

Article

Comprehensive Benchmarking of Secure Computation Technologies for Machine Learning on General Purpose Hardware [†]

Marcus Taubert , Adam Skuta  and Thomas Loruenser * 

AIT Austrian Institute of Technology, 1210 Vienna, Austria; marcus.taubert@ait.ac.at (M.T.); adam.skuta@ait.ac.at (A.S.)

* Correspondence: thomas.loruenser@ait.ac.at

[†] This paper is an extended version of our paper published in Taubert, M.; Skuta, A.; Loruenser, T. A Pragmatic Comparison of Cryptographic Computation Technologies for Machine Learning. In Proceedings of the 12th International Conference on Information Systems Security and Privacy—Volume 2: ICISPP, Marbella, Spain, 4–6 March 2026. doi: 10.5220/0014567700004061.

Abstract

The increasing need for privacy in machine learning has driven interest in cryptographic methods that enable computation on sensitive data. Secure multi-party computation and fully-homomorphic encryption are two prominent approaches, but their practical trade-offs are often unclear for real-world use. This work presents a pragmatic benchmark and comparison of these technologies for machine learning inference. Using representative open source frameworks, we evaluate performance across basic operations, distance metrics, regression models, and common deep learning architectures. The evaluation accounts for realistic deployment conditions, including communication overhead and partial plaintext execution. The results show that secure multi-party computation scales well to large and complex models but is strongly affected by network latency, while fully-homomorphic encryption is easier to deploy and performs well for small models and regression tasks. We conclude that the two approaches are complementary rather than competing and provide concrete guidance to help practitioners select the most suitable technology for privacy preserving machine learning in practice.

Keywords: secure computation; secure multi-party computation; fully-homomorphic encryption; machine learning



Academic Editor: Jose María Alvarez Rodríguez

Received: 9 February 2026

Revised: 11 June 2026

Accepted: 17 June 2026

Published: 30 June 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

In an increasingly digitized world where vast amounts of data flow through the internet, privacy is a growing concern for users, governments, and organizations alike. This has strengthened the need for fast, reliable, and secure cryptographic technologies, leading to extensive research over the last decade. Concepts such as fully-homomorphic encryption (FHE), which allows calculations on encrypted data, and secure multi-party computation (MPC), where parties collaborate without revealing private inputs, have evolved from theoretical ideas into frameworks and schemes now fast enough for real-world deployment. While they rely on different security assumptions, such as non-collusion for MPC versus LWE related hardness for FHE, both can be applied to similar use cases to enhance security on third-party cloud platforms. MPC can be utilized by single input and output parties, while FHE can be adapted for multi-party scenarios using, e.g., threshold FHE. Both of

these technologies are comprised of a multitude of techniques and frameworks. In our work we do not aim to provide a thorough comparison of all these methods, but pick out the best candidate in FHE and MPC for machine learning purposes, to benchmark and compare them. Similarly, after examining the secure computation technologies, we are exploring usable frameworks and deciding on one promising library each to run our tests on. Having done these a pragmatic comparison is performed by benchmarking MPC and FHE on the models with similar workloads. We do not aim to provide a unified cost model for ML inference depending on various variables. Such is almost impossible, as evaluation times are dependent on way too many factors to viably create such a formula. Therefore we believe that benchmarks are the correct approach to address this issue and also offer a clearer insight for practitioners as opposed to an abstract formula. This article is the extension of our conference paper [1].

1.1. Motivation

Our motivation for this work stems from the fact that secure multi-party computation (MPC) and fully-homomorphic encryption (FHE) can fulfill similar needs, despite utilizing different internal techniques. This overlap often makes it difficult for practitioners to understand and determine which technology is most appropriate for their needs. We intend to provide a helping hand by testing both technologies across various machine learning use cases. Through this, we aim to identify the advantages and disadvantages of each paradigm, providing engineers with a clear understanding of their capabilities. By clarifying which framework should be used in specific scenarios to achieve peak efficiency and privacy, we hope to foster further research and widespread adoption.

1.2. Related Work

In recent years, numerous secure multi-party computation (MPC) frameworks have emerged. MP-SPDZ [2] is a research-oriented platform created to test and benchmark various programs across an extensive range of protocols, accommodating all categories of adversaries. Additionally, it facilitates the integration and privacy-preserving training of deep learning models. MPyC (<https://github.com/lschoe/mpyc> (accessed on 16 June 2026)) serves as a versatile MPC framework that enables diverse computations programmed in Python. Recent advancements in privacy-preserving machine learning via MPC have largely centered on attaining viable performance for large language models (LLMs) [3–5]. Researchers utilize diverse strategies, including the simplification of activation functions to lower inference latency and the application of knowledge distillation to preserve model accuracy. A different methodology involves hybrid systems where the client evaluates neural network activation functions in plaintext or the model weights are permuted. This allows weights to remain public to accelerate inference while simultaneously safeguarding the model against theft.

In recent years, significant research has focused on benchmarking basic operations within FHE frameworks. The study in [6] evaluates fundamental tasks such as additions, multiplications, and rotations across various FHE schemes. Alternatively, “Cross-Platform Benchmarking of the FHE Libraries: Novel Insights into SEAL and OpenFHE” [7] provides a comparison of two prominent libraries through the benchmarking of simple calculations, concluding that OpenFHE demonstrates superior performance over the SEAL library. Further research has investigated the execution times required for bootstrapping across different schemes [8], identifying TFHE as the most efficient candidate by a substantial margin. Significant progress has also been made regarding machine learning applications using FHE. FHEMaLe [9] serves as a framework for evaluating inference speeds across cloud environments utilizing CKKS and edge devices employing TFHE. This framework implements

various models, including k-nearest-neighbour, support vector machine, logistic regression, and neural networks. Their findings indicate that CKKS consistently outperformed TFHE, with the k-nearest-neighbour model demonstrating the slowest performance. Another contribution, Orion [10], provides a fully automated framework that translates neural networks into FHE for single-threaded inference. Utilizing the CKKS scheme, the authors benchmark their system on various architectures, such as different ResNet configurations and dense models. They introduce and incorporate an automatic bootstrap placement algorithm alongside a single-shot multiplexed packing strategy designed for arbitrary convolutions. Finally, the authors provide evaluation results for encrypted image recognition tasks.

As illustrated there is extensive literature examining practicability of machine learning in either FHE or MPC. However, direct comparisons are rare and often remain on a theoretical basis. In the papers [11,12] both secure computation technologies are examined with regards to cloud-based business intelligence and financial services, but they remain theoretical. Kalyan Cheerla et al. compare FHE and garbled circuits, a MPC variation, concerning machine learning [13]. They also implement a two-layer neural network and evaluate its inference times, as well as deviated error and memory usage. They find that MPC is significantly faster, but also requires interaction, resulting in a harder setup, and leaks of model structure.

1.3. Contributions

This work substantially extends previous work from [1], where the approach was first presented. Specifically, we provide a more comprehensive introduction and comparison of the secure computation technologies frameworks. We extend our conference paper by adding further model and calculation tests. In particular distance measure benchmarks, logistic regression and LSTM inference were added. We additionally expand these tests to real-world context where we do network studies for MPC regarding communication time delays and packet loss. Furthermore, different well-known face recognition models are tested. For FHE a novel approach to model inference called hybrid models, where evaluation is split to client and server side, is discussed and benchmarked. Furthermore quantization effects on model inference, security and power consumption are examined. Lastly we give a brief overview of existing frameworks designed to combine both technologies to get the best of both worlds and summarize our findings. Through those various additions readers and practitioners of privacy-enhancing technologies should get a much more thorough insight into both technologies and therefore be able to better weigh advantages and disadvantages of MPC and FHE.

1.4. Paper Structure

The remainder of this paper is organized as follows. Section 2 introduces the general principles of secure multi-party computation (MPC) and fully-homomorphic encryption (FHE) and compares their protocol-level assumptions, security models, and deployment characteristics. Section 3 presents the selected software frameworks, Secretflow-SPU for MPC and Concrete-ML for FHE, and discusses their usability, supported models, documentation, maintenance, and implementation differences.

Subsequently, Section 4 contains the main benchmarking study. It first describes the test environment and security settings, and then evaluates both technologies on basic operations, distance metrics, regression models, and deep learning architectures. The section also considers more realistic deployment factors, including network delay and packet loss for MPC, as well as hybrid execution, quantization effects, security parameters, and power usage for FHE. Section 5 discusses the benchmark results and derives practical recommendations for choosing between MPC and FHE in different machine learning

scenarios. It also briefly reviews approaches that combine both technologies. Finally, Section 6 concludes the paper by summarizing the main findings and outlining future work, including hardware acceleration, realistic distributed deployments, and encrypted-computation-aware model design.

2. General Protocol Comparison

2.1. MPC Protocol

Secure multi-party computation (MPC) [14] consists of cryptographic protocols enabling a group of participants to jointly compute a function using their private inputs while maintaining total data confidentiality. The primary goal is to produce an output identical to one generated by a strictly trusted third party operating on centralized data. Formally, MPC involves n parties $P_1, P_2, \dots, P_n$, where each participant P_i holds a private value x_i , where $i \in [1, n]$. The objective is to evaluate the function (1) so that the result y is disclosed only to authorized parties without exposing any further information.

$$y \leftarrow f(x_1, x_2, \dots, x_n) \quad (1)$$

The role of MPC protocols is to ensure the privacy and integrity of the underlying computation. These protocols are often based on the security model they support. Based on the adversary types of the corrupted parties, the protocols can be categorized as following:

- **Semi-honest** adversaries (also known as honest-but-curious) are corrupted parties that follow the computation correctly, but can attempt to learn some of the information that should stay unknown to them. They can collude with other parties as well,
- **Malicious** adversaries deviate from the protocol and may actively attempt to compromise the privacy and correctness of the computation,
- **Covert** adversaries do not have to follow the protocol and can alternate between malicious and semi-honest behaviour, depending on their underlying strategy.

Another possible description of the security model is the number of corrupted parties. They are divided into two main categories:

- **Honest-majority** setting assumes that more than half of the party members are honest and not corrupted,
- **Dishonest-majority** setting assumes that half or more of the party members are corrupted.

Depending on the adversarial model, different security guarantees must be satisfied by the MPC protocol. For the **semi-honest** adversary model, the protocol must ensure both privacy and correctness. Privacy requires that each party learn no information about the other parties' computations beyond what is revealed by the final output. Correctness ensures that, as long as the honest parties execute the protocol properly, the final output is accurate, regardless of any passive observation by semi-honest adversaries. For the **malicious** adversary model, the protocol must guarantee privacy and correctness, and additionally robustness and verifiability. Robustness ensures that the protocol continues to operate as intended even if malicious parties attempt to alter the output or disrupt the computation. Verifiability requires that any malicious behaviour is detectable and that countermeasures can be taken against it. For the **covert** adversary model, the protocol must balance efficiency with robustness. A key requirement in this model is accountability, which ensures that covert misbehaviour can be detected with high probability. Another possible requirement is fairness, which guarantees that no party gains access to the output earlier than the others.

Secret-sharing is a cryptographic technique that serves as a fundamental component in MPC systems. It divides a secret value into n shares, which are distributed among n parties such that only a predefined subset of parties is required to reconstruct the original secret. Once distributed, these shares can be used to perform computations, supporting operations such as addition and multiplication, although some operations may require additional communication between parties. In the context of machine learning, two forms of secret sharing are particularly relevant: arithmetic secret sharing and binary secret sharing. Arithmetic secret shares [15] represents values as elements of a finite ring f.e. $\mathbb{Z}/Q\mathbb{Z}$ for some large integer Q . A value x is shared among n parties by splitting it into additive shares $[x]_p$ such that $x = \sum_{p \in P} [x]_p \bmod Q$. This type of sharing is suitable for computation of arithmetic operations such as multiplication and addition, which translates to usefulness in neural network operations like matrix multiplication and convolutions and other linear operations. Binary secret sharing, on the other hand based on the field $\mathbb{Z}/2\mathbb{Z}$ and operates on the individual bits of values. Each bit is split into binary shares so that their XOR (exclusive OR logical operation) reconstructs the original bit. Binary secret shares are inefficient in terms of arithmetic operations, but are suitable for operations like comparisons, which in the machine learning context are used for activation functions like ReLU.

2.2. FHE Protocol

Fully homomorphic encryption (FHE) can be a game-changer for cloud computing services. It enables calculating on encrypted data, thus ensuring continued privacy when outsourcing such to the cloud. It relies on the hardness assumption of the learning with errors (LWE) problem, which is said to be post-quantum secure for certain configurations [16]. By keeping the added error as small as possible and introducing other noise management features, LWEs' inherent ability for homomorphic operations like additions and multiplications can be utilized.

Learning with errors can be best described by an overdetermined system of equations. The parameters $n \in \mathbb{Z}$ and $N \in \mathbb{Z}$ define the dimension of these equations and their amounts. These are the most important parameters. Other relevant parameters include the noise distribution and the ciphertext modulus. They are chosen such that a desired security level (e.g., 128-bit security) is achieved. In general, larger values of the LWE dimension n and the ring dimension N increase the security level, but also result in higher computational cost and slower execution.

To determine parameter sets that achieve a target security level, the lattice estimator—a SageMath-based tool that provides security estimates for LWE instances with given parameters—can be used [17]. Furthermore, the homomorphic encryption standard provides recommended parameter sets for achieving specific security levels [18].

However, because of FHE's homomorphic nature, current practical FHE schemes are inherently not able to resist active attacks. They only offer security against passive attackers. Thus, instead of being IND-CCA secure [19], it only fulfills the weaker security notion of IND-CPA or IND-CPA-D [20] security. There are approaches to creating IND-CCA1 secure schemes by, for example, using succinct non-interactive arguments of knowledge (SNARKS) or other techniques [21]. They however often have a large overhead and restricted functionality. Similarly, integrity-enhancing homomorphic schemes relying on, e.g., trusted execution environments (TEEs) [22], hashes [23], message authentication codes (MACs) [24] or zero-knowledge proofs (ZKPs) [25], face comparable limitations and remain largely research prototypes rather than practical schemes. Thus servers should be semi-honest (also called honest-but-curious) at most. Additionally it should be combined with other cryptographic mechanisms, that preserve Integrity.

In order to make encryption schemes fully homomorphic, an efficient noise management needs to be applied; otherwise, the noise (added error to prevent dictionary attacks) gets too big, making correct decryption impossible. Via a so-called bootstrap, it is possible to reduce the error in a ciphertext. This bootstrap encrypts the cipher with a different key with minimal noise and then decrypts it with the original one, thus removing its noise. This functionality historically has been quite slow [8] and still is for some schemes. Thus two schools of FHE schemes emerged. The levelled one aims to reduce noise growth and configure applied parameters to fit the wanted calculation circuit into the noise budget. Bootstrapped schemes, on the other hand, aim to reduce bootstrapping-induced overhead by making their procedure especially simple.

BGV [26], BFV [27] and CKKS [28] are currently the most widely adopted levelled schemes. The first two operate over the integers and use exact arithmetic, while CKKS uses real numbers with fix-point representation and approximate calculations. All three schemes are able to compute additions and multiplications. The use of real numbers in CKKS also makes it intuitive to approximate non-linear functions. This would make it a strong contender for machine learning appliances. However, due to its noise restrictions, it presents significant challenges when implementing larger models.

TFHE [29] on the other hand is a bootstrapped scheme and is able to perform said computation in less than a millisecond. It operates over the torus and thus implements Boolean arithmetic. It can evaluate all unary and binary gates, as well as perform additions and multiplications. It furthermore implements programmable bootstrapping logic, with which it is possible to calculate a univariate function during the bootstrapping procedure. This is possible due to look-up tables (LUTs), which essentially map all possible inputs to all outputs and calculate the right entry. The combination of all those characteristics makes it possible to evaluate arbitrary functions. Given TFHE's wide functionality, we will use it for the rest of the paper and whenever we refer to FHE, we implicitly mean TFHE.

2.3. Protocol Discussion and Comparison

MPC provides several distinct benefits for privacy-preserving computation. It ensures privacy by allowing multiple participants to jointly evaluate a function while keeping all inputs and intermediate values confidential; only authorized parties receive the final output. Because of its multi-party architecture, it promotes decentralization, which removes the risk of a single point of failure and allows the system to be configured to withstand the failure or corruption of specific participants. Furthermore, MPC accommodates a diverse range of protocols and trust models, providing the flexibility to tailor the security parameters to the precise needs of a given application.

However, MPC's nature requires the deployment of multiple servers to create n computing parties. This results in having to purchase n sets of hardware, as well as their assembly and configuration, which increases deployment costs and efforts significantly. Furthermore, this decentralized character creates communication overhead with potential packet loss and transfer speed fluctuations, making inference slower and speed-up from faster hardware less significant.

On the contrary, FHE requires only client and server interaction, with the client being responsible for encryption and decryption and the server for function evaluation. This makes FHE quick and easy to set up, as well as allows changing existing servers to run on FHE, without having to build new systems or networks.

However, FHE's current induced overhead is non-significant, making it impractical for certain use cases. The major bottleneck lies in the need for noise reduction. As previously mentioned bootstrapping historically has been very slow [8]. The TFHE scheme has significantly reduced the overhead of bootstrapping to the order of milliseconds or less [29,30].

Nevertheless, its reliance on frequent bootstrapping operations—often after, or even for, individual computations—remains a major performance bottleneck, as practical circuits may require the evaluation of tens of thousands of such operations. Still, a steady reduction in bootstrapping latency has been observed over the years, suggesting that further research and development are likely to yield additional performance improvements.

Another limitation of FHE arises from its homomorphic nature. At present, no fully homomorphic encryption scheme is known to satisfy any notion of IND-CCA security [19]. Consequently, FHE can only be securely employed in settings with semi-honest adversaries, who follow the protocol but may attempt to extract information through passive observation. Nevertheless, it has been suggested that achieving IND-CCA-like security for FHE may be possible by combining it with zero-knowledge succinct non-interactive arguments of knowledge (SNARKs) [31].

3. Software Frameworks

3.1. Framework Selection

The selection of Secretflow-SPU was mainly driven by its specialization in ppML as well as its ease of use, which is introduced in a later subsection. Another strong candidate was MP-SPDZ [2], which supports more protocols and adversary settings, but it is much harder to use for ppML as it is mostly specialised for benchmarking MPC primitive. A further candidate was Crypten [15], which is another ppML MPC framework developed by Facebook. Crypten was not selected due to its recent deprecation and halt of further development. Due to these reasons SPU was selected as the framework of choice for MPC. The largest FHE libraries consist of OpenFHE [32], HEIR [33], SEAL [34] and the library family of Zama: TFHE-rs [35], Concrete [36] and Concrete-ML [37], where the second and third are relying on the first one. Out of those libraries only Concrete-ML implements ready to use machine learning functionality. It implements the TFHE scheme, which is a good suitor for ML applications due to its support of arbitrary function evaluation capabilities and fast bootstrapping procedure. Therefore, we argue it to be the best FHE library for those purposes and consequently use it in the subsequent library examination and benchmarks, with exceptions for the distance measures where Concrete is utilized.

3.2. Concrete-ML

Concrete-ML is, to the best of our knowledge, the only open source library for FHE specializing in machine learning. It relies on the underlying libraries Concrete and TFHE-rs, whereas the latter consists of the implementation of the TFHE scheme. Concrete implements the compiler, with which it is possible to translate Python functions into their FHE equivalent. Concrete-ML specializes in this compiler for machine learning. All three libraries are developed by Zama under the BSD-3 license. The base implementation in TFHE-rs is done in Rust for safe and fast calculation. The compiler is based on multi-level intermediate Representation (MLIR). Users, however, use a simple, high-level Python API. Additionally, Concrete-ML already offers ready-to-use regression and classifier models, which are based on Scikit-learn implementations. Furthermore it is possible to use one's own Torch [38], Onnx or Brevitas [39] deep learning models. However, there are some restrictions imposed on those models. Concrete-ML only offers programmable bootstrapping with integers up to 16 bits. In order to have some room for integer growth from multiplications and additions, all inputs, activation functions and weights have to be quantized to small bit ranges. Practitioners here can balance speed, with fewer bits, and accuracy, with more bits. Furthermore, Concrete-ML already provides communication protocols, serialization logic and a simple API to automatically run inference remotely after a client and server are implemented. Additionally so-called hybrid models are offered, which can be used

with large models to split inference between the client and server. Specifically, parts are evaluated in plaintext on the client side and parts are calculated encrypted on the server side. From a security perspective, all inference is run using a 128-bit security level, which cannot be changed through API usage. IND-CPA-D security is ensured. Concrete-ML does not implement any model protection. Therefore, weights and model architecture remain plain and thus are accessible to the hosting party. Consequently, in instances where the client aims to outsource self-trained models, the server has to be trusted not to steal them. For servers offering secure model inference, black box attacks are a security concern, as clients have access to inputs and outputs to a given query.

3.3. Secretflow-SPU

Secretflow-SPU [40] is a versatile framework for privacy-preserving machine learning (ppML) that streamlines the creation of MPC-based models. The architecture consists of a frontend compiler providing a Python API and a backend execution runtime. The frontend transforms machine learning programs into a specialized intermediate representation called privacy-preserving high-level operations (pphlos). This representation is then interpreted by the backend, which runs the operations on a virtual device formed by a cluster of nodes utilizing a customizable MPC protocol.

The process of compiling into this intermediate representation draws inspiration from non-MPC machine learning compilers like Google's XLA (<https://github.com/openxla/xla> (accessed on 16 June 2026)). Such compilers are designed to simplify model development across various frameworks and hardware architectures. XLA utilizes high-level operations (HLOs) as its intermediate representation to capture the computational graph of a model. This graph then undergoes hardware-specific optimizations before being lowered into machine code for execution on CPUs, GPUs, or TPUs. SPU follows a comparable path: it leverages XLA to create the initial HLO representation and then introduces further compilation phases. These stages annotate the graph with specific privacy semantics and data types to ultimately generate the pphlo representation.

By default both the inputs and model weights are secret-shared. The model architecture is kept in plaintext as it is part of the executed code.

3.4. Software Framework Comparison

To evaluate and compare both frameworks, we established several specific criteria. We first assessed the library size by calculating the total lines of code using the command line tool cloc [41], which distinguishes between blank lines, comments, and actual source code across different programming languages. The languages identified provide insight into both the developer interface and the underlying backend implementation. We further evaluated the frameworks based on documentation quality, maintenance activity, and the variety of supported models. Our documentation assessment included an analysis of official guides, code examples, inline comments, and community forums. To determine the level of maintenance, we examined the frequency of Git commits, version history, and the status of open issues. The supported models category highlights the functional capabilities of each framework by identifying the specific machine learning architectures, it can integrate. Finally, the Ease of Use metric provides a summary of developer approachability, encompassing the learning curve, API intuitiveness, and initial setup complexity.

Lines of code. As we can see in Table 1, Concrete-ML is much larger than Secretflow-SPU. The repository of Concrete-ML itself makes up only a very small chunk of that. Most of the code comes from the underlying backend of TFHE-rs, implementing all calculation logic, and Concrete, implementing the compiler.

Programming language. Both frameworks use the same frontend API language Python. However, the backend of Secretflow-SPU uses C++, while Concrete-ML uses mainly Rust with some C++. Thus, both have an easy-to-use user API while relying on fast and safe languages in the backend.

Table 1. Comparison of both software libraries. We evaluated multiple criteria including lines of code, programming languages used, documentation quality, maintenance, supported models and ease of use.

	SPU	Concrete-ML
Lines of code	≈150k	≈600k
Prog. language	Python (v:3.11,<3.13), C++ (v:C++17)	Python (v:>=3.8.1,<3.13), Rust (v:1.91.1)
Documentation	Minimal + Examples	Extensive + Examples
Maintenance	Semi-regular	Regular
Supported models	JAX	multiple libraries
Ease of use	Simple	Simple

Documentation. The documentation of Concrete-ML is quite extensive [37]. It offers examples for almost all models, some tutorial videos, docs of the API and a forum to ask questions. Next to that, it is also possible to join the.org community channels (<https://fhe.org/community.html> (accessed on 16 June 2026)) to ask for advice there. Secretflow-SPU documentation contains quite a lot of good examples to get started, as well as minimal documentation of the public Python API. However, a portion of the documentation is not translated into English.

Maintenance. Secretflow-SPU is semi-regularly maintained, receiving some commits a few times per month. Most of the commits are basic maintenance commits (updating documentation, upgrading dependencies), some big fixes and a small amount of new features. Albeit SPU belongs to a greater family of Secretflow products and some of their products and features can be used with SPU as well. Concrete-ML, on the other hand, is quite well-maintained. Issues are fixed, commits are made and the newest version was released in early 2025 in April. Historically, there has been a new version released every 3 months. However, this cycle broke after the newest version. We can also see a decline in commit frequency. This might indicate a decreasing support in the future. The underlying framework Concrete has not received any commits since June 2025, but TFHE-rs is actively maintained.

Supported models. In Concrete-ML there is a wide variety of models already implemented, which are based on scikit-learn implementations. These range from linear models, trees and nearest neighbour to neural networks. It is also possible to compile one's own Torch or ONNX deep learning models or even integrate encrypted inference for LLMs like GPT via their hybrid model approach. Yet, Concrete-ML does not implement compilation of complex LSTM and transformer models. SPU uses an HLO intermediate representation to run machine learning programs on the backend. Due to this feature any model that is written in JAX (which natively outputs the machine learning Python program to HLO) is supported by SPU.

Ease of use. Concrete-ML is very simple to set up. It implements an easy-to-understand and use Python API with only a couple of calls to either compile new models or use already integrated ones. Furthermore, the naming system for their integrated models is similar to the scikit-learn library. Most of the cryptography is done under the hood, which means developers need very little knowledge about it in order to be able to use the library. In summary, we can say that it is a high-level and easy-to-use library. SPU is similarly very easy to use. To start writing privacy-preserving machine learning code, only a few

Python decorators are required in order to specify the origin of the input data. Furthermore, another decorator is required to specify which function is required to run under the MPC protocol. In a machine learning context, this is mostly the inference function.

4. Benchmarking

In this section we compare MPC and FHE in regard to machine learning. Consequently, we test four different basic operations, distance measures, which are a common tool to evaluate similarity of feature vectors, different regression models and deep learning networks. All encrypted tests were performed over a large enough sample that the mean is roughly ten times or more the standard deviation. Detailed results of standard deviations and means can be found in Appendix A for Concrete-ML benchmarks and Appendix B for SPU ones. For all figures and tables, the Concrete-ML plaintext inference times were used. In SPU, correctness checks were performed, and all false results were discarded.

4.1. Test Environment

All tests were performed on a workstation running Ubuntu 24.04. It uses a general-purpose Intel Xeon W-2245 CPU, which was released in 2019 and is intended to be used in professional systems. The CPU supports simultaneous multithreading on 8 cores and 16 threads in total. Cache sizes are 32 KB, 1 MB, and 16.5 MB for the respective levels one to three. No GPU was present or used in the tests, and the RAM was 125 GB. Tests utilized all available cores and threads with no additional specifications apart from automatically applied library defaults. All benchmarks were preceded by a warm-up period of 1 to 5 rounds to stabilize performance measurements. All MPC benchmarks were conducted in a localhost environment. For testing network delay, the UNIX utility *tc* was used. FHE benchmarks consist only of the evaluation of the calculation, as well as the encryption and decryption of inputs and outputs. Compilation or key generation times are not considered because both are generally done only once in a setup phase and not during production usage. The latest versions, 1.9.0 and 2.10.0 for Concrete-ML and Concrete, were used. For Secretflow-SPU, version 0.9.4.dev20250225 with Python 3.10.16 was used. For power usage statistics, the command line tool powerstat [42] with the running average power limit (RAPL) interface and continuous tracking every second was used.

4.2. Security Setting

Concrete and Concrete-ML don't allow adjustment of the security level of FHE. The default value, which is to be achieved, is 128-bit security. For our test, no additional configuration was done. Therefore, each benchmark operates on automatically selected parameter sets to ensure this level of security. Both libraries use the lattice estimator [17] library, which is well known in the community, to select all parameters accordingly. Additionally, both libraries guarantee IND-CPA-D security. No protocol-level adversarial model is instantiated, as the experiments focus solely on the computational performance of homomorphic evaluation.

Secretflow-SPU uses underlying protocols for adversary configuration. In our case the ABY3 [43] protocol was selected, which uses a semi-honest, honest-majority type with three computing nodes and a ring of 2^{64} .

4.3. Basic Operations

We first benchmark basic operations of machine learning. Specifically, we test inference times in Secretflow-SPU and Concrete-ML for matrix multiplication and the activation functions rectified linear unit (ReLU), Gaussian error linear unit (GELU) and sigmoid, which are all basic building blocks used in almost every neural network. In FHE these were implemented using the Concrete-ML library, by creating a single-layer neural network

and the respective torch method. For compilation the flags `rounding_threshold_bits` and `n_bits` were set with a value of 8.

Matrix Multiplication. Matrix multiplications are used to, instead of calculating one input at a time, batch data together and evaluate them simultaneously. Given that the amount of data can vary, we benchmarked matrices of size 10×10 , 50×50 and 100×100 . Our results are shown in Table 2. SPU inference is very fast, taking only milliseconds to compute. Notably, with increasing matrix size, the evaluation time only increases slightly. This is not the case for Concrete-ML, where inference times rise much more steeply. Nonetheless, Concrete-ML evaluation performs better for matrices of magnitude of 10×10 . Due to necessary quantization of inputs in Concrete-ML, first to represent integers instead of floats and second to remain in the 16-bit table lookup limit, a slight error was perceived. Specifically, a mean squared error between 0.016 and 0.158 occurred. Notably, smaller matrix sizes in matrix multiplication resulted in larger errors.

Table 2. Calculation time comparison in seconds $\pm$ the standard deviation interval for matmul operations with SPU and Concrete-ML execution time as t_{SPU} and t_{Concr} .

Matrix Size	t_{SPU}	t_{Concr}
10×10	0.0504 ± 0.0050	0.0268 ± 0.00006
50×50	0.0537 ± 0.0049	0.6096 ± 0.00743
100×100	0.0678 ± 0.0057	2.6858 ± 0.01801

Activation functions. The activation function results behave similarly to the matrix multiplication and can be seen in Table 3. While having constant evaluation times across the different functions, it rises linearly with the vector size, making it more impractical the more inputs one has. SPUs' calculation times, on the other hand, are constant for different input sizes, but are different in regard to the activation function. The fastest activation in SPU is ReLU. Overall, SPU is much faster than Concrete-ML. The mean squared error for all activation functions in Concrete-ML was between 0.003 and 0.103. The sigmoid function performed the best and GELU the worst.

Table 3. Calculation time comparison in seconds $\pm$ the standard deviation interval for activation functions with SPU and Concrete-ML execution times as t_{SPU} and t_{Concr} .

Function	Vec. Size	t_{SPU}	t_{Concr}
ReLU	8	0.0467 ± 0.0057	2.4972 ± 0.064
	16	0.0452 ± 0.0062	4.6311 ± 0.075
	32	0.0478 ± 0.0052	7.9786 ± 0.146
GELU	8	0.0623 ± 0.0056	2.4996 ± 0.018
	16	0.0626 ± 0.0054	4.5809 ± 0.065
	32	0.0628 ± 0.0054	7.8465 ± 0.142
Sigmoid	8	0.0752 ± 0.0053	2.5071 ± 0.012
	16	0.0753 ± 0.0051	4.5843 ± 0.067
	32	0.0757 ± 0.0052	7.9062 ± 0.143

4.4. Feature Distance Metrics

This section compares another important tool in machine learning. Distance measures are often used for classifications or to compare feature vectors. Being able to compute them over encrypted data in a reasonable time is important to keep results private. An example could be extracting a biometric feature vector of a person. We implement the most commonly used cosine distance, l2 distance and l1 distance for the vector sizes of 128, 256 and 512 with 32-bit fixed-point numbers in SPU. This, however, is not possible in Concrete-ML, where we thus implement the same vector sizes over 8 bits for the l1 distance, 4 bit/3 bit

for the l2 distance and 5-bit integers for the cosine distance. Compilation and tests were performed using the Concrete library with the compiler flag `loop_parallelize` as true. The results can be seen in Table 4. As can be observed, SPU is significantly faster than Concrete-ML in every measurement. Calculation times are very much constant for different vector sizes and differ only slightly between metrics. In Concrete we can observe a linear increase in execution time depending on the vector size across all measurements. Overall calculation times are very slow compared to SPU. This can be attributed to the TFHE scheme Concrete implements. It focuses on Boolean arithmetic and thus gate evaluations. Highly arithmetic calculations, albeit being possible, are not optimized and especially larger bit sizes are problematic in TFHE. Consequently, doing a lot of multiplications, summations and exponentiations is highly inefficient, as can also be observed in Table 4. Coupled with the 16-bit limit on programmable bootstrapping (look-up table evaluation), for example, the square root or division, only small input bit widths are possible. For l1 and vectors of 512 only 3 bits and 4 bits for the other configurations were possible. Similarly cosine distance evaluation was limited to 5 bits. Each calculation could be sped up by doing certain calculations, especially the lookup table evaluations, in the clear. However, this would break security if done on the server and also would often break the applied use case. It might be possible to achieve more efficient execution using a different FHE scheme. For example, leveled schemes are designed for arithmetic circuits and also implement ring learning with error (RLWE), which supports SIMD-like operations.

Table 4. Comparison of calculation times in seconds $\pm$ the standard deviation interval on different distance measures with different input vector sizes for Plaintext, SPU and Concrete-ML execution times as t_{Plain} , t_{SPU} and t_{Concr} .

Distance	Vec. Size	t_{Plain}	t_{SPU}	t_{Concr}
Cos	512	0.0003	0.0955 ± 0.0082	9284.41 ± 23.558
Cos	256	0.0003	0.0935 ± 0.0080	5385.89 ± 50.023
Cos	128	0.0003	0.0938 ± 0.0080	2001.38 ± 19.942
L2	512	0.0002	0.0779 ± 0.0073	461.38 ± 3.6780
L2	256	0.0002	0.0769 ± 0.0073	275.58 ± 4.4374
L2	128	0.0002	0.0768 ± 0.0073	115.97 ± 1.7230
L1	512	0.0001	0.0660 ± 0.0085	235.02 ± 1.1084
L1	256	0.0001	0.0639 ± 0.0084	105.3 ± 0.8884
L1	128	0.0001	0.0638 ± 0.0079	45.2 ± 0.4681

4.5. Regressions

Now we compare evaluation times of simple regressions for the two secure computation technologies and plaintext inference. Specifically, we will look at the two simplest and most common regressions, those being linear and logistic. Furthermore, with the random forest generator, a more complex model is benchmarked. In all tests we vary the number of features and additionally the number of estimators for the random forest model. Given Concrete-ML implementing the scikit-based models with plaintext and ciphertext inference modes, we will use those for the plain and Concrete-ML benchmarks. A quantization of 8 bits is used with no additional compiler flags. For faster execution, a smaller one could be applied. With the applied quantization, no deviations from quantized plaintext inference were observed across all regressions in Concrete-ML. In other words, ciphertext execution does not introduce additional inference error. The SPU benchmark is implemented using the JAX library by creating a single-layer network without an activation function for linear and with Sigmoid for logistic regression. Here 64-bit integers are used and no error was observed compared to plaintext inference.

Linear Reg. Table 5 illustrates the results of the linear regression benchmarks. The plaintext inference is very fast and slightly increases with more features. SPU follows its trend of having constant evaluation times with execution times of around 0.079 s, whereas Concrete-ML takes longer the more features there are. However, it remains faster than SPU in all but one configuration, which is most pronounced for a small number of features, where inference is up to 17 times faster. Both secure computation frameworks take only minimal time to compute linear regression and could be used for real-world appliances.

Table 5. Evaluation time comparison in seconds $\pm$ the standard deviation interval for linear regression with Plain, SPU and Concrete-ML execution times as t_{Plain} , t_{SPU} and t_{Concr} .

Features	t_{Plain}	t_{SPU}	t_{Concr}
10	13×10^{-5}	0.0791 ± 0.0090	0.0049 ± 0.00046
50	13×10^{-5}	0.0790 ± 0.0091	0.0105 ± 0.00032
100	13×10^{-5}	0.0786 ± 0.0091	0.0182 ± 0.0004
250	16×10^{-5}	0.0793 ± 0.0090	0.0422 ± 0.0008
500	22×10^{-5}	0.0794 ± 0.0088	0.0845 ± 0.0023

Logistic Reg. Looking at Table 6, logistic regression behaves analogously to linear regression. SPU inference times remain constant, and Concrete-ML ones continue to follow a linear increase across all feature amounts. Notably, execution times for SPU are higher here than those observed for linear regression, whereas Concrete-ML exhibits slightly better runtime performance. This widens the gap between the two secure computation technologies, providing a stronger rationale for adopting Concrete-ML.

Table 6. Evaluation time comparison in seconds $\pm$ the standard deviation interval for logistic regression with Plain, SPU and Concrete-ML execution times as t_{Plain} , t_{SPU} and t_{Concr} .

Features	t_{Plain}	t_{SPU}	t_{Concr}
10	14×10^{-5}	0.1119 ± 0.0098	0.0046 ± 0.0002
50	14×10^{-5}	0.1120 ± 0.0094	0.0096 ± 0.0003
100	14×10^{-5}	0.1121 ± 0.0098	0.015 ± 0.0003
250	17×10^{-5}	0.1127 ± 0.0091	0.0372 ± 0.0015
500	23×10^{-5}	0.1128 ± 0.0098	0.0771 ± 0.0038

Random Forest. The random forest regressor is also a commonly used model and is applied for banking, medicine, trading, and e-commerce [44]. The results can be seen in Figure 1. The first row shows inference times over plaintext and the second over ciphertext. The left column illustrates the calculation time in relation to the number of features when the number of estimators is constant. The right column displays the exact opposite case. As can be seen, the plaintext inference times rise linearly with more features, but stay fairly constant for different estimator amounts. This behaviour is seemingly flipped for the Concrete-ML computation. This can be explained by looking at the structure of the random forest regressor and Concrete-ML. The regressor consists of tree structures over a random sample of features, the estimators, which are traversed during the inference. This requires a lot of comparisons and some simple additions. In Concrete-ML comparisons are very expensive. Thus, having more estimators increases overall inference times. The same might be the case with having more features and thus bigger trees, but apparently only slightly. In plaintext, this problem does not arise and parameter selection or aggregation might create more overhead here. Overall, we can see that encrypted evaluation takes roughly one to 60 s. Detailed evaluation times can be found in Appendix A, Table A1. The increased inference calculation times for large amounts of estimators make this regressor impractical to use in real-world applications. However, small amounts of estimators are efficient

enough. SPU does not implement the random forest in a Jax library and therefore, was omitted due to time constraints.

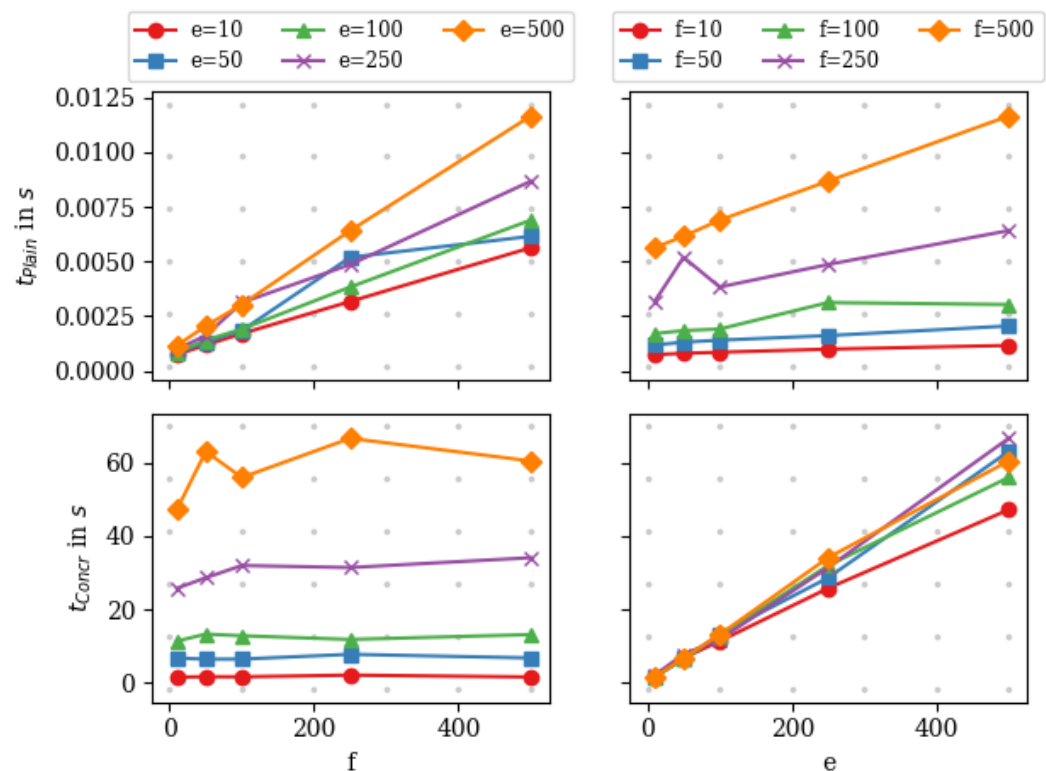


Figure 1. Benchmarks of random forest regressor inference times in plain (t_{plain}) and FHE encrypted (t_{Concr}) settings, varying the number of features = f and estimators = e .

4.6. Deep Learning

This section presents benchmarks of several fundamental deep learning models, focusing on ubiquitous and versatile architectures: dense, convolutional, and long short-term memory (LSTM) networks. These models were evaluated across a range of depths and widths. For the FHE-based benchmarks, the Concrete-ML compiler was configured with 6-bit quantization (specifically the flags `rounding_threshold_bits` and `n_bits`), a setting chosen to prioritize computational performance despite a potential decrease in accuracy. This specific configuration was necessary because higher quantization levels caused the compiler to fail during the compilation of models into FHE circuits. These failures typically occurred when internal calculations surpassed the 16-bit limit required for table lookups or when the compiler could not identify a valid set of security parameters within the defined constraints. Other than that, no additional compiler flags were used. The errors perceived in the subsequent benchmarks represent those induced by ciphertext inference and do not necessarily reflect the potential accuracy achieved by a trained model. Such will be examined in a subsequent chapter.

Dense Models. Dense models, often referred to as multi-layer perceptrons (MLPs), are composed exclusively of linear layers separated by activation functions, specifically ReLU in our instance. Our benchmarks evaluated these models across a spectrum of trainable parameters (1000–10,000) and varying depths (1–4 layers). As depicted in the first row of Figure 2, plaintext inference exhibits standard linear behaviour: execution time scales linearly with the number of layers but remains constant as the parameter count increases within a fixed layer structure. The second row of Figure 2 displays the SPU results, maintaining the same column alignment. The performance trends here mirror those of the

plaintext evaluation; although total inference time increases, the underlying relationships regarding layer depth and parameter volume remain consistent.

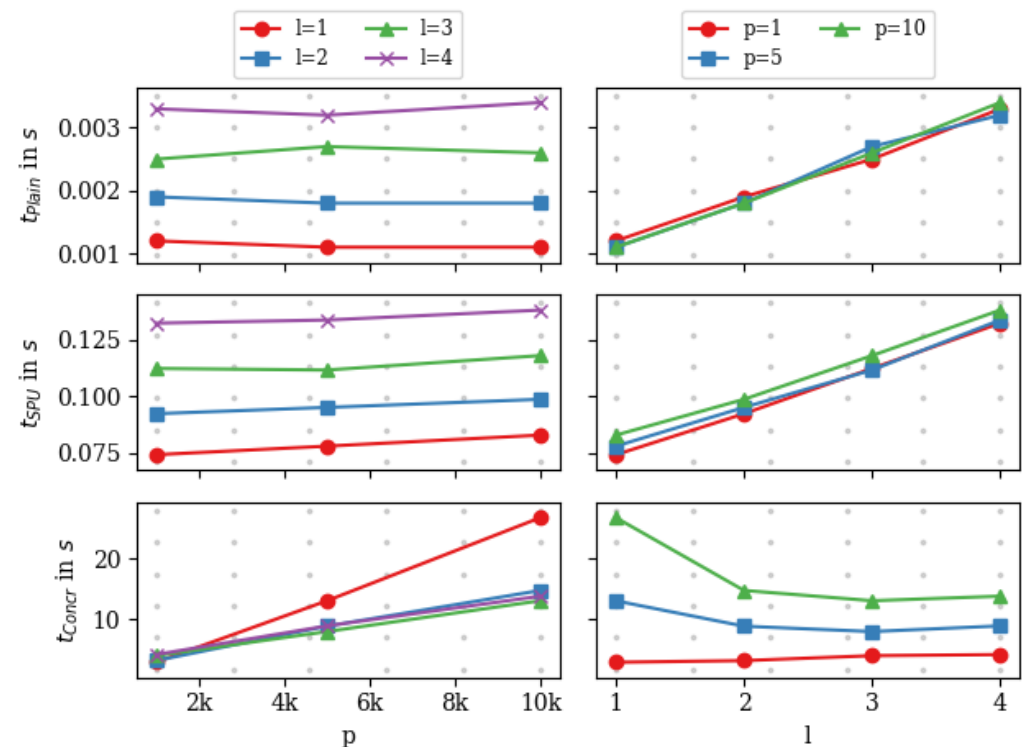


Figure 2. Benchmarks of dense model inference times in plain (t_{plain}), MPC encrypted (t_{SPU}) and FHE encrypted (t_{Concr}) settings, varying the number of parameters = p and layers = l .

In contrast, the Concrete-ML results in the final row reveal a distinct shift in behaviour. Notably, single-layer models show higher inference times across both subfigures. Furthermore, the number of layers does not significantly influence the execution time, whereas the number of parameters has a linear impact across all tested models. This effect is most pronounced in the one-layer network. This disparity likely stems from the high computational cost of large matrix calculations, which are, as we have seen in Section 4.3, very inefficient in Concrete-ML. Moreover, since inference was conducted on a CPU with only 15 cores, the capacity for parallel execution is heavily constrained when processing up to 1000 nodes. For dense model inference in Concrete-ML, a mean squared error (MSE) of a maximum of 0.0062 was observed, which represents the error induced by ciphertext inference. Detailed measurements are illustrated in Table A2.

Convolutional Models. Convolutional neural networks (CNNs) represent a category of architectures primarily utilized for image processing, centered around convolutional layers as their fundamental building blocks. For these benchmarks, a standard model was designed to process $3 \times 112 \times 112$ images. Each individual layer within this generic CNN configuration includes a convolution operation, a max pooling operation, and a ReLU activation function. The evaluation explored variations in both model depth and the total number of parameters. The first row of Figure 3 illustrates the plaintext results. Much like the MLP findings, the correlation between the number of layers and inference time is approximately linear, whereas the inference time remains relatively constant regardless of the parameter count. The SPU results, shown in the second row of Figure 3, demonstrate a similar linear progression when increasing the number of layers. However, unlike the plaintext implementation, the relationship between the number of parameters and inference time in SPU also transitions to a linear behaviour.

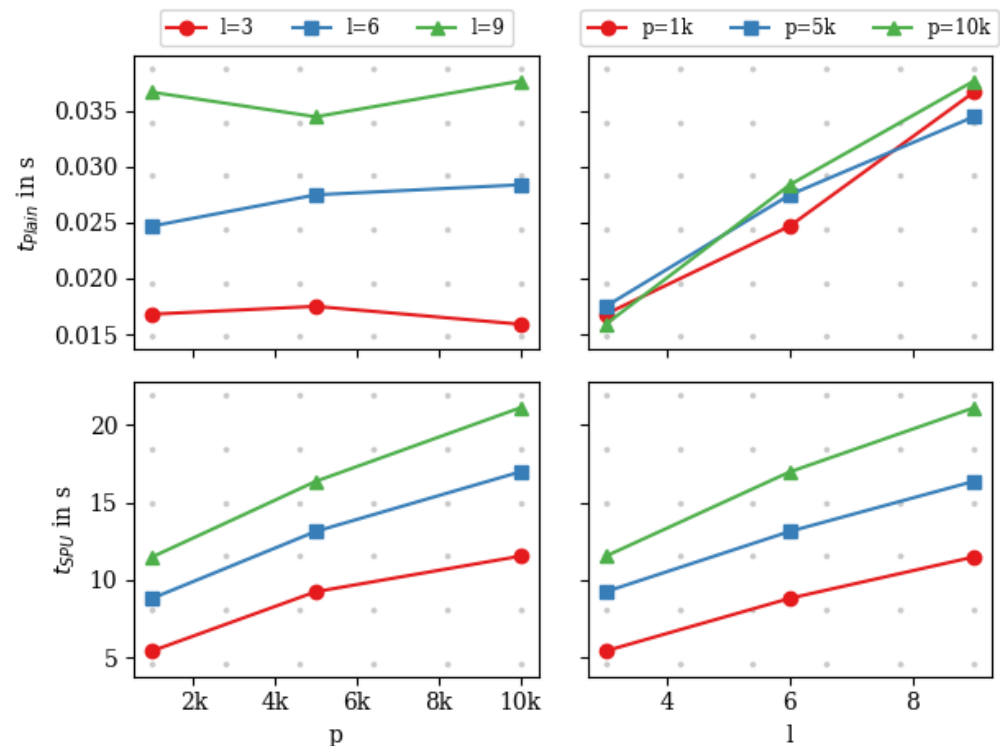


Figure 3. Benchmarks of CNN inference times in plain (t_{plain}) and MPC-encrypted (t_{SPU}) settings, with varying the number of parameters = p and layers = l .

Benchmarking Concrete-ML inference times for $3 \times 112 \times 112$ images proved too time consuming to yield meaningful results. Consequently, we adjusted the benchmark to evaluate varying image dimensions and parameter counts using the same model architecture. We tested input sizes of $3 \times 8 \times 8$, $3 \times 16 \times 16$, and $3 \times 32 \times 32$ pixels. Due to these reduced dimensions, we could not evaluate multiple layers, as deeper structures caused inputs to zero out, leading to non-representative timing data. The results are presented in Figure 4, with the left column displaying evaluation time relative to image size and the right column relative to parameter count. The first row contains plaintext data, while the second contains encrypted data. Both plaintext and encrypted inference exhibit similar trends; as image size increases, evaluation time grows roughly quadratically in both cases. Regarding parameter scaling, we observed a sharp increase in calculation time, approximately 2 to 3 times higher, when moving from 1000 to 5000 parameters, followed by a more moderate increase of about 1.5 times when reaching 10,000. This suggests a potential square root relationship between parameter count and inference latency. Notably, even with these small input sizes, inference times can exceed 10,000 s, rendering them impractical for real-world applications. CNN-encrypted inference induces some error from quantization inaccuracies. We calculated the mean squared error (MSE) and it is in the range from 0.0042 to 0.0146. It is illustrated in Table A4.

LSTM Models. Long short-term memory models are a deep neural network type that is mostly suitable for time-series analysis, as they can effectively handle ordered datasets. For benchmarking, a few simple models were constructed. The model generally accepts inputs in shape (1, 1, 14). A single layer consists of a single LSTM cell and the number of parameters is adjusted by changing the output size of a cell. The benchmarking results can be seen in Figure 5.

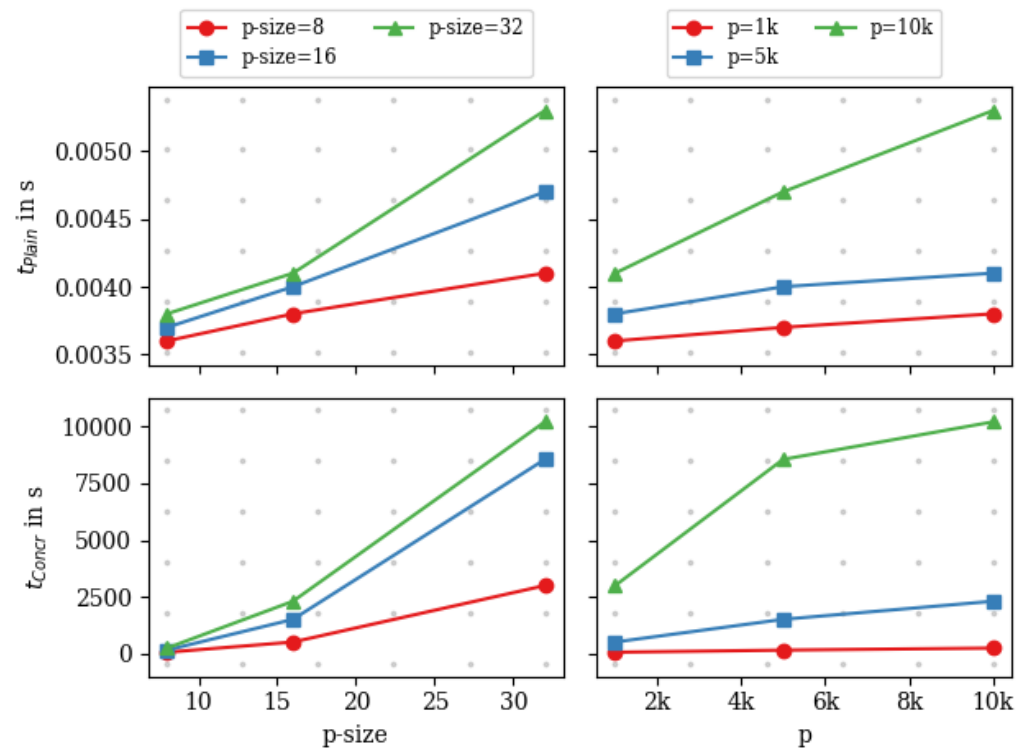


Figure 4. Benchmarks of CNN inference times in plain (t_{plain}) and FHE-encrypted (t_{SPU}) settings, with varying the number of parameters = p and picture sizes = s .

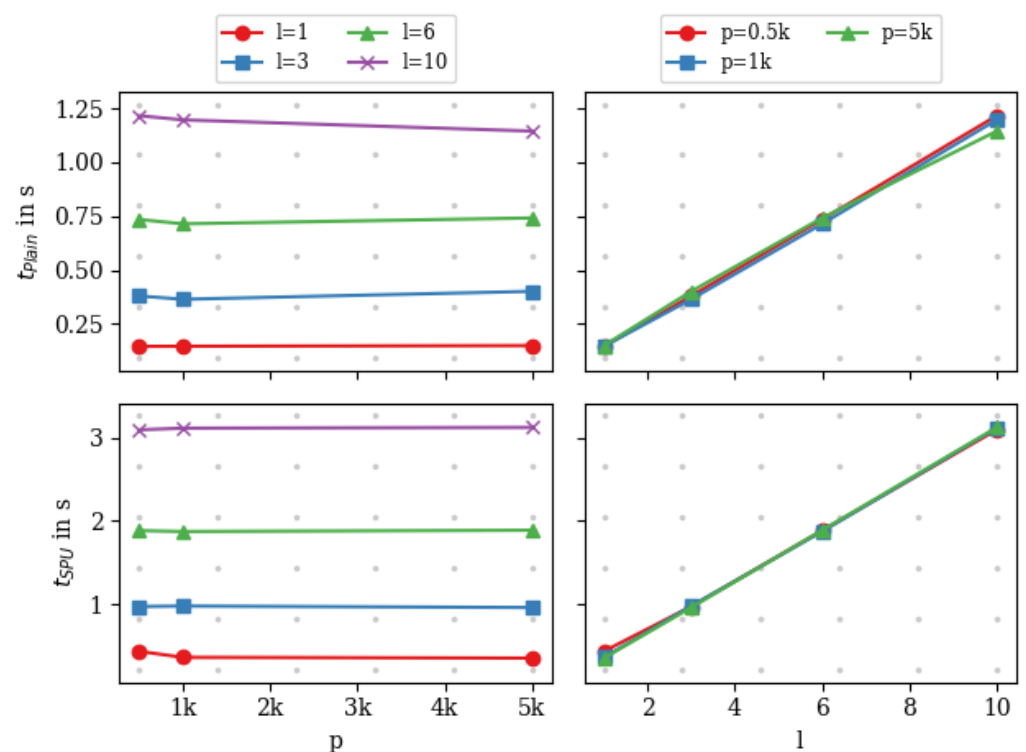


Figure 5. Benchmarks of LSTM inference times in plain (t_{plain}) and MPC-encrypted (t_{SPU}) settings, with varying the number of parameters = p and layers = l .

From the results it can be observed that LSTMs in this parameter range behave similarly to the MLP model introduced earlier. A constant behaviour can be seen when looking at the relationship between the number of parameters and inference time, both in

plaintext and SPU scenarios. When it comes to the number of layers, a linear relationship can be seen again in both scenarios.

The used library Concrete-ML does not currently support the torch layer LSTM for compilation, which is apparent in their implementation. While it might be possible to integrate them manually, we decided to leave it out and restrict our benchmarks to the supported layers.

4.7. Scaling and Real-World Performance

4.7.1. Secure Multiparty Computation

Network. MPC's main disadvantage is the reliance on distributed computing, which results in a strong influence from network artifacts such as network delay and packet loss. To illustrate this, a benchmark was conducted using the CNN model previously discussed, but with a bigger network size. The effect of network delay is visualized in Figure 6. Here it can be observed that network delay introduces an additional latency overhead, effectively shifting inference times upward. Larger networks are affected more strongly, reaching inference over 35 s with a 20 ms delay.

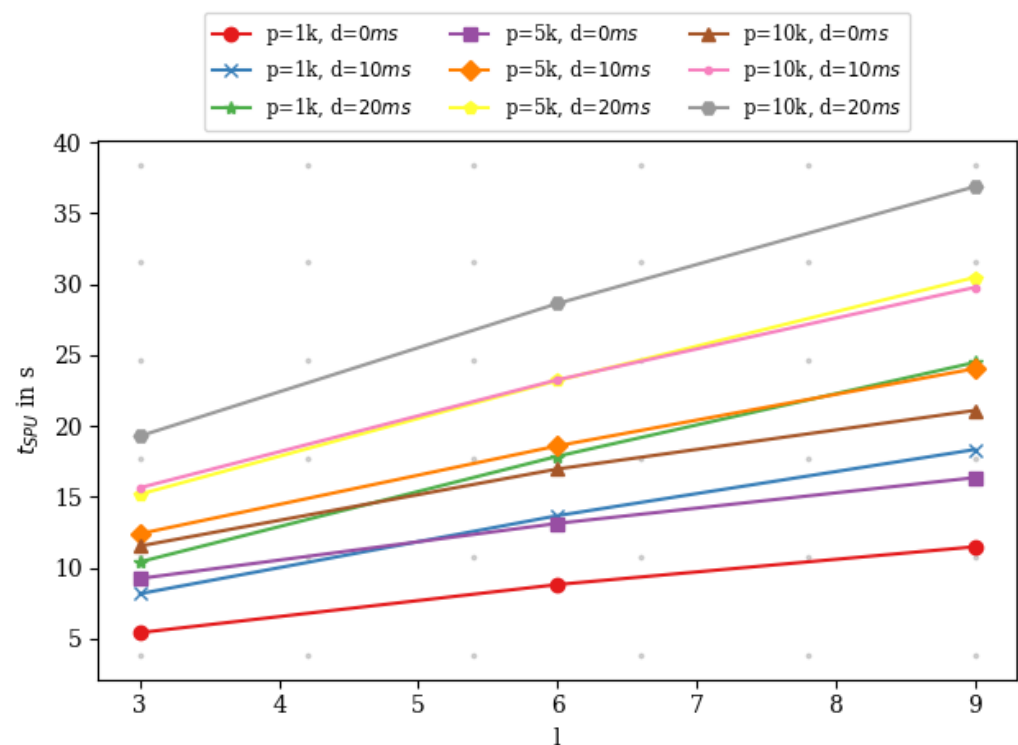


Figure 6. Benchmarks studying the effect of network delays (d) on CNN inference times in MPC-encrypted (t_{SPU}) settings with a varying number of layers = l and parameters = p.

The effect of packet loss is visualized in Figure 7. Similar to network delay, packet loss results in a systematic increase in inference times, with larger networks being affected more strongly.

To also showcase the network requirements of a CNN model, a throughput graph can be seen in Figure 8. Here a clear trend can be observed, where the increasing number of layers decreases the network throughput, while larger network sizes result in increased throughput.

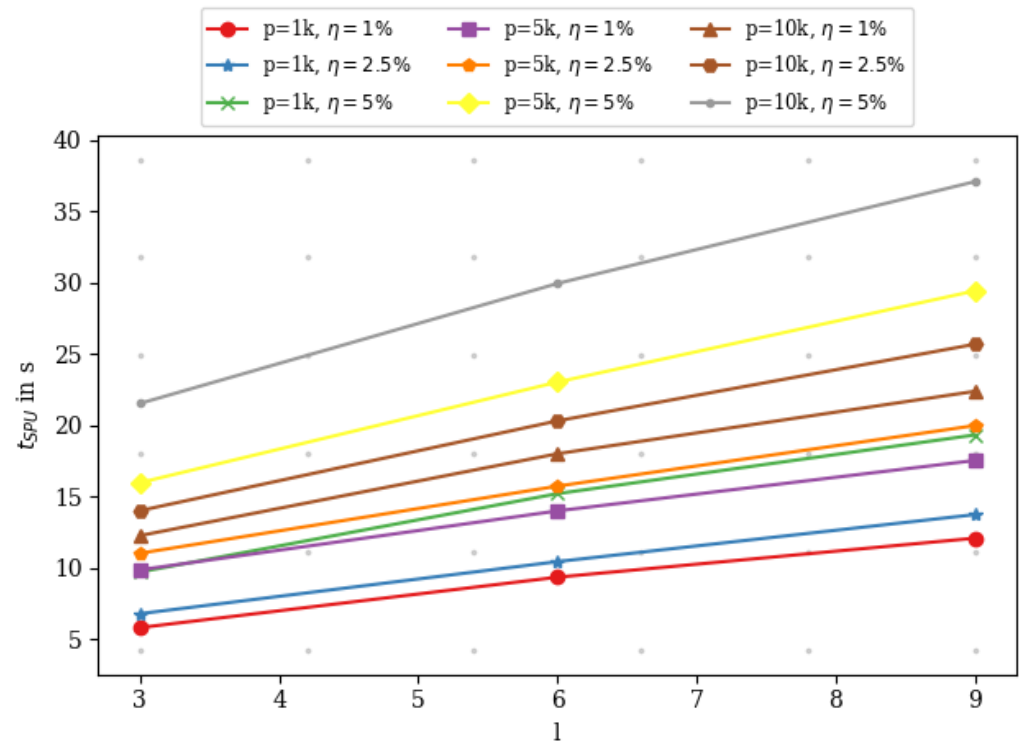


Figure 7. Benchmarks studying the effect of varying packet loss percentages (η) on CNN inference times in MPC-encrypted (t_{SPU}) settings with a varying number of layers = l and parameters = p .

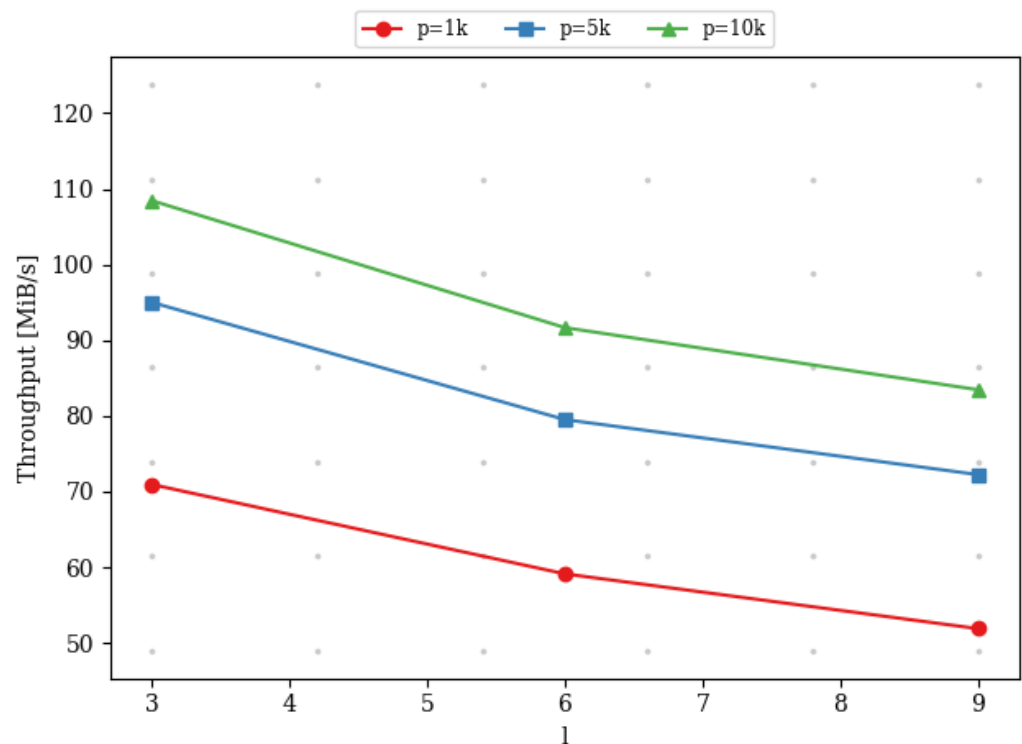


Figure 8. Benchmark showing the throughput measured on the localhost network across different parameter = p and layer = l sizes.

Complex Models. A big advantage of Secretflow-SPU is the flexibility of the intermediate representation system. This allows running quite complex real-life models in a reasonable time when talking about a localhost setting. Example runtimes for a few neural networks can be seen in Table 7. Here various models are benchmarked.

ResNet50 [45] is a well-known convolutional architecture used in computer vision. MobileFaceNet [46] and GhostFaceNet [47] are both small mobile networks specialized in face verification/recognition scenarios. Visual Transformer (ViT) [48] is a variation of the Transformer architecture transformed for computer vision tasks and convolutional vision transformer (CvT) [49] is an improved version of ViT, utilizing additional convolutions for increased performance. In the table, one optimization is also introduced and showcased. In SPU one can choose which inputs to the inference function should be kept private and which ones should be kept public. For open-source models it might be beneficial to keep the trained weights public and therefore speed up the inference time. Some models like GhostFaceNet benefit from this optimization greatly, having an inference time of 6.86 s. This optimization inherently introduces a privacy-performance trade-off that is specific to the application used and should be taken into consideration when designing the SPU system.

Table 7. Inference times in seconds $\pm$ the standard deviation interval under SPU for different models. Hybrid denotes encrypted inference with plaintext weights.

Name	t_{enc}	t_{hybrid}	t_{plain}
ResNet50	41.0637 ± 0.3600	26.2798 ± 0.2693	0.3265
MobileFaceNet	24.6422 ± 0.2144	17.0413 ± 0.1844	0.1682
GhostFaceNet	19.1515 ± 0.4033	6.8552 ± 0.2073	0.5079
ViT	65.3405 ± 0.5238	59.0618 ± 0.4334	0.0230
CvT	139.8947 ± 1.6031	129.7735 ± 0.9959	0.5274

Ring size. Secretflow-SPU supports three different ring sizes. For each ring size, the root mean square error (RMSE) between plaintext result and SPU result was captured along with the average inference time. The results can be seen in Table 8. From the comparison, it is clear that across all three ring sizes, the inference time stays relatively stable for this particular MLP configuration. On the other hand, the increase in ring size introduces substantial improvement in terms of RMSE when compared to the plaintext result of the same network.

Table 8. Comparison of different ring size and its effect on inference time and RMSE in comparison to the plaintext result. The tested network is a MLP with input size 10, output size 1 and two hidden layers, each with 40 neurons.

Ring Size	t_{spu}	RMSE
2^{32}	0.1206 ± 0.0122	1.70×10^{-2}
2^{64}	0.1205 ± 0.0122	2.35×10^{-5}
2^{128}	0.1226 ± 0.0123	8.15×10^{-8}

4.7.2. Fully-Homomorphic Encryption

Hybrid Models. We can see that Concrete-ML is very fast for simple regression prediction. Furthermore, we can observe that dense models and especially CNNs seem to perform much worse under Concrete-ML. Even with added network delays in SPU, there is still a major gap. Additionally, real-world models are often more complex than the ones we have benchmarked. To mitigate this, Concrete-ML implements hybrid models. Here the model is partitioned such that certain layers are calculated by the server and others by the client. All client-side evaluation is done on plaintext, whereas server-side calculations are done encrypted. In this workflow the client is responsible for encryption and decryption of ciphertexts as well as calculating intermediate results. Before each server-side task, all data is encrypted and only decrypted again on the client side. Consequently, the server only ever works on and sees ciphertexts. Under the assumption of an established semi-honest server and given IND-CPA-D security of the implemented TFHE scheme, the server is computa-

tionally unable to recover any information about the plaintext data from the ciphertexts it processes. However, as models are stored in plaintext, some data is leaked there. For a client aiming to outsource model inference, hybrid models in Concrete-ML can reduce the attack surface for model IP exposure by splitting computation between client and server, instead of outsourcing it completely. However, model stealing attacks remain possible, and hybrid execution does not provide formal guarantees for model intellectual property protection. Conversely, a cloud provider offering secure inference of different models would need to expose parts of the models for hybrid settings, thereby reducing model intellectual property protection. However, even in fully server-side deployments, models may still be vulnerable to black-box model stealing attacks. Still, hybrid models are primarily seen to offer inference time speed-up. To achieve the best possible improvement, client-side execution should contain FHE hard calculations such as non-linear functions (e.g., activation functions). As an example, we benchmarked the same MLP model and its configuration as above, with only the ReLU function being calculated in plain on the client-side. In Figure 9 the results are shown. Both exhibit the same behaviour, as when doing everything encrypted, but the execution time is significantly lower. Here we can see inference times in the range of 20 to 180 ms. This is even faster than evaluation under SPU. These benchmarks however do not include network delays, as they were done in a localhost setting, which would increase inference time in a non-trivial manner depending on the server-client communication latency. Nevertheless, there are at maximum only two communications needed per torch layer, which is significantly less than in SPU, where each multiplication requires one. This results in less traffic and thus overhead. It is also possible to use the hybrid model for CNN inference, but there, we can see a much smaller impact on the inference times because of the convolutional layers. Even though they are linear transformations, they consist of thousands or even millions of multiplications depending on the size of the input and output channels, as well as the kernel and picture size. Calculating this many takes very long in Concrete-ML and is considerably more expensive than activation functions. Still, some speed-up through hybrid models could be achieved.

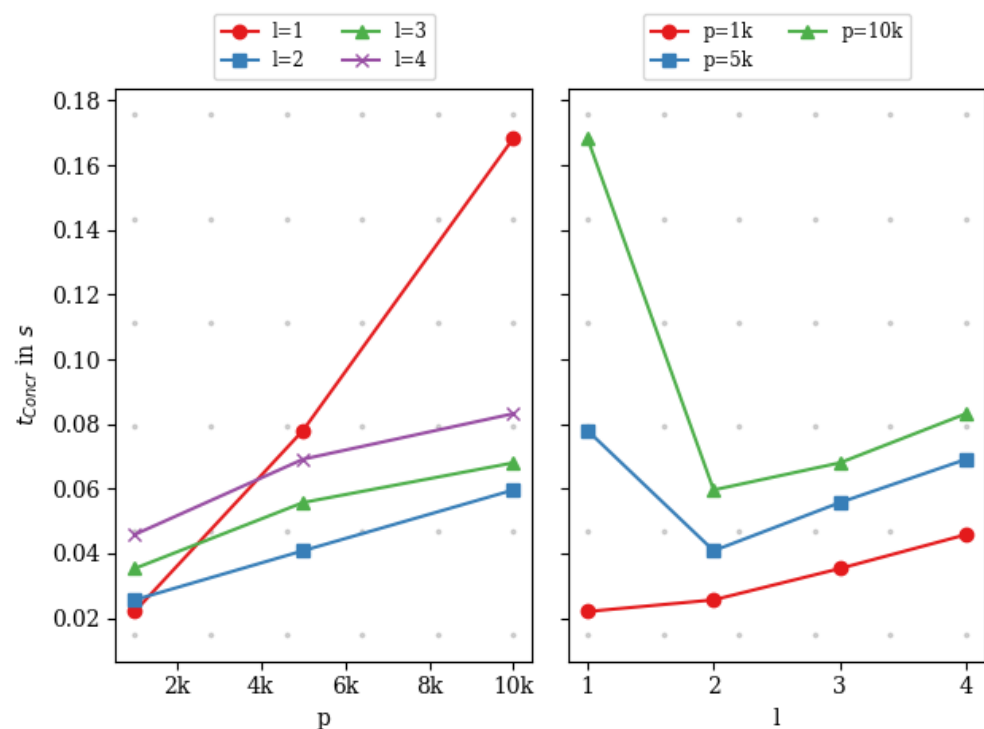


Figure 9. Concrete-ML encrypted inference on MLP hybrid models.

Quantization. Another factor of TFHE and especially the Concrete-ML implementation, is its integer size restriction. Only integers up to a 16-bit size can be operated on to allow for programmable bootstrapping. Consequently, all machine learning applications in Concrete-ML use quantization to convert floating-point computations into low-precision integer operations compatible with encrypted arithmetic. However, this could lead to evaluation inaccuracies. On the other hand, it can shrink model size and increase inference efficiency. In Concrete-ML, quantization is implemented by choosing a bit width and applying it after the training phase (post-training quantization (PTQ)) or before it using the *brevitas* [39] library (quantization aware training (QAT)). According to the documentation, the latter choice is seen as the better one in terms of accuracy, because the model can adjust to these inaccuracies during training. To show quantization effects on model inference results, we benchmarked the linear regression with different bit widths ranging from 3 to 11 bit. Figure 10 shows our results. The left y-axis with the cyan graphs illustrates the evaluation times of regression models with regards to the bit width with different feature amounts, just like in Section 4.5. A linear increase can be seen, which is most pronounced in models with larger feature amounts. The orange graphs and right y-axis illustrate the coefficient of determination or R^2 score, which is a common metric used in machine learning to determine how well the model explains the data compared to the mean error of the results of the encrypted quantized model in comparison to the plaintext not quantized one. It can be observed that linear regressions with 500 features are very inaccurate in both instances. Additionally, 250 features models are also not as accurate as lesser feature amounts. For encrypted inference we can see a steep increase in the R^2 value aligning with the plaintext evaluation, with the alignment becoming more pronounced as the quantization bit width increases. Sometimes the encrypted R^2 score even exceeds the plaintext one. Measured mean absolute error (MAE) and maximum absolute error (MaxAE) follow a similar trend. The more features, the larger the errors and the bigger the quantization bit width, the smaller the errors. For example, with a bit width of 3 bits, a MAE of ≈ 32 –380 and MaxAE of ≈ 155 –1753, which would not qualify as functionally equivalent. However, it should also be considered that regressions with more features lead to an increase in output magnitude, making the larger perceived errors consistent in relative terms. For a quantization bit width of 8 bit, the measured MAE and MaxAE only amount to ≈ 1 –10 and ≈ 5 –45, which is much smaller and can be considered functionally equivalent to the plaintext regression. These errors decrease further with even larger bit widths.

We did the same for dense models. At first we tried using our previous benchmarked model and expanded it for quantization tests. However, no good results were achieved using various error metrics. We therefore switched to the already implemented *Neural-NetworkRegressor* model of Concrete-ML, which implements an MLP network. Again different layer amounts were tested, as well as the quantization bit width from 3 to 6 bits. For each configuration the model was trained on *sklearn*-generated regression data with 2000 epochs. After that, the coefficient of determination, or R^2 score, was calculated. In Figure 11 we can see that inference is faster than the previously benchmarked MLP model. This can be attributed to the reduced number of parameters (minimum ≈ 11 and maximum ≈ 4000). The increase of inference time for more layers is explained by the larger amount of parameters within the models, which is not the case for the previously benchmarked MLP. The quantization bit width has a negative impact on the execution time, which is most pronounced on larger models with more layers. The larger it is, the longer the prediction time. The accuracy degradation, conversely, decreases compared to the not quantized plaintext evaluations with larger bit width. At around five-bit quantization, the accuracy degradation is small enough to get very similar results. This result aligns with other research papers that concluded 5-bit quantization can achieve good accuracy results

and even smaller ones are possible [50–52]. We further tested the R^2 values using different amounts of training. We found that with longer training even smaller quantization bit widths could achieve good R^2 values. Similar to the linear regression benchmarks, we have calculated MAE and MaxAE values. Again, these measurements decrease with large bit width quantizations. At 3 bit they range between ≈ 11 –47 and 34–185 and at 6 bit between ≈ 1 –10 and 4–33. So a gradual decline can be seen, ultimately making the quantized, encrypted model functionally equivalent to the plaintext, not quantized one. Furthermore, it has to be noted that models with more layers consist of more parameters and therefore the parameter amount has a negative impact on the accuracy. In our examples quantization for inputs and weights was set to be equal. Additional studies might explore to differ these two.

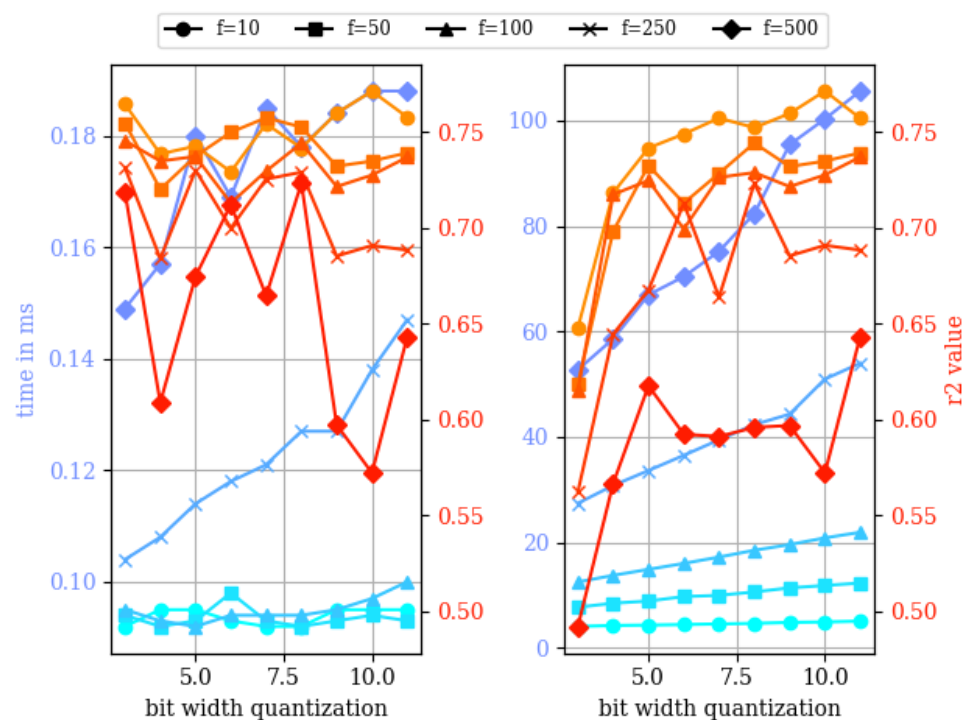


Figure 10. Inference times (displayed in cyan) and R^2 score (displayed in orange) of linear regression with different feature amounts depending on the quantization bit width. Plaintext inference is on the left and ciphertext evaluation on the right.

Security. In real-world applications the achieved security is another key consideration, as different use cases impose different protection requirements. In our tests a security of 128 bits was achieved. Parameter selection was done automatically by the compiler of either Concrete-ML or Concrete, with no further added compiler flags apart from bit quantizations. Varying the security level and other parameters could result in faster or slower executions. The security is reliant on different parameters of the LWE system. Specifically, the number of equations $N; N \in \mathbb{Z}$ and their dimension $n; n \in \mathbb{Z}$. Changing these results in smaller or larger keys and ciphertexts, as well as lower or higher security. Generally, the larger the parameters are, the more storage space and calculation time are needed. We performed a basic benchmark to test the evaluation time increase when changing the security level from medium, which is roughly 100-bit security, to 128-bit. These were done using already implemented benchmarks of the OpenFHE library for gate evaluation. The results show that computation time is roughly 30% slower for 128-bit security. It is furthermore possible to reduce evaluation times by fine-tuning other parameters, such as bootstrapping algorithms, the level after how many multiplications and additions bootstrapping should be applied and the integer modulus. In [53] the authors achieved up to a 39% speed-up by tweaking

these. Additionally it is possible to reduce bootstrapping accuracy in Concrete-ML, which can be specified with a compilation flag, to reduce runtimes further [54].

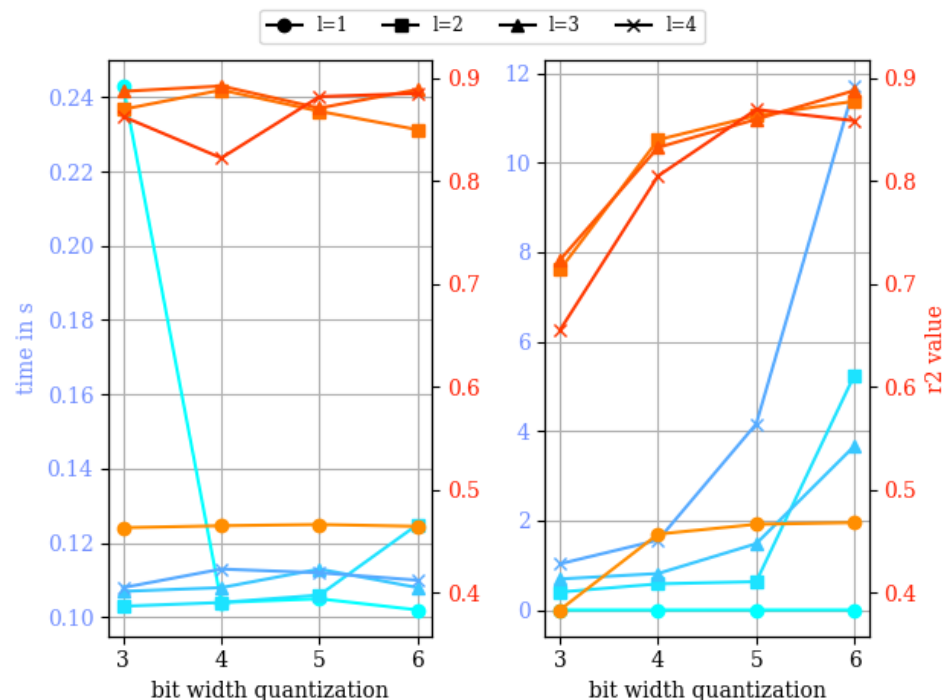


Figure 11. Inference times (displayed in cyan) and R^2 score (displayed in orange) of fitted MLP models of different layer amounts depending on the quantization bit width. Plaintext inference is on the **left** and ciphertext evaluation on the **right**.

Power Usage. We additionally evaluated our benchmarks with respect to energy consumption. Prior to all measurements, an idle system power draw of approximately 20 W was observed. For simple regression models, such as linear regression, no substantial difference between plaintext and ciphertext inference power draw could be observed. In both cases, the measured power consumption ranged between 60 and 70 W, corresponding to an effective workload-related consumption of approximately 40 to 50 W after subtracting the idle power usage.

For more complex architectures, such as dense neural network models, a clear distinction became apparent. Plaintext inference remained at roughly 40 W, whereas ciphertext inference resulted in an average power draw of approximately 160 W, or about 140 W after accounting for idle consumption.

Consequently, encrypted inference for larger models exhibits a significantly higher energy demand compared to their plaintext counterparts. Even for simpler regression models, where the instantaneous power draw is comparatively similar, encrypted inference still consumes substantially more energy due to its considerably longer execution time. Using the following energy consumption formula, we can calculate the difference: $E = P \cdot t$. A sample configuration from the linear regression benchmark (100 features, 8-bit quantization, taken from Table 5) yields an energy consumption of $E = 45 \text{ W} \cdot 1.4 \cdot 10^{-4} \text{ s} = 0.0063 \text{ J}$ for plaintext and $E = 45 \text{ W} \cdot 0.0015 \text{ s} = 0.0675 \text{ J}$ for ciphertext evaluation. Thus, encrypted inference requires approximately two orders of magnitude more energy, with this disparity becoming even more pronounced for larger and more complex models. These power usage differences need to be taken into account when evaluating the practical feasibility and scalability of encrypted machine learning deployments.

5. Discussion

5.1. Summary Recommendations

Both secure computation technologies present distinct advantages and limitations. SPU demonstrates strong performance with larger architectures; dense models, CNNs, and LSTMs are handled efficiently, making them viable for real-world applications. However, communication overhead significantly impacts performance, leading us to recommend the use of low-latency networks or high-speed channels. Regression models and basic calculations, such as distance metrics, are exceptionally fast in SPU, often requiring less than 100 ms. While SPU is versatile across use cases, the requirement for multiple servers makes the initial setup complex.

In contrast, Concrete-ML offers a much simpler deployment process. It excels in regression tasks. Through the use of hybrid models, and provided that the trade-off between performance and model IP exposure discussed earlier is acceptable, dense model inference can be accelerated to meet or exceed SPU speeds. The right choice for quantization bit width can offer further acceleration while achieving similar results. We saw that regressions achieve similar R^2 scores for small bit quantizations of 6 to 7 bits. MLP models already achieve an alignment to the plaintext R^2 scores at 5- to 6-bit quantization. However, even smaller bit widths are possible with enough training in some use cases. Yet completely encrypted training is only possible in very restricted and small use cases, because it is still too inefficient. Further performance improvements could be realized by changing LWE parameters, such as the security level, or reducing bootstrapping precision in TFHE. In conclusion, while deployment remains simple, achieving optimized and accurate computation requires careful tuning of several different parameters. Using only general-purpose CPUs for inference, we can see that Concrete-ML is highly inefficient for larger models like CNNs and does not support more complex model structures, such as LSTMs or Transformers. Furthermore, highly arithmetic calculations, such as distance measures, are equally inefficient in Concrete-ML. Here an exploration of other FHE schemes might achieve better efficiency, as leveled schemes are generally used for such computation, in contrast to bootstrapped schemes like TFHE.

Consequently, for computations using CPUs, we recommend Concrete-ML for traditional machine learning and simple dense models, especially in combination with hybrid models, due to its ease of setup and superior speed in those contexts. SPU remains the preferred choice for more complex, large-scale models and distance measures.

5.2. Combining Both Worlds

As we have seen FHE and MPC both come with certain advantages and disadvantages. In order to mitigate those weaknesses, one might consider combining both worlds into one technology and achieve even higher security and efficiency. There is already extensive research covering the use of FHE for multiple parties. In threshold FHE, parties create a single public key and a secret key that is split between them. To decrypt a ciphertext, all or a subset of them need to come together. Refs. [55,56] are examples of related studies. Both however rely on a trusted setup with a fixed number of clients and scale pretty badly with the number of parties. In [57] or [58] the authors propose a scheme that allows for a changing number of participating parties and was able to significantly improve the efficiency of the protocol.

Another idea, called multi-key FHE (MKFHE), is to let multiple parties with their own key pairs come together and combine them for MPC computation. It was first proposed here [59] for an on-the-fly selection of multiple clients by a cloud to share their data and let it compute arbitrary functions on it. Afterwards all participants need to combine their keys to correctly decrypt the result. This, having one cloud and multiple parties, is the most

common use case for MKFHE. Over the years this first proposal, based on NTRU, has been found to be insecure [60], but new ideas [61] and improvements [62] have been publicized, making it more efficient and secure.

Multi-party FHE (MPFHE) shares the use case of MKFHE, but functions more like threshold FHE. MPFHE uses MPC to create a single key pair, where the secret key is split and shared among all parties. This key pair can then be used to encrypt and threshold decrypt data for the server. An advantage of this scheme is that we have single-key FHE opposed to multi-key FHE. However, this results in much faster noise growth, effectively reducing circuit depth and increasing bootstrapping usage. In [63] the authors achieve a 2.46–71.16× reduction in total key material compared to the mentioned NTRU-based MKFHE scheme above [62]. Additionally, they compared bootstrapping times to other state-of-the-art schemes and achieved a time reduction of over 10 times.

Looking at the architectures above, we can see that MPFHE and MKFHE still work with only one server that calculates everything. The multiple parties are just there to assemble the data, encrypt and decrypt it again. Additionally, threshold FHE on its own defines only the encryption and decryption protocol, leaving out on how multiple parties can calculate functions together. So all the schemes above do not completely combine FHE and MPC, but rather give ideas how multiple parties can be involved in normal FHE. An example of a full combination of MPC and FHE is given here [64]. In this paper the authors present a full-fledged FHE-based MPC protocol for a variable number of input, output and computation parties. This architecture implements four key points. First, it implements a threshold FHE scheme. Second, it is fully secure, with security proofs given in the UC framework, and guarantees correct outputs for honest parties. Third, it accounts for some output parties being public and some private through the use of a Declassify operation. Additionally, the FHE computation model supports FHE-PRF evaluation, and lastly, it implements transciphering in order to enable resource-bound clients be able to participate too.

6. Conclusions

While FHE and MPC are now mature and well-researched technologies, they are frequently studied in isolation. By bringing them together for a direct comparison, we provide a cohesive perspective that clarifies the nuances of each approach, ultimately fostering broader adoption and further research. Our analysis highlights specific advantages and disadvantages that are essential for practitioners when selecting the appropriate technology for a given application. Specifically, we demonstrated that while both frameworks are theoretically capable of any computation, they involve significant tradeoffs—particularly in use cases like CNNs, where choosing one over the other drastically affects performance. However, our comparisons do not exhaust the vast range of potential applications for MPC and FHE, indicating that further research is necessary to fully map their capabilities.

Future Work

FHE and MPC are rapidly evolving fields with new research emerging annually. Consequently, the impacts of novel protocols and advancements warrant further study. To date, our benchmarks have relied exclusively on general-purpose CPUs. CPUs are generally perceived as a bad fit for FHE evaluation, as parallelization is limited and ciphertext sizes exceed CPU speed, like storage space and therefore extensive communication between RAM and CPU is required, slowing down evaluation speed. Consequently, other hardware methods like GPUs or FHE customized FPGAs and ASICs have been explored [65]. These could achieve significant speedups, so investigating how they influence execution times would be highly valuable. Generalizing CPU performance and behaviour to those

accelerators is not possible, as it is dependent on too many variables, such as hardware, acceleration method, implementation and more, that are hard to scope. Therefore, further benchmarks are needed. Similarly, exploring whether such hardware acceleration is viable for MPC is a crucial next step. Also due to the nature of MPC to properly measure power consumption and memory usage, a realistic setup would be required with multiple physical computers, which is also an interesting area for future study. Future research could also focus on developing machine learning models specifically architected for encrypted domains, such as those that minimize non-linear functions or replace them with FHE-friendly approximations like low-degree polynomials.

Author Contributions: Conceptualization, M.T., A.S. and T.L.; methodology, M.T., A.S. and T.L.; software, M.T. and A.S.; benchmarking, M.T. and A.S.; investigation, M.T., A.S. and T.L.; resources, T.L.; writing—original draft preparation, M.T. and A.S.; writing—review and editing, M.T., A.S. and T.L.; visualization, M.T. and A.S.; supervision, T.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was in part funded by the European Union’s Horizon Europe research and innovation programme under Grant Agreement n° 101168311 (LICORICE) and the Grant Agreement No. 101225635 (OnMoveID). Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union or granting authority. Neither the European Union nor the granting authority can be held responsible for them.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Our Code used for benchmarking and selected system, as well as compilation artifacts can be accessed in the git repo <https://github.com/ait-crypto/fhe-vs-mpc-4-ml> (accessed on 16 June 2026).

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. Detailed Results from Concrete-ML

Table A1. The complete collected data of the random forest regression benchmarks. Time was measured in seconds. The error is the calculated mean of all differences between plain and ciphertext results.

Features	Estimators	t_{Plain}	σ_{Plain}	t_{Concr}	σ_{Concr}
10	10	0.00074	2×10^{-5}	1.45059	0.00824
10	50	0.0008	5×10^{-5}	6.67242	0.15194
10	100	0.00085	2×10^{-5}	11.29582	0.17248
10	250	0.00099	2×10^{-5}	25.75356	0.25319
10	500	0.00116	2×10^{-5}	47.25149	0.38816
50	10	0.00118	2×10^{-5}	1.56283	0.10903
50	50	0.00131	1×10^{-5}	6.36573	0.03045
50	100	0.0014	4×10^{-5}	13.2169	0.02228
50	250	0.00161	3×10^{-5}	28.62728	0.2138
50	500	0.00205	2×10^{-5}	63.18776	0.26511
100	10	0.0017	2×10^{-5}	1.54024	0.00449
100	50	0.00184	2×10^{-5}	6.38462	0.30031
100	100	0.00191	0.00021	12.76922	0.1031
100	250	0.00312	5×10^{-5}	31.98638	0.6916
100	500	0.00303	8×10^{-5}	56.06664	1.08039
250	10	0.00317	8×10^{-5}	2.00454	0.0246
250	50	0.00519	0.00545	7.69274	0.40583
250	100	0.00383	0.00011	11.75653	0.60092
250	250	0.00486	5×10^{-5}	31.42815	0.2674
250	500	0.00641	0.00011	66.77785	1.38189

Table A1. *Cont.*

Features	Estimators	t_{Plain}	σ_{Plain}	t_{Concr}	σ_{Concr}
500	10	0.00563	4×10^{-5}	1.48872	0.01147
500	50	0.00615	8×10^{-5}	6.69472	0.18958
500	100	0.00689	0.00012	13.09014	0.12372
500	250	0.00867	7×10^{-5}	34.11808	1.99469
500	500	0.01165	0.00013	60.57889	0.23341

Table A2. The complete collected data of the MLP benchmarks for Concrete-ML. Time was measured in seconds. The error is the mean of all calculated mean squared errors.

Parameters	Layers	t_{Plain}	σ_{Plain}	t_{Concr}	σ_{Concr}	MSE
1000	1	0.0012	0.0001	3.0218	0.0772	0.0022
1000	2	0.0019	0.0003	3.2905	0.0774	0.0065
1000	3	0.0025	0.0004	4.1029	0.0709	0.001
1000	4	0.0033	0.0005	4.2804	0.0561	0.0002
5000	1	0.0011	0.0001	13.0818	0.155	0.0003
5000	2	0.0018	0.0005	8.9292	0.1911	0.0034
5000	3	0.0027	0.0011	8.0452	0.193	0.0021
5000	4	0.0032	0.0005	8.977	0.174	0.0002
10,000	1	0.0011	0.0001	26.706	0.2035	0.0028
10,000	2	0.0018	0.0003	14.7837	0.2391	0.0004
10,000	3	0.0026	0.0008	13.0979	0.2284	0.0002
10,000	4	0.0034	0.0013	13.8455	0.2683	0.0001

Table A3. The complete collected data of the hybrid MLP benchmarks for Concrete-ML. Time was measured in seconds. The error is the mean of all calculated mean squared errors.

Parameters	Layers	t_{Concr}	σ_{Concr}	MSE
1000	1	0.022	0.0015	0.2203
1000	2	0.0256	0.0023	0.0349
1000	3	0.0353	0.0031	0.0316
1000	4	0.0458	0.0019	0.0335
5000	1	0.078	0.0097	0.2195
5000	2	0.0408	0.002	0.044
5000	3	0.0557	0.0028	0.0095
5000	4	0.0691	0.0034	0.0062
10,000	1	0.1684	0.0102	0.2034
10,000	2	0.0596	0.0025	0.0401
10,000	3	0.068	0.0025	0.0129
10,000	4	0.0832	0.0025	0.0037

Table A4. The complete collected data of the CNN benchmarks for Concrete-ML. Time was measured in seconds. The error is the mean of all calculated mean squared errors. Due to the long calculation times only 3 repetitions were done for the 10,000 parameter configuration on 32×32 .

Parameters	p-Size	t_{Plain}	σ_{Plain}	t_{Concr}	σ_{Concr}	MSE
1000	8	0.0036	0.0003	70.2287	0.8443	0.0109
5000	8	0.0037	0.0003	163.8177	2.1757	0.0146
10,000	8	0.0038	0.0003	253.1516	2.382	0.0039
1000	16	0.0038	0.0004	519.2235	3.7639	0.0091
5000	16	0.004	0.0004	1516.1285	4.7551	0.0056
10,000	16	0.0041	0.0004	2311.901	35.8898	0.0086
1000	32	0.0041	0.0004	3000.1875	13.3217	0.0089
5000	32	0.0047	0.0006	8544.2489	32.9393	0.0042
10,000	32	0.0053	0.0007	10,194.2407	0	0.0129

Appendix B. Detailed Results from SPU

Table A5. The complete collected data of the MLP benchmarks for SPU. Time was measured in seconds. All encrypted results matched the plain results.

Parameters	Layers	t_{plain}	σ_{plain}	t_{SPU}	σ_{SPU}
1000	1	0.00876	0.00056	0.07195	0.00891
1000	2	0.01499	0.00115	0.09152	0.01095
1000	3	0.02019	0.00149	0.11142	0.01239
1000	4	0.02461	0.00184	0.13087	0.01311
5000	1	0.00889	0.00051	0.07739	0.00878
5000	2	0.01519	0.00091	0.09488	0.01033
5000	3	0.02023	0.00149	0.11408	0.01229
5000	4	0.02457	0.00189	0.13292	0.01334
10,000	1	0.00902	0.0004	0.08289	0.00853
10,000	2	0.01531	0.00089	0.09817	0.01057
10,000	3	0.02039	0.00151	0.11722	0.01219
10,000	4	0.02462	0.00206	0.13549	0.01366

Table A6. The complete collected data of the LSTM benchmarks for SPU. Time was measured in seconds. All encrypted results matched the plain results.

Parameters	Layers	t_{plain}	σ_{plain}	t_{SPU}	σ_{SPU}
1000	3	0.01678	0.00498	5.44266	0.07837
1000	6	0.02471	0.00759	8.82986	0.16202
1000	9	0.03668	0.01080	11.4944	0.16619
5000	3	0.01753	0.00535	9.25728	0.07078
5000	6	0.02748	0.00716	13.1413	0.08262
5000	9	0.03447	0.01061	16.3655	0.13836
10,000	3	0.01587	0.00430	11.54590	0.10230
10,000	6	0.02845	0.00852	16.97504	0.09231
10,000	9	0.03769	0.01079	21.10129	0.13237

Table A7. The complete collected data of the CNN benchmarks for SPU, done on $3 \times 112 \times 112$ pictures. Time was measured in seconds. All encrypted results matched the plain results.

Parameters	Layers	t_{plain}	σ_{plain}	t_{SPU}	σ_{SPU}
500	1	0.13944	0.01491	0.37556	0.04554
500	3	0.37669	0.06314	0.98507	0.11904
500	6	0.7558	0.16406	1.8939	0.24616
500	10	1.30493	0.22597	2.97009	0.45787
1000	1	0.15701	0.0144	0.38809	0.10084
1000	3	0.40963	0.12758	1.00046	0.15523
1000	6	0.78839	0.20316	1.93482	0.21767
1000	10	1.31471	0.23703	3.14805	0.44031
5000	1	0.16933	0.09119	0.3836	0.04637
5000	3	0.43357	0.16142	1.00425	0.12395
5000	6	0.8329	0.17755	1.95504	0.25796
5000	10	1.29806	0.2231	3.2461	0.40449

Table A8. The complete collected data of the CNN benchmarks for SPU with 10 ms communication delay done on $3 \times 112 \times 112$ pictures. Time was measured in seconds. All encrypted results matched the plain results.

Parameters	Layers	t_{plain}	σ_{plain}	t_{SPU}	σ_{SPU}
1000	3	0.015	0.00126	8.18431	0.05045
1000	6	0.02697	0.00423	13.68423	0.10527
1000	9	0.03619	0.00275	18.34431	0.11529
5000	3	0.01766	0.005	12.41237	0.06434

Table A8. *Cont.*

Parameters	Layers	t_{Plain}	σ_{Plain}	t_{SPU}	σ_{SPU}
5000	6	0.0309	0.00834	18.60298	0.16614
5000	9	0.03582	0.00858	24.04591	0.18056
10,000	3	0.0158	0.00511	15.65256	0.06923
10,000	6	0.02777	0.00646	23.25883	0.09898
10,000	9	0.0446	0.0124	29.77412	0.08742

Table A9. The complete collected data of the CNN benchmarks for SPU with 20 ms communication delay, done on $3 \times 112 \times 112$ pictures. Time was measured in seconds. All encrypted results matched the plain results.

Parameters	Layers	t_{Plain}	σ_{Plain}	t_{SPU}	σ_{SPU}
1000	3	0.01708	0.00557	10.4285	0.07826
1000	6	0.02498	0.0009	17.86829	0.11977
1000	9	0.03827	0.00642	24.48244	0.14138
5000	3	0.01783	0.00537	15.19634	0.05627
5000	6	0.0351	0.00913	23.20118	0.06653
5000	9	0.03257	0.00203	30.4767	0.15704
10,000	3	0.0197	0.00563	19.29566	0.1057
10,000	6	0.02648	0.00413	28.63264	0.15038
10,000	9	0.04792	0.00529	36.87332	0.19553

Table A10. The complete collected data of the CNN benchmarks for SPU with 1% packet loss, done on $3 \times 112 \times 112$ pictures. Time was measured in seconds. All encrypted results matched the plain results.

Parameters	Layers	t_{Plain}	σ_{Plain}	t_{SPU}	σ_{SPU}
1000	3	0.01527	0.00321	5.80257	0.20026
1000	6	0.02497	0.00687	9.33501	0.29233
1000	9	0.03185	0.00601	12.06953	0.35974
5000	3	0.01656	0.00502	9.84976	0.2557
5000	6	0.02916	0.00967	13.98081	0.3547
5000	9	0.03621	0.01229	17.5132	0.41187
10,000	3	0.01694	0.00543	12.24659	0.21321
10,000	6	0.02886	0.00918	17.98501	0.32278
10,000	9	0.03745	0.01159	22.359	0.29816

Table A11. The complete collected data of the CNN benchmarks for SPU with 2.5% packet loss, done on $3 \times 112 \times 112$ pictures. Time was measured in seconds. All encrypted results matched the plain results.

Parameters	Layers	t_{Plain}	σ_{Plain}	t_{SPU}	σ_{SPU}
1000	3	0.01754	0.005	6.77659	0.43985
1000	6	0.02643	0.00773	10.42625	0.46351
1000	9	0.03788	0.01126	13.7213	0.59121
5000	3	0.0178	0.00524	11.01357	0.5559
5000	6	0.02955	0.00869	15.7136	0.7349
5000	9	0.03604	0.01239	19.95943	0.69378
10,000	3	0.0167	0.0049	13.99751	0.48123
10,000	6	0.02679	0.00814	20.28912	0.69958
10,000	9	0.04377	0.01533	25.65867	0.77047

Table A12. The complete collected data of the CNN benchmarks for SPU with 5% packet loss, done on $3 \times 112 \times 112$ pictures. Time was measured in seconds. All encrypted results matched the plain results.

Parameters	Layers	t_{Plain}	σ_{Plain}	t_{SPU}	σ_{SPU}
1000	3	0.01628	0.00393	9.69478	0.95705
1000	6	0.02729	0.00753	15.18811	1.22776
1000	9	0.03456	0.00651	19.30261	1.42329
5000	3	0.01798	0.00498	15.95897	1.00135
5000	6	0.03009	0.00706	23.00529	1.57291
5000	9	0.0398	0.01345	29.39708	2.11431
10,000	3	0.01642	0.00472	21.51006	1.53781
10,000	6	0.02846	0.00819	29.91287	1.58287
10,000	9	0.0462	0.01533	37.05196	2.03245

References

1. Taubert, M.; Skuta, A.; Lorünser, T. A Pragmatic Comparison of Cryptographic Computation Technologies for Machine Learning. In *Proceedings of the 12th International Conference on Information Systems Security and Privacy—Volume 2: ICISSP; INSTICC, SciTePress: Marbella, Spain, 2026*; pp. 422–430. [\[CrossRef\]](#)
2. Keller, M. MP-SPDZ: A Versatile Framework for Multi-Party Computation. In *Proceedings of the SIGSAC Conference on Computer and Communications Security, CCS '20, New York, NY, USA, 9–13 November 2020*; pp. 1575–1590. [\[CrossRef\]](#)
3. Dong, Y.; Lu, W.j.; Zheng, Y.; Wu, H.; Zhao, D.; Tan, J.; Huang, Z.; Hong, C.; Wei, T.; Chen, W. PUMA: Secure Inference of LLaMA-7B in Five Minutes. *arXiv* **2023**, arXiv:2307.12533. [\[CrossRef\]](#)
4. Akimoto, Y.; Fukuchi, K.; Akimoto, Y.; Sakuma, J. Privformer: Privacy-preserving Transformer with MPC. In *Proceedings of the 2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*; IEEE: Delft, The Netherlands, 2023; pp. 392–410. [\[CrossRef\]](#)
5. Luo, J.; Chen, G.; Zhang, Y.; Liu, S.; Wang, H.; Yu, Y.; Zhou, X.; Qi, Y.; Xu, Z. Centaur: Bridging the Impossible Trinity of Privacy, Efficiency, and Performance in Privacy-Preserving Transformer Inference. *arXiv* **2024**, arXiv:2412.10652. [\[CrossRef\]](#)
6. Jiang, L.; Ju, L. FHEBench: Benchmarking Fully Homomorphic Encryption Schemes. *arXiv* **2022**, arXiv:2203.00728. [\[CrossRef\]](#)
7. Ahmad, J.; Ghaleb, B.; Jan S.U.; Buchanan, W.J. Cross-Platform Benchmarking of the FHE Libraries: Novel Insights into SEAL and OpenFHE. *Cryptol. ePrint Arch.* **2025**, Paper 2025/473. Available online: <https://eprint.iacr.org/2025/473> (accessed on 16 June 2026).
8. Badawi, A.A.; Polyakov, Y. Demystifying Bootstrapping in Fully Homomorphic Encryption. *Cryptol. ePrint Arch.* **2023**, Paper 2023/149. Available online: <https://eprint.iacr.org/2023/149> (accessed on 16 June 2026).
9. Reddy, B.P.K.; Goyal, S.; Meel, R.; Chatterjee, A. FHEMaLe: Framework for Homomorphic Encrypted Machine Learning. *Cryptol. ePrint Arch.* **2025**, Paper 2025/1684. Available online: <https://eprint.iacr.org/2025/1684> (accessed on 16 June 2026).
10. Ebel, A.; Garimella, K.; Reagen, B. Orion: A Fully Homomorphic Encryption Framework for Deep Learning. *arXiv* **2025**, arXiv:2311.03470. [\[CrossRef\]](#)
11. Noah, A.; Okunola, A. A Comparative Analysis of Homomorphic Encryption and Secure Multi-Party Computation for Preserving Data Privacy in Cloud-Based Financial Services. 2025. Available online: https://www.researchgate.net/publication/392165415_A_Comparative_Analysis_of_Homomorphic_Encryption_and_Secure_Multi-Party_Computation_for_Preserving_Data_Privacy_in_Cloud-Based_Financial_Services (accessed on 16 June 2026).
12. Chadhar, M.; Oladele, S.; Rouf, A. A Comparative Analysis of Homomorphic Encryption and Secure Multi-Party Computation for Preserving Data Confidentiality in Cloud-Based BI Analytics. 2025. Available online: https://www.researchgate.net/publication/393775133_A_Comparative_Analysis_of_Homomorphic_Encryption_and_Secure_Multi-Party_Computation_for_Preserving_Data_Confidentiality_in_Cloud-Based_BI_Analytics (accessed on 16 June 2026).
13. Cheerla, K.; Othmane, L.B.; Morozov, K. Comparison of Fully Homomorphic Encryption and Garbled Circuit Techniques in Privacy-Preserving Machine Learning Inference. *arXiv* **2025**, arXiv:2510.07457.
14. Evans, D.; Kolesnikov, V.; Rosulek, M. A pragmatic introduction to secure multi-party computation. *Found. Trends® Priv. Secur.* **2018**, 2, 70–246. [\[CrossRef\]](#)
15. Knott, B.; Venkataraman, S.; Hannun, A.; Sengupta, S.; Ibrahim, M.; Maaten, L.v.d. CrypTen: Secure Multi-Party Computation Meets Machine Learning. *arXiv* **2022**, arXiv:2109.00984. [\[CrossRef\]](#)
16. Balbás, D. The Hardness of LWE and Ring-LWE: A Survey. *Cryptol. ePrint Arch.* **2021**, Paper 2021/1358. Available online: <https://eprint.iacr.org/2021/1358> (accessed on 16 June 2026).
17. Albrecht, M.R.; Player, R.; Scott, S. On the concrete hardness of Learning with Errors. *Cryptol. ePrint Arch.* **2015**, Paper 2015/046. Available online: <https://eprint.iacr.org/2015/046> (accessed on 16 June 2026).

18. Albrecht, M.; Chase, M.; Chen, H.; Ding, J.; Goldwasser, S.; Gorbunov, S.; Halevi, S.; Hoffstein, J.; Laine, K.; Lauter, K.; et al. *Homomorphic Encryption Security Standard*; Technical Report; HomomorphicEncryption.org: Toronto, ON, Canada, 2018.
19. Fauzi, P.; Hovd, M.N.; Raddum, H. On the IND-CCA1 Security of FHE Schemes. *Cryptography* **2022**, *6*, 13. [\[CrossRef\]](#)
20. Li, B.; Micciancio, D.; Schultz, M.; Sorrell, J. Securing Approximate Homomorphic Encryption Using Differential Privacy. *Cryptol. ePrint Arch.* **2022**, Paper 2022/816. Available online: <https://eprint.iacr.org/2022/816> (accessed on 16 June 2026).
21. Canetti, R.; Raghuraman, S.; Richelson, S.; Vaikuntanathan, V. Chosen-Ciphertext Secure Fully Homomorphic Encryption. In Proceedings of the International Conference on Theory and Practice of Public Key Cryptography, Amsterdam, The Netherlands, 28–31 March 2017.
22. Natarajan, D.; Loveless, A.; Dai, W.; Dreslinski, R. CHEX-MIX: Combining Homomorphic Encryption with Trusted Execution Environments for Two-party Oblivious Inference in the Cloud. *Cryptol. ePrint Arch.* **2021**, Paper 2021/1603. Available online: <https://eprint.iacr.org/2021/1603> (accessed on 16 June 2026).
23. Lou, Q.; Santriaci, M.; Yudha, A.W.B.; Xue, J.; Solihin, Y. vFHE: Verifiable Fully Homomorphic Encryption with Blind Hash. *arXiv* **2023**, arXiv:2303.08886.
24. Fiore, D.; Gennaro, R.; Pastro, V. Efficiently Verifiable Computation on Encrypted Data. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS '14, Scottsdale, AZ, USA, 3–7 November 2014; pp. 844–855. [\[CrossRef\]](#)
25. Viand, A.; Knabenhans, C.; Hithnawi, A. Verifiable Fully Homomorphic Encryption. *arXiv* **2023**, arXiv:2301.07041.
26. Brakerski, Z.; Gentry, C.; Vaikuntanathan, V. Fully Homomorphic Encryption without Bootstrapping. *Cryptol. ePrint Arch.* **2011**, Paper 2011/277. Available online: <https://eprint.iacr.org/2011/277> (accessed on 16 June 2026).
27. Fan, J.; Vercauteren, F. Somewhat Practical Fully Homomorphic Encryption. *Cryptol. ePrint Arch.* **2012**, Paper 2012/144. Available online: <https://eprint.iacr.org/2012/144> (accessed on 16 June 2026).
28. Cheon, J.H.; Kim, A.; Kim, M.; Song, Y. Homomorphic Encryption for Arithmetic of Approximate Numbers. *Cryptol. ePrint Arch.* **2016**, Paper 2016/421. Available online: <https://eprint.iacr.org/2016/421> (accessed on 16 June 2026).
29. Chillotti, I.; Gama, N.; Georgieva, M.; Izabachène, M. TFHE: Fast Fully Homomorphic Encryption over the Torus. *Cryptol. ePrint Arch.* **2018**, Paper 2018/421. Available online: <https://eprint.iacr.org/2018/421> (accessed on 16 June 2026).
30. Zhou, T.; Yang, X.; Liu, L.; Zhang, W.; Li, N. Faster Bootstrapping with Multiple Addends. *IEEE Access* **2018**, *6*, 49868–49876. [\[CrossRef\]](#)
31. Manulis, M.; Nguyen, J. Fully Homomorphic Encryption beyond IND-CCA1 Security: Integrity through Verifiability. *Cryptol. ePrint Arch.* **2024**, Paper 2024/202. Available online: <https://eprint.iacr.org/2024/202> (accessed on 16 June 2026).
32. Badawi, A.A.; Alexandru, A.; Bates, J.; Bergamaschi, F.; Cousins, D.B.; Erabelli, S.; Genise, N.; Halevi, S.; Hunt, H.; Kim, A.; et al. OpenFHE: Open-Source Fully Homomorphic Encryption Library. *Cryptol. ePrint Arch.* **2022**, Paper 2022/915. Available online: <https://eprint.iacr.org/2022/915> (accessed on 23 June 2023).
33. Ali, A.; Choi, J.; Gipson, B.; Gorantala, S.; Kun, J.; Legiest, W.; Lim, L.; Viand, A.; Demissie, M.Z.; Zheng, H. HEIR: A Universal Compiler for Homomorphic Encryption. *arXiv* **2025**, arXiv:2508.11095.
34. Microsoft SEAL (Release 4.1); Microsoft Research: Redmond, WA, USA, 2023. Available online: <https://github.com/Microsoft/SEAL> (accessed on 16 June 2026).
35. Zama. TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics over Encrypted Data. 2022. Available online: <https://github.com/zama-ai/tfhe-rs> (accessed on 16 June 2026).
36. Zama. Concrete: TFHE Compiler That Converts Python Programs into FHE Equivalent. 2022. Available online: <https://github.com/zama-ai/concrete> (accessed on 16 June 2026).
37. Zama. Concrete ML: A Privacy-Preserving Machine Learning Library Using Fully Homomorphic Encryption for Data Scientists. 2022. Available online: <https://github.com/zama-ai/concrete-ml> (accessed on 16 June 2026).
38. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *arXiv* **2019**, arXiv:1912.01703.
39. Franco, G.; Pappalardo, A.; Fraser, N.J. Xilinx/brevitas. *Zenodo* **2025**. [\[CrossRef\]](#)
40. Ma, J.; Zheng, Y.; Feng, J.; Zhao, D.; Wu, H.; Fang, W.; Tan, J.; Yu, C.; Zhang, B.; Wang, L. SecretFlow-SPU: A Performant and User-Friendly Framework for Privacy-Preserving Machine Learning. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*; USENIX: Berkeley, CA, USA, 2023; pp. 17–33.
41. Danial, A. cloc: V1.98. *Zenodo* **2023**. [\[CrossRef\]](#)
42. King, I.C. Powerstat. Available online: <https://github.com/ColinIanKing/powerstat> (accessed on 13 May 2026).
43. Mohassel, P.; Rindal, P. ABY³: A Mixed Protocol Framework for Machine Learning. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS'18), Toronto, ON, Canada, 15–19 October 2018.
44. H2O.ai. Random Forest—H2O Wiki. Available online: <https://h2o.ai/wiki/random-forest/> (accessed on 30 October 2025).
45. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. *arXiv* **2015**, arXiv:1512.03385.

46. Chen, S.; Liu, Y.; Gao, X.; Han, Z. MobileFaceNets: Efficient CNNs for Accurate Real-Time Face Verification on Mobile Devices. *arXiv* **2018**, arXiv:1804.07573.
47. Alansari, M.; Hay, O.A.; Javed, S.; Shoufan, A.; Zweiri, Y.; Werghi, N. GhostFaceNets: Lightweight Face Recognition Model from Cheap Operations. *IEEE Access* **2023**, *11*, 35429–35446. [[CrossRef](#)]
48. Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *arXiv* **2021**, arXiv:2010.11929.
49. Wu, H.; Xiao, B.; Codella, N.; Liu, M.; Dai, X.; Yuan, L.; Zhang, L. CvT: Introducing Convolutions to Vision Transformers. *arXiv* **2021**, arXiv:2103.15808.
50. Kanda, R.; Onizawa, N.; Leonardon, M.; Gripon, V.; Hanyu, T. Design Environment of Quantization-Aware Edge AI Hardware for Few-Shot Learning. In *Proceedings of the 2024 IEEE 67th International Midwest Symposium on Circuits and Systems (MWSCAS)*; IEEE: Piscataway, NJ, USA, 2024; pp. 928–931. [[CrossRef](#)]
51. Zhou, A.; Yao, A.; Guo, Y.; Xu, L.; Chen, Y. Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights. *arXiv* **2017**, arXiv:1702.03044.
52. Ducasse, Q.; Cotret, P.; Lagadec, L.; Stewart, R. Benchmarking Quantized Neural Networks on FPGAs with FINN. *arXiv* **2021**, arXiv:2102.01341.
53. Klemsa, J. Hitchhiker’s Guide to a Practical Automated TFHE Parameter Setup for Custom Applications. *Cryptol. ePrint Arch.* **2022**, Paper 2022/1315. Available online: <https://eprint.iacr.org/2022/1315> (accessed on 16 June 2026).
54. Hong, D.; Kim, Y.S.; Lee, Y.; Seo, E. A New Fine Tuning Method for FHEW/TFHE Bootstrapping with IND-CPAD Security. *Cryptol. ePrint Arch.* **2024**, Paper 2024/1052. Available online: <https://eprint.iacr.org/2024/1052> (accessed on 16 June 2026).
55. Cheon, J.H.; Cho, W.; Kim, J. Improved Universal Thresholdizer from Iterative Shamir Secret Sharing. *Cryptol. ePrint Arch.* **2023**, Paper 2023/545. Available online: <https://eprint.iacr.org/2023/545> (accessed on 16 June 2026).
56. Boudgoust, K.; Scholl, P. Simple Threshold (Fully Homomorphic) Encryption from LWE with Polynomial Modulus. *Cryptol. ePrint Arch.* **2023**, Paper 2023/016. Available online: <https://eprint.iacr.org/2023/016> (accessed on 16 June 2026).
57. Brakerski, Z.; Friedman, O.; Marmor, A.; Mutzari, D.; Spiizer, Y.; Trieu, N. Threshold FHE with Efficient Asynchronous Decryption. *Cryptol. ePrint Arch.* **2025**, Paper 2025/712. Available online: <https://eprint.iacr.org/2025/712> (accessed on 16 June 2026).
58. Chang, Y.; Li, S. Arbitrary-Threshold Fully Homomorphic Encryption with Lower Complexity. *arXiv* **2025**, arXiv:2501.11235.
59. Lopez-Alt, A.; Tromer, E.; Vaikuntanathan, V. On-the-Fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption. *Cryptol. ePrint Arch.* **2013**, Paper 2013/094. Available online: <https://eprint.iacr.org/2013/094> (accessed on 16 June 2026).
60. Albrecht, M.; Bai, S.; Ducas, L. A subfield lattice attack on overstretched NTRU assumptions: Cryptanalysis of some FHE and Graded Encoding Schemes. *Cryptol. ePrint Arch.* **2016**, Paper 2016/127. Available online: <https://eprint.iacr.org/2016/127> (accessed on 16 June 2026).
61. Zhou, T.; Chen, L.; Che, X.; Liu, W.; Zhang, Z.; Yang, X. Multi-key Fully Homomorphic Encryption Scheme with Compact Ciphertexts. *Cryptol. ePrint Arch.* **2021**, Paper 2021/1131. Available online: <https://eprint.iacr.org/2021/1131> (accessed on 16 June 2026).
62. Park, J.; Leeuwen, B.V.; Zajonc, O. FINALLY: A Multi-Key FHE Scheme Based on NTRU and LWE. *Cryptol. ePrint Arch.* **2024**, Paper 2024/1505. Available online: <https://eprint.iacr.org/2024/1505> (accessed on 16 June 2026).
63. Jadoul, R.; van Leeuwen, B.; Zajonc, O. An Efficient Framework for Collusion Resistant Multiparty FHE. *Cryptol. ePrint Arch.* **2025**, Paper 2025/965. Available online: <https://eprint.iacr.org/2025/965> (accessed on 16 June 2026).
64. Smart, N.P. Practical and Efficient FHE-based MPC. *Cryptol. ePrint Arch.* **2023**, Paper 2023/981. Available online: <https://eprint.iacr.org/2023/981> (accessed on 16 June 2026).
65. Gong, Y.; Chang, X.; Mišić, J.; Mišić, V.B.; Wang, J.; Zhu, H. Practical Solutions in Fully Homomorphic Encryption—A Survey Analyzing Existing Acceleration Methods. *arXiv* **2023**, arXiv:2303.10877.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.