In *34th USENIX Security Symposium*, 2025.
 - [63] James Massey. Shift-register synthesis and bch decoding. *IEEE transactions on Information Theory*, 15(1):122–127, 1969.
 - [64] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2505–2522. USENIX Association, August 2020.
 - [65] Payman Mohassel and Yupeng Zhang. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 19–38, Los Alamitos, CA, USA, May 2017. IEEE Computer Society.
 - [66] Qi Pang, Jinhao Zhu, Helen Möllering, Wenting Zheng, and Thomas Schneider. BOLT: Privacy-preserving, accurate and efficient inference for transformers. In *2024 IEEE Symposium on Security and Privacy (S&P)*, pages 4753–4771, 2024.
 - [67] Kabir Peshawaria, Zeyu Liu, Ben Fisch, and Eran Tromer. Laminar: Succinct SIMD-friendly verifiable FHE. *Cryptology ePrint Archive*, Paper 2025/2285, 2025.
 - [68] E. Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5):5–9, 1962.
 - [69] Wenjie Qu, Yanpei Guo, Yue Ying, and Jiaheng Zhang. VerfCNN, Optimal Complexity zkSNARK for Convolutional Neural Networks. In *2026 IEEE Symposium on Security and Privacy (SP)*, pages 515–532, Los Alamitos, CA, USA, May 2026. IEEE Computer Society.
 - [70] Wenjie Qu, Yijun Sun, Xuanming Liu, Tao Lu, Yanpei Guo, Kai Chen, and Jiaheng Zhang. zkGPT: An efficient non-interactive zero-knowledge proof framework for LLM inference. In *34th USENIX Security Symposium (USENIX Security 25)*, pages 2045–2063, 2025.
 - [71] Srinivasan Raghuraman, Peter Rindal, and Titouan Tanguy. Expand-convolute codes for pseudorandom correlation generators from lpn. In *Advances in Cryptology – CRYPTO 2023: 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20–24, 2023, Proceedings, Part IV*, page 602–632, Berlin, Heidelberg, 2023. Springer-Verlag.
 - [72] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow2: Practical 2-party secure inference. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, CCS '20*, page 325–342, New York, NY, USA, 2020. Association for Computing Machinery.

- [73] Chiara Ravazzi and Fabio Fagnani. Spectra and minimum distances of repeat multiple-accumulate codes. *IEEE Trans. Inf. Theor.*, 55(11):4905–4924, November 2009.
- [74] RedPill AI. Verify real end-to-end private AI. RedPill AI, 2026. Accessed August 3, 2026.
- [75] Muhammad Husni Santriaji, Jiaqi Xue, Yancheng Zhang, Qian Lou, and Yan Solihin. DataSeal: Ensuring the Verifiability of Private Computation on Encrypted Data. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 2378–2394, Los Alamitos, CA, USA, May 2025. IEEE Computer Society.
- [76] Haochen Sun, Jason Li, and Hongyang Zhang. zkLLM: Zero knowledge proofs for large language models. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 4405–4419, 2024.
- [77] Qwen Team. Qwen3 technical report, 2025.
- [78] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. *CoRR*, abs/1304.3812, 2013.
- [79] Louis Tremblay Thibault, Michael Walter, and Jiapeng Zhang. Practical SNARGs for matrix multiplications over encrypted data. *Cryptology ePrint Archive*, Paper 2026/027, 2026.
- [80] Florian Tramer and Dan Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. In *International Conference on Learning Representations*, 2019.
- [81] Vinod Vaikuntanathan and Or Zamir. Improving algorithmic efficiency using cryptography: Trapdoored matrices and applications. In *Proceedings of the 2026 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2554–2574. SIAM, 2026.
- [82] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [83] Alexander Viand, Christian Knabenhans, and Anwar Hithnawi. Verifiable fully homomorphic encryption. *arXiv:2301.07041*, 2023.
- [84] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *2021 IEEE Symposium on Security and Privacy (S&P)*, pages 1074–1091, 2021.
- [85] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. Mystique: Efficient conversions for zero-knowledge proofs with applications to machine learning. In *30th USENIX Security Symposium (USENIX Security)*, pages 501–518, 2021.
- [86] Chenkai Weng, Kang Yang, Zhaomin Yang, Xiang Xie, and Xiao Wang. AntMan: Interactive zero-knowledge proofs with sublinear communication. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 2901–2914, 2022.
- [87] Tianshi Xu, Wen-jie Lu, Jiangrui Yu, Yi Chen, Chenqi Lin, Runsheng Wang, and Meng Li. Breaking the layer barrier: remodeling private transformer inference with hybrid ckks and mpc. In *Proceedings of the 34th USENIX Conference on Security Symposium, SEC ’25*, USA, 2025. USENIX Association.
- [88] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 2986–3001, 2021.
- [89] Kyeongcheol Yang. Weighted nonbinary repeat-accumulate codes. *IEEE Transactions on Information Theory*, 50(3):527–531, 2004.
- [90] James Hsin yu Chiang, Sheila Zingg, Kari Kostinen, and Srdjan Capkun. Mosaic: Masked outsourcing of secure ai computations, 2026.
- [91] Hadas Zeilberger, Binyi Chen, and Ben Fisch. Basefold: Efficient field-agnostic polynomial commitment schemes from foldable codes. In *Advances in Cryptology – CRYPTO 2024: 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2024, Proceedings, Part X*, page 138–169, Berlin, Heidelberg, 2024. Springer-Verlag.
- [92] Jiawen Zhang, Xinpeng Yang, Lipeng He, Kejia Chen, Wen jie Lu, Yinghao Wang, Xiaoyang Hou, Jian Liu, Kui Ren, and Xiaohu Yang. Secure transformer inference made non-interactive. *Cryptology ePrint Archive*, Paper 2024/136, 2024.
- [93] Liang Zhang and Reihaneh Safavi-Naini. Protecting data privacy in publicly verifiable delegation of matrix and polynomial functions. *Designs, Codes and Cryptography*, 88, 04 2020.
- [94] Xinxuan Zhang, Ruida Wang, Zeyu Liu, Binwu Xiang, Yi Deng, Ben Fisch, and Xianhui Lu. Phalanx: An efficient snark for verifiable computation on encrypted

data. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security, CCS '25*, page 4035–4048, New York, NY, USA, 2025. Association for Computing Machinery.

A Learning Parity with Noise

We begin by defining the dual-LPN assumption over a field $\mathbb{F}$ with respect to a distribution of code generators $\mathcal{G}$, dimension n , block length $N > n$, and noise distribution $\mathcal{E}$.

Definition A.1 (Dual-LPN). *The $(\mathcal{E}, \mathcal{G}, \mathbb{F})$ -dual-LPN assumption for dimensions $n = n(\lambda)$ and $N = N(\lambda)$ states that*

$$(\mathbf{G}, \mathbf{G}\mathbf{e}) \approx_c (\mathbf{G}, \mathbf{u})$$

where $\mathbf{G} \leftarrow \mathcal{G}$, $\mathbf{e} \leftarrow \mathcal{E}$, and $\mathbf{u} \leftarrow \mathbb{F}^n$. Here, $\mathcal{G}$ samples code generators $\mathbf{G} \in \mathbb{F}^{n \times N}$ and $\mathcal{E}$ samples noise vectors $\mathbf{e} \in \mathbb{F}^N$. Further, we say $\mathcal{G}$ is LPN-friendly if this assumption is believed to hold under standard noise distributions $\mathcal{E}$.

Observe that $\mathbf{G}\mathbf{e}$ is not a *codeword*, but is instead a *syndrome* of the dual code. As such, the search problem version of dual-LPN is also referred to as *syndrome decoding* [20].

Equivalence to (primal) LPN. The dual-LPN assumption is equivalent to the LPN assumption [16], which informally states that

$$(\mathbf{H}, \mathbf{H}\mathbf{s} + \mathbf{e}) \approx_c (\mathbf{H}, \mathbf{u})$$

where $\mathbf{H} \in \mathbb{F}^{N \times n'}$ is a parity-check matrix for the code dual to $\mathbf{G}$ and $\mathbf{s} \leftarrow \mathbb{F}^{n'}$, $\mathbf{e} \leftarrow \mathcal{E}$, and $\mathbf{u} \leftarrow \mathbb{F}^N$. Equivalence holds for $n' = N - n$ by solving for $\mathbf{G} \in \mathbb{F}^{n \times N}$ such that $\mathbf{G}\mathbf{H} = 0$ via row-reduction. It follows that

$$\mathbf{G}(\mathbf{H}\mathbf{s} + \mathbf{e}) = (\mathbf{G}\mathbf{H})\mathbf{s} + \mathbf{G}\mathbf{e} = \mathbf{G}\mathbf{e}$$

If dual-LPN holds over a distribution of generators $\mathcal{G}$ of a dual code, primal-LPN holds over the corresponding distribution of parity-check matrices $\mathcal{H}$ for the primal code. See, e.g., [13, 21] for further details.

Selecting a noise distribution $\mathcal{E}$. For LPN-based constructions, exact noise is typically used [20], where $\mathcal{E} = \mathcal{E}_t^N$ is the uniform distribution over the set of t -sparse vectors in $\mathbb{F}^N$. However, alternative distributions such as Bernoulli noise or regular noise may be considered.

Selecting code generators $\mathcal{G}$. A natural question arises of which $\mathcal{G}$ are LPN-friendly. The standard LPN assumption instantiates $\mathcal{G}$ as the uniform distribution over matrices in $\mathbb{F}^{n \times N}$. More recently, two separate lines of work in code-based encryption [2, 7] and pseudorandom correlation generators [19–21, 71] have sought to instantiate LPN from families of codes which enable linear or quasilinear encoding time. Such code families include quasi-cyclic, Toeplitz, LDPC [4], expand-accumulate (EA) [20], expand-convolute (EC) [71], and repeat-accumulate-accumulate (RAA) [24],

and more [13, 22]. In order to test the concrete security of various possibly LPN-friendly codes, [20] introduced the following unified framework.

Security against linear tests. Suppose $\mathcal{G}$ is a distribution over code generators $\mathbf{G}$ such that the code does not admit an efficient decoding algorithm. For instance, Reed-Solomon codes are highly algebraic and enable fast decoding via the Berlekamp-Massey algorithm [63], which trivially breaks the LPN assumption for these codes. Non-algebraic codes typically do not admit such decoding algorithms.

The key observation of [20] is that all known attacks on LPN over such codes reduce to finding a vector $\mathbf{v} \in \mathbb{F}^n$ (or equivalently a linear functional) such that LPN samples have some non-negligible bias with respect to $\mathbf{v}$. This procedure is called a *linear test* and is done by analyzing $\mathbf{v}^\top \mathbf{b}$ for $\mathbf{b} = \mathbf{G}\mathbf{e} \in \mathbb{F}^N$ and $\mathbf{e} \leftarrow \mathcal{E}$. Linear tests encapsulate information set decoding (ISD) attacks [68], the BKW algorithm [17], Gaussian elimination, and all other existing practical distinguishing attacks [20].

Then, [20] shows that the codes which are resistant to linear tests are precisely those which have high minimum distance, as this ensures that the inner product $\mathbf{v}^\top \mathbf{b} = (\mathbf{v}^\top \mathbf{G})\mathbf{e}$ has $\mathbf{v}^\top \mathbf{G}$ of large Hamming weight and sufficiently high probability of colliding with sparse $\mathbf{e}$. We formally state these ideas below.

Definition A.2 (Bias of a distribution). *For a nonzero vector $\mathbf{v} \in \mathbb{F}^n$ and a distribution $\mathcal{D}$ over $\mathbb{F}^N$, the bias of $\mathcal{D}$ with respect to $\mathbf{v}$ is defined as*

$$\text{bias}_{\mathbf{v}}(\mathcal{D}) := \max_{c \in \mathbb{F}} \left| \Pr_{\mathbf{x} \leftarrow \mathcal{D}} [\mathbf{v}^\top \mathbf{x} = c] - \frac{1}{|\mathbb{F}|} \right|.$$

The bias of $\mathcal{D}$ is defined as

$$\text{bias}(\mathcal{D}) = \max_{\mathbf{v} \neq 0} \text{bias}_{\mathbf{v}}(\mathcal{D}).$$

For a noise distribution $\mathcal{E} \subset \mathbb{F}^N$ and a code generator $\mathbf{G} \in \mathbb{F}^{n \times N}$, we let $\mathcal{E}_{\mathbf{G}} \subset \mathbb{F}^N$ denote the distribution of $\mathbf{G}\mathbf{e}$ where $\mathbf{e} \leftarrow \mathcal{E}$, as in [1]. Then, *linear testing* corresponds to the search problem where a PPT adversary must output some vector $\mathbf{v} \in \mathbb{F}^n$ such that $\text{bias}_{\mathbf{v}}(\mathcal{E}_{\mathbf{G}}) > \text{negl}(\lambda)$. As shown in [20, Lemma 2.6], if the minimum distance of the code generated by $\mathbf{G}$ is $\geq d$, then

$$\text{bias}(\mathcal{E}_{\mathbf{G}}) = \max_{\substack{\mathbf{u} \in \mathbb{F}^N \\ \mathbf{u} = \mathbf{v}^\top \mathbf{G} \text{ for some } \mathbf{v} \neq 0}} \text{bias}_{\mathbf{u}}(\mathcal{E}) \leq \max_{\substack{\mathbf{u} \in \mathbb{F}^N \\ \text{wt}(\mathbf{u}) \geq d}} \text{bias}_{\mathbf{u}}(\mathcal{E}).$$

For $\mathcal{E} = \mathcal{E}_t^N$ the uniform distribution over t -sparse vectors in $\mathbb{F}^N$, the bias for any $\mathbf{u} \in \mathbb{F}^N$ with Hamming weight at least d can be bounded by

$$\text{bias}_{\mathbf{u}}(\mathcal{E}_t^N) \leq \left(1 - \frac{t}{N}\right)^d < e^{-td/N} = e^{-\delta t},$$

where $\delta = d/N$ is the relative distance of the given code [20]. Since the search problem requires finding a suitable vector

$\mathbf{u} \in \mathbb{F}^N$, we can attain $\lambda - \log_2 N$ bits of security as in [1, 20] by setting $e^{-\delta t} \leq 2^{-\lambda + \log_2 N}$, or equivalently,

$$t \geq \frac{(\ln 2)(\lambda - \log_2 N)}{\delta}$$

where δ depends on the field $\mathbb{F}$ and dimensions n and N . For an asymptotically good code with $\delta = \Theta(1)$, this justifies the parameter selection of $t = O_\lambda(1)$.

The linear-test framework does not account for when the noise distribution is highly structured and an adversary is given sufficiently many samples, which enables algebraic attacks such as Arora-Ge [9]. We ensure that noise is generated via the *exact distribution* $\mathcal{E}_t^N$ to mitigate this risk².

B Repeat-Accumulate-Accumulate Codes

We instantiate the dual-LPN assumption with $\mathcal{G}$ being a generator for Repeat-Accumulate-Accumulate (RAA) linear codes. We provide a summary of these codes and previous work supporting the conjecture that RAA codes are LPN-friendly.

For any field $\mathbb{F}$, dimension n , block length N divisible by n , and constant $r = N/n$, an RAA code with generator $\mathbf{G}_{\text{RAA}} \in \mathbb{F}^{n \times N}$ is constructed via the composition

$$\mathbf{G}_{\text{RAA}} = \mathbf{R} \cdot \mathbf{\Pi}_1 \cdot \mathbf{V}_1 \cdot \mathbf{A} \cdot \mathbf{\Pi}_2 \cdot \mathbf{V}_2 \cdot \mathbf{A}.$$

Each of the component matrices is sampled and defined as follows:

- Repetition matrix $\mathbf{R} \in \mathbb{F}^{n \times N}$: repeats each entry of the input vector r times, given by

$$R_{i,j} = \begin{cases} 1 & \text{if } (i-1)r < j \leq ir \\ 0 & \text{otherwise} \end{cases}.$$

- Permutation matrix $\mathbf{\Pi} \in \mathbb{F}^{N \times N}$: samples and performs a uniformly random permutation $\pi : [N] \rightarrow [N]$ to the input vector, given by

$$\Pi_{i,j} = \begin{cases} 1 & \text{if } \pi(j) = i \\ 0 & \text{otherwise} \end{cases}.$$

- Randomization matrix $\mathbf{V} \in \mathbb{F}^{N \times N}$: samples a uniformly random nonzero vector $\mathbf{v} \leftarrow (\mathbb{F}^*)^N$ along the diagonal of a matrix

$$V_{i,j} = \begin{cases} v_i & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}.$$

- Accumulation matrix $\mathbf{A} \in \mathbb{F}^{N \times N}$: computes the prefix-sum of the input vector by applying the upper triangular matrix

$$A_{i,j} = \begin{cases} 1 & \text{if } i \leq j \\ 0 & \text{otherwise} \end{cases}.$$

² [20] also remarks that when instantiating the linear-test framework over a ring, one must be wary of projection attacks onto ideals. These do not concern our construction since fields do not contain non-trivial ideals.

Linear-time encoding. We observe that the encoding map $\mathbb{F}^n \rightarrow \mathbb{F}^N$ by $\mathbf{w} \mapsto \mathbf{G}_{\text{RAA}}^\top \mathbf{w}$ can be computed in $O(N)$ field operations. By the transposition principle [18], the syndrome map $\mathbb{F}^N \rightarrow \mathbb{F}^n$ by $\mathbf{v} \mapsto \mathbf{G}_{\text{RAA}} \mathbf{v}$ can similarly be computed in $O(N)$ field operations. We show that for each component matrix of $\mathbf{G}_{\text{RAA}}$, matrix–vector multiplication can be computed in time $O(N)$.

- Repetition matrix $\mathbf{w} \mapsto \mathbf{R}^\top \mathbf{w}$: requires N copies of field elements—no field operations.
- Permutation matrix $\mathbf{w} \mapsto \mathbf{\Pi}^\top \mathbf{w}$: requires N copies of field elements—no field operations.
- Randomization matrix $\mathbf{w} \mapsto \mathbf{V}^\top \mathbf{w}$: requires N field multiplications.
- Accumulation matrix $\mathbf{w} \mapsto \mathbf{A}^\top \mathbf{w}$: requires N field additions to compute the prefix-sum.

Composing the above operations yields a circuit which computes encodings $\mathbf{w} \mapsto \mathbf{G}_{\text{RAA}}^\top \mathbf{w}$ in at most $4N$ field operations. The concrete count for syndromes $\mathbf{v} \mapsto \mathbf{G}_{\text{RAA}} \mathbf{v}$ sits similarly at $5N$ field operations.

LPN friendliness and asymptotic goodness. RAA codes are non-algebraic due to the accumulation and permutation matrices. As such, it is conjectured that no efficient decoding algorithm exists, and that RAA codes are LPN-friendly [1, 13].

To motivate this conjecture, many previous papers have conducted analyses on the minimum distance of RAA codes, from which security against linear tests directly follows. Early analyses focused on asymptotic bounds for the minimum distance of RAA codes and limited their instantiation to binary fields [11, 57, 73]. Later works [50, 54, 55, 89] showed similarly strong minimum-distance properties and asymptotic goodness for RAA codes over arbitrary prime fields.

Since asymptotic analysis is insufficient for instantiating cryptographic protocols with provable guarantees, further work has provided concrete bounds on the minimum distance of RAA codes over various parameters. Binary fields are analyzed in the Blaze SNARK [24] and proven to achieve constant relative distance, while [1] analyzes prime fields for $n \in [2^4, 2^{10}]$ using combinatorial techniques and extrapolates to larger n . More recently, [3] gives a rigorous prime-field analysis proving a constant relative distance which is larger than in the binary case.

Failure probabilities. Crucially, when considering code generators $\mathbf{G}_{\text{RAA}}$ sampled from a distribution, guarantees of good minimum distance are probabilistic rather than exact. That is, for a specified dimension n , block length N , rate $r = N/n$, field $\mathbb{F}$, and relative distance δ , the code has a *failure probability* that a randomly sampled generator matrix $\mathbf{G}_{\text{RAA}}$ from this distribution will not have relative distance at least δ . We denote this failure probability by $p_{\text{fail}} = p_{\text{fail}}(\mathcal{G}, N, \delta)$. The focus of prior work is to demonstrate that the failure probability can be made arbitrarily small under practical parameters.

In particular, [3] and [54] jointly show that for any fixed $\mathbb{F}_q$ with $q > 2^{31}$, and for $\delta = 0.5$, the failure probability is $\tilde{O}(N^{2-r})^3$. Thus, setting $r = N/n \geq 3$ ensures that the failure probability decreases polynomially with N . Furthermore, a framework introduced in Blaze [24] enables publicly testing the distance of an RAA code in polynomial time $O(N^w)$ for any integer $w \in \mathbb{N}$ in order to reduce the failure probability to $\tilde{O}(N^{(w+1)(2-r)})$ [3].

C Private Matrix Extension

Π_{pvMVMD} can be extended to support a client who holds a private matrix $\mathbf{M} \in \mathbb{F}^{m \times n}$ in addition to a private vector $\mathbf{x} \in \mathbb{F}^n$, and wishes to delegate $\mathbf{M}\mathbf{x}$ without revealing either operand to the compute server. In private LLM inference, this corresponds to outsourcing linear-layer computation for a proprietary model owned by the client.

We next formalize its syntax and security.

Definition C.1 (Private-Matrix pvMVMD). A private-matrix pvMVMD protocol Π is a tuple of PPT algorithms $(\text{PREPROCESS}, \text{ENCRYPT}, \text{VERIFY}, \text{DECRYPT})$ defined by:

- $\text{PREPROCESS}(1^\lambda, 1^\kappa, \mathbf{M}) \rightarrow (\text{pp}, \hat{\mathbf{M}}, \text{st}_{\mathbf{M}})$. On input security parameter λ , statistical parameter κ , and matrix $\mathbf{M} \in \mathbb{F}^{m \times n}$, outputs public parameters pp , masked matrix $\hat{\mathbf{M}}$, and private matrix state $\text{st}_{\mathbf{M}}$.
- $\text{ENCRYPT}(\text{pp}, \mathbf{x}) \rightarrow (\hat{\mathbf{x}}, \text{st}_{\mathbf{x}})$. On input public parameters pp and vector $\mathbf{x} \in \mathbb{F}^n$, outputs ciphertext $\hat{\mathbf{x}}$ and private vector state $\text{st}_{\mathbf{x}}$.
- $\text{VERIFY}(\text{pp}, \hat{\mathbf{x}}, \hat{\mathbf{y}}) \rightarrow \{0, 1\}$. On input public parameters pp , ciphertext $\hat{\mathbf{x}} \in \mathbb{F}^n$, and server output $\hat{\mathbf{y}} \in \mathbb{F}^m$, outputs 1 (accept) or 0 (reject).
- $\text{DECRYPT}(\text{pp}, \text{st}_{\mathbf{M}}, \text{st}_{\mathbf{x}}, \hat{\mathbf{y}}) \rightarrow \mathbf{y}$. On input public parameters pp , private states $\text{st}_{\mathbf{M}}, \text{st}_{\mathbf{x}}$, and ciphertext $\hat{\mathbf{y}} \in \mathbb{F}^m$, outputs plaintext vector $\mathbf{y} \in \mathbb{F}^m$.

The protocol satisfies:

- **Completeness:** For any $\mathbf{M} \in \mathbb{F}^{m \times n}$ and $\mathbf{x} \in \mathbb{F}^n$,

$$\Pr \left[\mathbf{y} = \mathbf{M}\mathbf{x} \wedge b = 1 \mid \begin{array}{l} (\text{pp}, \hat{\mathbf{M}}, \text{st}_{\mathbf{M}}) \leftarrow \text{PREPROCESS}(1^\lambda, 1^\kappa, \mathbf{M}) \\ (\hat{\mathbf{x}}, \text{st}_{\mathbf{x}}) \leftarrow \text{ENCRYPT}(\text{pp}, \mathbf{x}) \\ \hat{\mathbf{y}} \leftarrow \hat{\mathbf{M}}\hat{\mathbf{x}} \\ b \leftarrow \text{VERIFY}(\text{pp}, \hat{\mathbf{x}}, \hat{\mathbf{y}}) \\ \mathbf{y} \leftarrow \text{DECRYPT}(\text{pp}, \text{st}_{\mathbf{M}}, \text{st}_{\mathbf{x}}, \hat{\mathbf{y}}) \end{array} \right] = 1.$$

- **Soundness:** There exists a negligible function $\text{negl}(\cdot)$ such that for any $\mathbf{M} \in \mathbb{F}^{m \times n}$ and $\mathbf{x} \in \mathbb{F}^n$,

$$\Pr \left[\mathbf{y} \neq \mathbf{M}\mathbf{x} \wedge b = 1 \mid \begin{array}{l} (\text{pp}, \hat{\mathbf{M}}, \text{st}_{\mathbf{M}}) \leftarrow \text{PREPROCESS}(1^\lambda, 1^\kappa, \mathbf{M}) \\ (\hat{\mathbf{x}}, \text{st}_{\mathbf{x}}) \leftarrow \text{ENCRYPT}(\text{pp}, \mathbf{x}) \\ \hat{\mathbf{y}} \leftarrow \mathcal{A}(\text{pp}, \hat{\mathbf{M}}, \hat{\mathbf{x}}) \\ b \leftarrow \text{VERIFY}(\text{pp}, \hat{\mathbf{x}}, \hat{\mathbf{y}}) \\ \mathbf{y} \leftarrow \text{DECRYPT}(\text{pp}, \text{st}_{\mathbf{M}}, \text{st}_{\mathbf{x}}, \hat{\mathbf{y}}) \end{array} \right] < \text{negl}(\kappa).$$

³ [54] has later refined this to $\tilde{O}(N^{1-r})$ under certain conditions.

- **Client privacy:** There exists a PPT simulator Sim such that for any $\mathbf{M} \in \mathbb{F}^{m \times n}$ and $\mathbf{x} \in \mathbb{F}^n$,

$$\text{Sim}(1^\lambda, 1^\kappa, 1^m, 1^n) \approx_c \text{View}_{\text{Server}}^\Pi(\mathbf{M}, \mathbf{x}),$$

where $\text{View}_{\text{Server}}^\Pi(\mathbf{M}, \mathbf{x}) := (\text{pp}, \hat{\mathbf{M}}, \hat{\mathbf{x}})$ is generated by $(\text{pp}, \hat{\mathbf{M}}, \text{st}_{\mathbf{M}}) \leftarrow \text{PREPROCESS}(1^\lambda, 1^\kappa, \mathbf{M})$ and $(\hat{\mathbf{x}}, \text{st}_{\mathbf{x}}) \leftarrow \text{ENCRYPT}(\text{pp}, \mathbf{x})$.

Our construction leverages the notion of a *trapdoored matrix* [23, 81], which is a pseudorandom matrix sampled alongside a trapdoor that enables fast MVMS by that matrix.

Definition C.2 (Trapdoored matrix [23, 81]). A trapdoored matrix scheme TDM is a pair of PPT algorithms $(\text{TDM.GEN}, \text{TDM.EVAL})$ with the following syntax:

- $\text{TDM.GEN}(1^\lambda, 1^m, 1^n) \rightarrow (\mathbf{M}', \text{td}_{\mathbf{M}'})$ outputs a matrix $\mathbf{M}' \in \mathbb{F}^{m \times n}$ and a trapdoor $\text{td}_{\mathbf{M}'}$.
- $\text{TDM.EVAL}(\text{td}_{\mathbf{M}'}, \mathbf{x})$ takes trapdoor $\text{td}_{\mathbf{M}'}$ and vector $\mathbf{x} \in \mathbb{F}^n$, and outputs a vector in $\mathbb{F}^m$.

TDM satisfies the following properties:

- **Correctness:** For every $(\mathbf{M}', \text{td}_{\mathbf{M}'})$ output by $\text{TDM.GEN}(1^\lambda, 1^m, 1^n)$ and every $\mathbf{x} \in \mathbb{F}^n$,

$$\text{TDM.EVAL}(\text{td}_{\mathbf{M}'}, \mathbf{x}) = \mathbf{M}'\mathbf{x}.$$

- **Pseudorandomness:** The distribution of $\mathbf{M}'$ is computationally indistinguishable from the uniform distribution over $\mathbb{F}^{m \times n}$ to any PPT adversary that does not know $\text{td}_{\mathbf{M}'}$.

Braverman and Newman [23] and Vaikuntanathan and Zamiir [81] both construct TDMs using recursion, where TDM.EVAL runs in quasilinear time $T_{\text{TDM}}(m, n) = \tilde{O}(m + n)$. Using dual-LPN-friendly linear-time codes, EMVP [13] improves the runtime to linear $T_{\text{TDM}}(m, n) = O(m + n)$.

In our construction, the client masks $\mathbf{M}$ by sampling $(\mathbf{M}', \text{td}_{\mathbf{M}'}) \leftarrow \text{TDM.GEN}$ and uploading $\hat{\mathbf{M}} := \mathbf{M} + \mathbf{M}'$ to the server, storing $\text{td}_{\mathbf{M}'}$ locally. The preprocessing terms $\mathbf{P} := \hat{\mathbf{M}}\mathbf{G}_x$ and $\mathbf{Q} := \mathbf{G}_y^\top \hat{\mathbf{M}}$ are generated as part of pp . For each query, the client masks $\mathbf{x}$ as $\hat{\mathbf{x}} := \mathbf{x} + \mathbf{G}_x \mathbf{e}_x$, and the server computes $\hat{\mathbf{y}} := \hat{\mathbf{M}}\hat{\mathbf{x}} = (\mathbf{M} + \mathbf{M}')(\mathbf{x} + \mathbf{x}')$. The client checks the claim using VERIFY and decrypts $\mathbf{y} := \hat{\mathbf{y}} - \mathbf{P}\mathbf{e}_x - \text{TDM.EVAL}(\text{td}_{\mathbf{M}'}, \mathbf{x})$. Protocol 4 presents the full protocol.

Theorem C.3. Suppose that TDM satisfies correctness and pseudorandomness, $\mathcal{G}_x$ is a distribution over codes with relative distance at least δ_x satisfying the $(\mathcal{E}_{\mathcal{G}_x}^{N_x}, \mathcal{G}_x, \mathbb{F})$ -dual-LPN assumption for polynomially many independent samples, and the code generated by $\mathbf{G}_y$ has relative distance δ_y . Then $\Pi_{\text{pvMVMD}}^{\text{privM}}$ satisfies completeness, soundness, and client privacy as defined in Definition C.1 when instantiated with $|\mathbb{F}| \geq 4$, $\ell = \left\lceil \frac{\kappa}{\log_2(|\mathbb{F}|-1)-1} \right\rceil$, and $t_y = \left\lceil \frac{\max(\log_2(|\mathbb{F}|-1), \kappa+1)}{-\log_2(1-\delta_y)} \right\rceil$.

$\Pi_{\text{pvMVMD}}^{\text{privM}}$

Input parameters:

- Trapdoored matrix scheme TDM = (TDM.GEN, TDM.EVAL) for matrices in $\mathbb{F}^{m \times n}$.
- Code distributions $\mathcal{G}_x$ and $\mathcal{G}_y$, sparsity parameters t_x, t_y , repetition parameter ℓ , and block lengths N_x, N_y as in Protocol 3.
- Field oracle $\mathbb{F}$.

PREPROCESS($1^\lambda, 1^\kappa, \mathbf{M}$) $\rightarrow$ (pp, $\hat{\mathbf{M}}$, st_M):

- Sample $(\mathbf{M}', \text{td}_{\mathbf{M}'}) \leftarrow \text{TDM.GEN}(1^\lambda, 1^m, 1^n)$
- Compute $\hat{\mathbf{M}} := \mathbf{M} + \mathbf{M}' \in \mathbb{F}^{m \times n}$.
- Sample $\mathbf{G}_x \leftarrow \mathcal{G}_x$ and $\mathbf{G}_y \leftarrow \mathcal{G}_y$.
- Compute $\mathbf{P} := \hat{\mathbf{M}}\mathbf{G}_x \in \mathbb{F}^{m \times N_x}$.
- Compute $\mathbf{Q} := \mathbf{G}_y^\top \hat{\mathbf{M}} \in \mathbb{F}^{N_y \times n}$.
- Output pp := ($\mathbf{G}_x, \mathbf{G}_y, \mathbf{P}, \mathbf{Q}, t_x, t_y, \ell$), $\hat{\mathbf{M}}$, and st_M := td_{M'}.

ENCRYPT(pp, $\mathbf{x}$) $\rightarrow$ ($\hat{\mathbf{x}}$, st_x):

- Sample $\mathbf{e}_x \leftarrow \mathcal{E}_{t_x}^{N_x}$ and compute $\mathbf{x}' := \mathbf{G}_x \mathbf{e}_x \in \mathbb{F}^n$.
- Output $\hat{\mathbf{x}} := \mathbf{x} + \mathbf{x}'$ and st_x := ($\mathbf{e}_x, \mathbf{x}$).

VERIFY(pp, $\hat{\mathbf{x}}, \hat{\mathbf{y}}$) $\rightarrow$ {0, 1}:

- Compute $\mathbf{G}_y^\top \hat{\mathbf{y}} \in \mathbb{F}^{N_y}$.
- Repeat ℓ times:
 - Sample $\mathbf{e}_y \leftarrow \mathcal{E}_{t_y}^{N_y}$.
 - Compute $v_L := (\mathbf{e}_y^\top \mathbf{Q}) \hat{\mathbf{x}}$ and $v_R := \mathbf{e}_y^\top (\mathbf{G}_y^\top \hat{\mathbf{y}})$.
 - If $v_L \neq v_R$, output 0 (reject) and halt.
- If all ℓ checks pass, output 1 (accept).

DECRYPT(pp, st_M, st_x, $\hat{\mathbf{y}}$) $\rightarrow$ $\mathbf{y}$:

- Parse st_M = td_{M'} and st_x = ($\mathbf{e}_x, \mathbf{x}$).
- Compute $\mathbf{y}_1 := \mathbf{P}\mathbf{e}_x$ and $\mathbf{y}_2 := \text{TDM.EVAL}(\text{td}_{\mathbf{M}'}, \mathbf{x})$.
- Output $\mathbf{y} := \hat{\mathbf{y}} - \mathbf{y}_1 - \mathbf{y}_2 \in \mathbb{F}^m$.

Protocol 4: Protocol $\Pi_{\text{pvMVMD}}^{\text{privM}}$ for private-matrix and private-vector delegation.

Proof. For any vector $\hat{\mathbf{y}}$, the decrypted output is $\mathbf{y} := \hat{\mathbf{y}} - \mathbf{P}\mathbf{e}_x - \text{TDM.EVAL}(\text{td}_{\mathbf{M}'}, \mathbf{x})$. Using $\mathbf{P} = \hat{\mathbf{M}}\mathbf{G}_x$, $\text{TDM.EVAL}(\text{td}_{\mathbf{M}'}, \mathbf{x}) = \mathbf{M}'\mathbf{x}$, and $\hat{\mathbf{M}} = \mathbf{M} + \mathbf{M}'$, we derive

$$\begin{aligned} \mathbf{y} - \mathbf{M}\mathbf{x} &= \hat{\mathbf{y}} - \hat{\mathbf{M}}\mathbf{G}_x\mathbf{e}_x - \mathbf{M}'\mathbf{x} - \mathbf{M}\mathbf{x} \\ &= \hat{\mathbf{y}} - \hat{\mathbf{M}}\mathbf{G}_x\mathbf{e}_x - (\mathbf{M} + \mathbf{M}')\mathbf{x} \\ &= \hat{\mathbf{y}} - \hat{\mathbf{M}}(\mathbf{x} + \mathbf{G}_x\mathbf{e}_x) \\ &= \hat{\mathbf{y}} - \hat{\mathbf{M}}\hat{\mathbf{x}}. \end{aligned}$$

Completeness. If the server honestly evaluates $\hat{\mathbf{y}} = \hat{\mathbf{M}}\hat{\mathbf{x}}$, then the identity above yields $\mathbf{y} = \mathbf{M}\mathbf{x}$. Further, VERIFY accepts with probability 1 by Theorem 5.2.

Soundness. If the server returns an output with $\mathbf{y} \neq \mathbf{M}\mathbf{x}$, the identity above implies $\hat{\mathbf{y}} \neq \hat{\mathbf{M}}\hat{\mathbf{x}}$. By Theorem 5.2, VERIFY rejects with probability at least $1 - 2^{-\kappa}$ for these parameters.

Client privacy. We prove client privacy via two hybrid steps. In the first hybrid, we replace the trapdoored matrix $\mathbf{M}'$

with a uniformly random matrix $\mathbf{U}_M \leftarrow \mathbb{F}^{m \times n}$, which is computationally indistinguishable by the pseudorandomness of TDM. After this replacement, $\hat{\mathbf{M}} = \mathbf{M} + \mathbf{U}_M$ is uniformly distributed over $\mathbb{F}^{m \times n}$ and independent of $\mathbf{M}$. In the second hybrid, we replace $\mathbf{G}_x\mathbf{e}_x$ with an independently sampled uniform vector $\mathbf{u} \leftarrow \mathbb{F}^n$, which is computationally indistinguishable under the $(\mathcal{E}_{t_x}^{N_x}, \mathcal{G}_x, \mathbb{F})$ -dual-LPN assumption. After this replacement, $\hat{\mathbf{x}} = \mathbf{x} + \mathbf{u}$ is uniformly distributed over $\mathbb{F}^n$ and independent of $\mathbf{x}$. The resulting distribution is simulated by a PPT simulator $\text{Sim}(1^\lambda, 1^\kappa, 1^m, 1^n)$ that samples uniform $\tilde{\mathbf{M}} \leftarrow \mathbb{F}^{m \times n}$ and $\tilde{\mathbf{x}} \leftarrow \mathbb{F}^n$, computes $\tilde{\text{pp}}$ from $\tilde{\mathbf{M}}$, and outputs $(\tilde{\text{pp}}, \tilde{\mathbf{M}}, \tilde{\mathbf{x}}) \approx_c \text{View}_{\text{Server}}^\Pi(\mathbf{M}, \mathbf{x})$. Computational indistinguishability follows from these two hybrids. $\square$

Efficiency. The online client computes TDM.EVAL in some time $T_{\text{TDM}}(m, n)$ and all other ENCRYPT, VERIFY, and DECRYPT steps in time $O_{\lambda, \kappa}(m + n)$ by Section 6.3. Choosing a linear evaluation time TDM (e.g., in [13]) provides asymptotically optimal online time $O_{\lambda, \kappa}(m + n)$. The server performs exactly one field MVM, and preprocessing requires $O(mn)$ field operations to compute $\mathbf{P}$ and $\mathbf{Q}$.

Drawback on preprocessing. In the public-matrix setting pvMVMD, preprocessing is *transparent* and can be computed once by the server and published globally for all clients. In contrast, for private-matrix delegation, each client must sample their own independent trapdoor td_{M'} and mask the matrix $\hat{\mathbf{M}}$ privately, requiring $O(mn)$ *private* preprocessing time. This construction is therefore most attractive when the preprocessing cost can be amortized across many queries.

D Soundness Composition

We show that any private delegation protocol can be made sound by composing with a sound verifiable delegation protocol. This is the same technique we use in constructing Π_{pvMVMD} from Π_{vMVMD} in Protocol 3, and this demonstrates that a lightweight verification layer can be applied to a private delegation protocol.

Let Π_{vMVMD} be a complete and sound vMVMD protocol, and let Π_{pvMVMD} be a complete and private but possibly unsound private delegation protocol. We informally define the protocol Π'_{pvMVMD} as follows:

- **PREPROCESS:** run $\Pi_{\text{vMVMD}}.\text{PREPROCESS}$ and $\Pi_{\text{pvMVMD}}.\text{PREPROCESS}$.
- **ENCRYPT:** run $\Pi_{\text{pvMVMD}}.\text{ENCRYPT}$.
- **VERIFY:** run $\Pi_{\text{vMVMD}}.\text{VERIFY}$.
- **DECRYPT:** run $\Pi_{\text{pvMVMD}}.\text{DECRYPT}$.

Completeness and client privacy of Π'_{pvMVMD} follow from the corresponding properties of Π_{vMVMD} and Π_{pvMVMD} .

For soundness, suppose the adversary returns $\hat{\mathbf{y}}$ such that $\mathbf{y} \neq \mathbf{M}\mathbf{x}$, where $\mathbf{y} = \Pi'_{\text{pvMVMD}}.\text{DECRYPT}(\text{pp}, \text{st}, \hat{\mathbf{y}})$. By the

completeness of Π_{pvMVMD} , if $\hat{\mathbf{y}} = \mathbf{M}\hat{\mathbf{x}}$ then $\mathbf{y} = \mathbf{M}\mathbf{x}$; taking the contrapositive, if $\mathbf{y} \neq \mathbf{M}\mathbf{x}$, it must hold that $\hat{\mathbf{y}} \neq \mathbf{M}\hat{\mathbf{x}}$. The VERIFY algorithm of Π_{MVMD} is applied to the claim

$$\hat{\mathbf{y}} \stackrel{?}{=} \mathbf{M}\hat{\mathbf{x}}.$$

By soundness of Π_{MVMD} , VERIFY detects this inconsistency except with negligible probability.

E Attack on Slalom at the Carnival

Slalom at the Carnival (S@C) by Bruhns et al. [25] proposes an addition to Slalom [80] which reduces the client online running time required to encrypt and decrypt vectors in a pvMVMD protocol. They claim to achieve client privacy; however, we show a flaw in their approach which breaks client privacy and enables a semi-honest server to uncover projections of the client's input vectors.

S@C protocol. Fix a matrix $\mathbf{M} \in \mathbb{F}^{m \times n}$ and a parameter $\ell \ll n$ which is dependent on the security parameter. In the S@C preprocessing, the server samples ℓ vectors $\mathbf{r}_1, \dots, \mathbf{r}_\ell \leftarrow \mathbb{F}^n$ and computes the corresponding projections $\mathbf{u}_1, \dots, \mathbf{u}_\ell \in \mathbb{F}^m$, where $\mathbf{u}_i = \mathbf{M}\mathbf{r}_i$. These projections are sent to the client. When the client wishes to delegate an MVM for a vector $\mathbf{x}$, they first sample $\mathbf{k} \leftarrow \{0, 1\}^\ell$, compute the masking term

$$\mathbf{x}' := \sum_{i=1}^{\ell} k_i \mathbf{r}_i \in \mathbb{F}^n,$$

and encrypt $\hat{\mathbf{x}} := \mathbf{x} + \mathbf{x}'$. This is sent to the server, which returns $\hat{\mathbf{y}} = \mathbf{M}\hat{\mathbf{x}}$. The client decrypts by subtracting the projections:

$$\mathbf{y} := \hat{\mathbf{y}} - \sum_{i=1}^{\ell} k_i \mathbf{u}_i.$$

The attack. The S@C protocol is efficient only when $\ell < n$. However, in this parameter range, the server can use Gaussian elimination to solve for a projection matrix $\mathbf{\Pi} \in \mathbb{F}^{(n-\ell) \times n}$ annihilating the vectors $\mathbf{r}_1, \dots, \mathbf{r}_\ell$ (i.e., $\mathbf{\Pi}\mathbf{r}_i = 0$ for each i).

For a client input $\hat{\mathbf{x}}$, the server applies $\mathbf{\Pi}$ and recovers

$$\mathbf{\Pi}\hat{\mathbf{x}} = \mathbf{\Pi} \left(\mathbf{x} + \sum_{i=1}^{\ell} k_i \mathbf{r}_i \right) = \mathbf{\Pi}\mathbf{x} \in \mathbb{F}^{n-\ell}.$$

This enables the server to distinguish $\hat{\mathbf{x}}$ from truly random and precludes the existence of a privacy simulator that works for all $\mathbf{x}$.