# White Paper: TruPS- Enhancing Trusted Transatlantic Data Processing and Storage with Fully Homomorphic Encryption

Kruno Miličević (Random Red), Davor Vinko (Random Red), Ivan Uglik (Random Red), Adrijan Đurin (Random Red), Madhurima Ray (Penn State University – Beaver Campus), Richard Lomotey (Penn State University – Beaver Campus)

*May, 2025*

*Version 1.1*

# Contents








Funded by the European Union. The project NGISARGASSO-2024-CALL3-4-TruPS „Enhancing Trusted Transatlantic Data Processing and Storage with Fully Homomorphic Encryption” has received funding within the framework of the NGI Sargasso project under grant agreement No 101092887.

# Abstract

In the era of cloud computing and third-party data analytics, ensuring data remains secure and private during processing is a critical challenge. Fully Homomorphic Encryption (FHE) offers a transformative solution by enabling computation on encrypted data, thus keeping data protected even while in use. This white paper provides a comprehensive overview of FHE security and an experimental performance assessment in the context of machine learning (ML). We compare several open-source FHE libraries and evaluate Concrete-ML, a privacy-preserving ML framework built on FHE, against traditional machine learning workflows. Different machine learning algorithms were implemented with and without FHE to measure the impact on training time, inference latency, and model accuracy. The results demonstrate that FHE can preserve model accuracy (usually within 1% of non-encrypted models) while incurring significant performance overheads - training and inference times were often orders of magnitude longer under encryption. We discuss current technical limitations (performance, usability, deployment challenges) that impede FHE's widespread adoption and examine how FHE aligns with evolving regulatory requirements for data privacy. Despite current limitations, FHE's inherent resilience against even quantum attacks positions it as a promising technology for secure data processing, including machine learning and AI. We conclude with a comparative analysis of FHE frameworks and outline future work needed to make FHE more practical for broad ML/AI deployment.

# Introduction

When choosing cloud platforms or external data processors, organizations must balance technical capabilities with compliance to data protection regulations. In recent years, numerous laws and regulations across jurisdictions have emerged to govern data storage and processing, especially for sensitive sectors like healthcare, finance, and government. For example, the United States relies on sector-specific laws (HIPAA for health data, FISMA for federal agencies, GLBA for finance, etc.), emphasizing encryption of sensitive data per NIST guidelines, while lacking a single comprehensive federal data law. The European Union enforces broad regulations like the General Data Protection Regulation (GDPR) and frameworks like eIDAS, which mandate strong protections for personal data, with strict requirements when data leaves its jurisdiction. Across these regimes, encryption is highlighted as a state-of-the-art measure to safeguard data.

Traditional encryption, however, only protects data at rest (storage) or in transit (communication). Once data is decrypted for processing or analysis - such as on a cloud server - the protective envelope is removed, potentially violating regulatory requirements and exposing data to unauthorized access. Certain regulatory interpretations (notably GDPR) increasingly expect encryption during third-party or cross-border processing unless access is provably constrained. This is where Fully Homomorphic Encryption (FHE) comes into play. FHE is a form of encryption that allows computations to be performed directly on ciphertexts, producing encrypted results that only the data owner can decrypt. In theory, this means data can remain encrypted throughout its entire lifecycle - during storage, transit, and *processing* - closing the last gap in data protection.

FHE permits seemingly impossible functionality: computing on data that one cannot read. The breakthrough came in 2009 when Craig Gentry described the first plausible FHE scheme using lattice-based cryptography. Gentry's approach involved a technique called bootstrapping (re-encrypting ciphertexts to reduce accumulated noise) to achieve unlimited computations on encrypted data. Early implementations were extremely slow - initially, a single bootstrapping operation took on the order of 30 minutes. Over the past decade, researchers have dramatically improved FHE's efficiency. Modern schemes can perform a bootstrap in the order of tens of milliseconds, thanks to advances in algorithms and optimized implementations.

Multiple generations of FHE schemes have evolved. Notably, there are two broad paradigms in contemporary FHE: one exemplified by TFHE (Torus FHE) which prioritizes fast bootstrapping (so every operation refreshes noise, enabling unlimited depth), and another exemplified by CKKS which uses approximate arithmetic and aims to minimize the need for bootstrapping by handling a fixed number of operations with tolerable noise growth. In short, TFHE takes the approach of performing a lightweight bootstrap after every operation, while CKKS (and similar "leveled" schemes like BGV/BFV) allow a predetermined circuit depth of operations without bootstrapping, as long as the noise budget is not exceeded. Each approach has trade-offs: TFHE can support arbitrary-length computations but each operation is slower, whereas leveled schemes can be faster per operation but require careful circuit planning or occasional expensive bootstraps for very deep computations.

Today, there is a growing ecosystem of FHE libraries and tools (discussed in a later section) that implement these schemes and make FHE more accessible. This paper focuses on applying FHE to machine learning tasks. We use Concrete-ML - an open-source framework built on Zama's Concrete library (an implementation of TFHE) - to evaluate how practical it is to train and run ML models on encrypted data. We present experimental results comparing standard ML models with their FHE-powered counterparts in terms of training time, inference time, and accuracy. We also outline the current technical limitations we encountered, ranging from performance bottlenecks to usability challenges. In addition, given the importance of data privacy compliance, we discuss how FHE can help satisfy regulatory requirements by protecting data in use. Finally, we provide a comparative analysis of available FHE solutions and highlight future research and development needed to fully realize secure and efficient machine learning on encrypted data.

# Regulatory Discussion

Data protection regulations worldwide increasingly call for strong technical safeguards for personal and sensitive data, especially when such data is processed by third parties or across borders. FHE presents a compelling tool to achieve compliance with many of these regulations by enabling data to remain encrypted during processing, effectively minimizing exposure of raw data.

Under GDPR in the EU, for instance, there is a principle of "data protection by design and by default," which encourages methods like encryption to be applied whenever possible to protect personal data. FHE could allow data controllers to use cloud processors or external analytics platforms without ever revealing personal data to those platforms, thus significantly reducing the risk of unauthorized exposure. This directly addresses GDPR's requirements for

confidentiality and integrity of data processing. Furthermore, GDPR imposes strict rules on international data transfer; using FHE, an EU entity could theoretically send encrypted data to a cloud in another jurisdiction, have it processed, and get back encrypted results, all without transferring any plaintext personal data. This could be a game-changer for compliance, effectively sidestepping data residency issues because the data "in transit" and "in use" remains unintelligible to any foreign entity.

In sector-specific regulations like HIPAA (healthcare in the U.S.), FHE could enable cloud-based medical data analysis or machine learning on patient data while keeping that data encrypted, thus maintaining compliance with the requirement to safeguard Protected Health Information (PHI). Since only the data owner (e.g., the hospital) holds the decryption key, any computations performed by a cloud service provider using Fully Homomorphic Encryption (FHE) would inherently prevent that provider from accessing PHI in plaintext. While HIPAA permits such processing under appropriate safeguards and business associate agreements, FHE provides a cryptographic guarantee that data remains unintelligible to third parties, thereby minimizing residual risk and strengthening compliance. Similarly, financial regulations (e.g., those by FINRA or the SEC) that require safeguarding customer data could benefit from FHE by enabling secure third-party analytics without exposing sensitive financial information.

It's important to note that while FHE can greatly enhance compliance, it is not explicitly mandated or referenced by most laws today. Rather, it serves as an *enabler* to meet requirements for encryption and minimizing data exposure. For example, U.S. federal guidelines (from NIST) and EU recommendations both stop short of requiring FHE (given its novelty) but do recommend encrypting data whenever feasible. By extending encryption to the processing phase, FHE provides a level of assurance beyond the current typical measures. In fact, FHE's protection could ease some requirements for other controls. If data is always encrypted when handled by a service provider, that provider might not need certain costly certifications or audits for handling sensitive data since they theoretically never "handle" it in an intelligible form. Regulators are beginning to take interest in advanced cryptographic techniques; for instance, the EU's ePrivacy Regulation drafts have considered homomorphic encryption as a promising technology for privacy-preserving data processing.

Another regulatory aspect is **post-quantum security**. Many traditional encryption schemes could be broken by future quantum computers, which has regulators and standards bodies urging a migration to quantum-resistant algorithms. FHE schemes like those based on lattice problems (RLWE) are believed to be quantum-resistant. Thus, adopting FHE not only protects data now but also is an investment in cryptographic resilience for the future. For organizations worried about "harvest-now, decrypt-later" attacks (where encrypted data might be stored by adversaries today to decrypt in the future when quantum attacks become feasible), using FHE (with its lattice-based encryption) for processing means that even intermediate computations remain safe against future quantum adversaries. In our experiments we used Concrete's TFHE, which is based on such lattice problems and is considered secure against known quantum algorithms, aligning with future regulatory expectations that systems move toward post-quantum cryptography.

Despite its promise, FHE also raises some new questions for regulators. For example, if a cloud service never sees plaintext data, how should compliance audits be conducted? How should

incident response be handled if an encrypted dataset is leaked (though arguably the risk is much lower)? Also, export control regulations sometimes classify strong encryption technology itself as munitions or restricted dual-use technology - as FHE matures, legal frameworks may need to catch up to explicitly allow cross-border use of such encryption for data processing.

In summary, FHE is a powerful tool to enhance regulatory compliance by keeping data encrypted at all times. It aligns well with the spirit of laws like GDPR and HIPAA, providing a technical means to achieve privacy-by-design in outsourced computing. Early adopters of FHE in industries handling highly sensitive data could set new best practices in compliance. That said, regulators and policymakers will need to stay informed about FHE developments to adjust compliance guidelines and perhaps to incentivize the use of such privacy-preserving technologies in the broader market.

# Comparative Analysis of FHE libraries

The field of fully homomorphic encryption has progressed from a theoretical concept to a practical, though still evolving, technology. A number of open-source FHE libraries are now available, each implementing one or more of the known FHE schemes. In this section, we compare some of the prominent FHE libraries and frameworks, highlighting their features, strengths, and limitations, especially in the context of machine learning applications.

Microsoft SEAL: SEAL is one of the most widely used FHE libraries (as evidenced by GitHub popularity). Developed by Microsoft Research, SEAL is written in C++ and provides implementations of the BFV (Brakerksi-Fan-Vercauteren) and CKKS (Cheon-Kim-Kim-Song) schemes. These are second- and fourth-generation FHE schemes respectively - both are *leveled* FHE schemes focusing on batching and efficient arithmetic on encrypted vectors. SEAL emphasizes ease of use and has well-optimized operations for both BFV (which is used for exact arithmetic on integers) and CKKS (which supports approximate arithmetic on real numbers). Microsoft SEAL does not natively support bootstrapping in all cases (especially for CKKS in early versions, though research add-ons exist), meaning encrypted computations have a limit to their depth unless a user implements an advanced extension. SEAL comes with a .NET wrapper and has been integrated into other tools; however, it does not officially provide a Python API or high-level machine learning integration out of the box. Users who want to work with SEAL in Python often use community projects or bindings (for example, the OpenMined community developed "TenSEAL", a library built on SEAL, to enable PyTorch tensor operations under CKKS). In terms of ML, SEAL/CKKS is well-suited for inference on standard neural network layers (since it can natively handle decimal approximations), and several research papers have shown neural network inference on encrypted data using SEAL. The main challenges with SEAL are the lack of native bootstrapping (limiting the depth of feasible computations) and the need to manage encryption parameters (poly modulus degree, coefficient modulus, etc.) which requires cryptographic expertise to tune for each application.

HElib: Originally developed by IBM, HElib is another pioneer FHE library. It is written in C++ and implements mainly the BGV scheme (and more recently CKKS as well). HElib was among the first open-source FHE libraries after Gentry's breakthrough, and it provided an early platform

to experiment with leveled FHE and bootstrapping techniques. It does not offer official wrappers for other languages, making it a bit less accessible to non-C++ users. HElib supports bootstrapping for BGV (IBM demonstrated bootstrapping in HElib a few years ago), but it is known to be somewhat lower-level and harder to use than SEAL. In our project, we decided not to use HElib as our primary tool, partly because its capabilities were being subsumed by newer frameworks and it lacked the ease-of-use features we desired. However, HElib remains a very powerful library for those who need to implement custom FHE computations and is continually updated by the research community.

PALISADE / OpenFHE: PALISADE was an open-source FHE library that has now evolved into OpenFHE. OpenFHE is essentially the successor, combining efforts from multiple organizations (including the original PALISADE team, DARPA, etc.) to create a comprehensive homomorphic encryption library. Impressively, OpenFHE supports all four major FHE scheme "families" - BFV, BGV, CKKS, and TFHE (through an integration) - making it one of the most feature-complete libraries. It is written in C++ but importantly provides an official Python API, which greatly lowers the barrier for use. OpenFHE is designed to be modular and includes many advanced features such as bootstrapping for CKKS, multi-party capabilities, and hybrid schemes. For someone building ML solutions, OpenFHE could be a strong choice because you could prototype in Python and then optimize in C++ if needed. The inclusion of TFHE means OpenFHE can do binary gate computations with bootstrapping (fast individual bit operations) as well as arithmetic with CKKS (fast batched linear algebra on approximate numbers). However, using OpenFHE's full power still requires understanding a lot of cryptographic settings, and being a newer consolidation, it might not have the same level of community support or documentation for each feature yet.

Zama's Concrete / Concrete-ML (TFHE-rs): Zama, a French cryptography startup, has developed the Concrete library, which is built on TFHE-rs - a Rust implementation of the TFHE scheme. TFHE-rs implements the third-generation TFHE scheme (focused on fast bootstrapped binary gates) and is highly optimized for boolean and small integer operations. Concrete provides a Python wrapper around TFHE-rs, allowing Python developers to define functions that get compiled to FHE circuits. On top of Concrete, Zama built Concrete-ML, which specifically targets machine learning use cases. This includes ready-made procedures to take simple ML models (like linear models, tree models, small neural networks) and automatically generate FHE equivalents. The advantage of Concrete-ML/TFHE is the ability to perform *bit-level accurate computation with bootstrapping*, which means no loss of precision due to scheme approximations - the only quantization error is the one you deliberately introduce to fit into a certain bit-width. For example, we saw that our FHE models had essentially the same accuracy as plaintext; this is partly thanks to using TFHE which doesn't inherently approximate arithmetic (unlike CKKS which introduces small errors). The trade-off, as our results show, is speed – operating bit-by-bit is slow when you need to simulate large arithmetic operations (as evidenced by Table 2 and our timing results). Zama's ecosystem is rapidly improving: they have demonstrated GPU acceleration for TFHE where appropriate, and their latest release (v0.7 of TFHE-rs) introduced multi-GPU support that drastically speeds up large integer operations. This kind of hardware acceleration is essential if TFHE-based libraries are to handle bigger ML workloads. Concrete-ML is relatively young and supports a limited set of models (as listed in our methodology), but it is under active development, and we found it quite user-friendly for those models. One limitation we hit is that certain ML algorithms are not yet implementable

in Concrete-ML - for example, there's no straightforward way to do *encrypted training* for most models aside from logistic regression (training generally remains plaintext). Also, models that require significant branching logic or very deep circuits (like large neural networks) are not practical due to performance constraints. Nonetheless, Concrete-ML's approach of automating FHE conversion is a glimpse into the future: if more complex models can be supported with acceptable performance, data scientists could train models normally and then deploy them to an FHE-enabled cloud service with minimal expertise in cryptography.

**Lattigo:** Lattigo is a library implemented in Go (Golang) that provides FHE capabilities, focusing on BGV, BFV, and CKKS schemes. It supports both integer and approximate arithmetic and even includes an implementation of CKKS bootstrapping. Lattigo is quite popular in academic circles for distributed and cloud applications, because Go is well-suited for concurrent computation and deployment. It's also one of the few libraries in a language other than C++ that is performance-oriented. We considered using Lattigo for our project, as it is well-maintained and has shown good performance, but ultimately we prioritized Python interoperability and high-level ML integration (which Lattigo does not directly offer). If one's use case involves building a cloud service in Go or integrating with Go-based systems, Lattigo is a compelling choice.

**HEAAN:** HEAAN is the name of both the original CKKS implementation (from the authors of CKKS scheme) and an acronym referring to "Homomorphic Encryption for Arithmetic of Approximate Numbers." As the name suggests, HEAAN (the library) implements only the CKKS scheme. It's often used as a research reference code for CKKS developments. For practical purposes, one might choose a more comprehensive library unless only CKKS is needed. The importance of HEAAN is that many CKKS-based applications (like robot navigation, signal processing on encrypted data, etc.) were prototyped on it. In ML context, CKKS (hence HEAAN or SEAL's CKKS) is attractive for things like inference on deep learning models because you can encode many data points in one ciphertext (using vector batching) and perform fast linear algebra on encrypted vectors. The downside is that any operation that doesn't fit the model of additions and multiplications (like non-polynomial activations) have to be approximated, and with each operation the numeric error grows slightly.

**Other Frameworks:** Beyond these libraries, the FHE landscape includes other notable tools: for example, **Tfhe** (C++ library for TFHE by the original TFHE authors, separate from Zama's Rust version), **CryptoAPI PALISADE** (the original PALISADE with slightly different interfaces), **IBM HElayers** (an IBM library built on top of HElib and SEAL, aimed at simplifying development of FHE applications, particularly ML; however HElayers is not fully open-source at the time of writing). The open-source machine learning community also has projects like **CryptoNet** (from Microsoft Research, an early prototype for neural networks on encrypted data) and **CHET** (Compiler for Homomorphic Evaluation of Tensor programs, an academic project to compile neural nets to FHE). These are specialized, but Concrete-ML is the one that has gained momentum recently for ease of use.

In **summary**, Table 1 provides a high-level comparison of select FHE libraries on key attributes.

Table 1. Comparison of Select Open-Source FHE Libraries and Frameworks

| Library | Schemes Supported | Language (APIs) | Notable Features | Notes for ML |
|---|---|---|---|---|
| Microsoft SEAL | BFV, CKKS | C++ (/.NET, some unofficial Python) | Optimized arithmetic, batching; no native bootstrapping by default | Good for batched linear algebra (CKKS) but requires external tooling for ML integration (e.g., TenSEAL) |
| IBM HElib | BGV, CKKS | C++ | Bootstrapping for BGV; research-oriented | Powerful but lower-level; fewer ML-specific utilities |
| OpenFHE (successor to PALISADE) | BFV, BGV, CKKS, TFHE (+ others) | C++ (Python API) | Very comprehensive, supports multi-party, bootstrapping, etc. | Versatile for various use cases; Python API eases experimentation |
| Zama Concrete / Concrete-ML (TFHE-rs) | TFHE (binary, integers) | Rust (Python via Concrete-ML) | Programmable bootstrapping (fast bootstraps), GPU acceleration available | Excellent for boolean circuits and exact integer ML inference; integrates directly with simple ML models |
| Lattigo (EPFL) | BFV, BGV, CKKS | Go | Native Go library, CKKS bootstrapping implemented | Useful for Go cloud services; can do CKKS neural net inference with bootstrapping |
| HEAAN (SNU) | CKKS | C++ (Python wrappers via HEAAN-Py?) | Original CKKS implementation, focus on approximate math | Good for floating-point like computations; lacks other scheme support |

For our project’s needs - building an ML service with FHE - the combination of Concrete-ML (for model integration) and TFHE/Concrete (for the cryptographic backend) was chosen, as it offered the quickest path to get machine learning models running on encrypted data. The choice was influenced by the availability of a Python framework (we could leverage our existing Python ML pipeline), and by the fact that TFHE’s fast bootstrapping aligns with needing to evaluate potentially unbounded depth circuits (e.g., loops in inference computations) without worrying about running out of noise budget. The downside, as the results showed, is performance - we accepted that for a prototype. If, instead, our priority was to maximize speed and we were willing to handle approximate results, a CKKS-based approach (like using OpenFHE’s CKKS or SEAL through TenSEAL) might have yielded faster inference for certain models (e.g., neural networks that fit in a single bootstrapping-free round). Ultimately, each library has strengths, and the “best” choice depends on the use case: TFHE-based libraries excel when you need many boolean operations (e.g., comparing bits, doing arbitrary logic, re-encrypting frequently), while CKKS-based libraries excel in scenarios like linear algebra on encrypted vectors (e.g., matrix multiplications for neural net layers) where the overhead per operation is lower and many operations can be done before needing to refresh.

It's encouraging to see a healthy competition and diversity in FHE frameworks - it spurs faster improvement. In the near future, we expect convergence where high-level frameworks (like Concrete-ML or others) might abstract away the differences, intelligently choosing the best scheme for a given task (even mixing them). For example, an ML framework could use CKKS for layers of a model that are linear and TFHE for layers that require bit-level decision logic. Such hybrid approaches could combine strengths to tackle larger ML workloads under encryption.

# Methodology

To assess the viability of FHE for machine learning, we conducted experiments on a selection of different machine learning and deep learning algorithms. Our goal was to compare each algorithm's implementation under conventional settings (plaintext processing) with an FHE-based implementation, focusing on Concrete-ML for the latter. Concrete-ML is a Python framework that compiles certain ML models into their FHE equivalents using Zama's Concrete library (built on the TFHE scheme). For comparison, we used standard implementations from Scikit-Learn (sklearn) for plaintext ML models.

The models evaluated span a range of simple to moderately complex algorithms, covering both classification and regression tasks:

- Linear Regression
- Logistic Regression
- Linear Support Vector Machine (SVM) for Regression (ε-SVR) and for Classification (linear SVC)
- Poisson Regression
- Decision Tree Classifier and Decision Tree Regressor
- Gradient Boosting (XGBoost) Classifier and Regressor
- Fully Connected Neural Network

Each model was first trained and evaluated using traditional methods on plaintext data (using sklearn or XGBoost library as appropriate). We then used Concrete-ML to create an equivalent model that operates on encrypted data. In most cases, Concrete-ML does inference on encrypted inputs using a model trained on plaintext data. The model training itself occurs in plaintext domain, after which the trained model is compiled into an FHE-compatible form. The one exception is logistic regression: Concrete-ML currently supports training logistic regression models while the training data is encrypted (using an encrypted variant of stochastic gradient descent).

We measured three key metrics for each model: Training Time, Inference Time, and Accuracy. For training time, we recorded the time to train the model (on plaintext data for all models, and additionally the time for encrypted-data training in the case of logistic regression). For inference time, we measured the time to make predictions on a dataset (with the model operating on plaintext vs operating on encrypted inputs). Accuracy was evaluated on a test

dataset, using standard metrics (e.g., classification accuracy or regression error) and comparing the FHE model's performance to its plaintext counterpart.

All experiments were run on the same hardware configuration to ensure consistency in comparisons (e.g., a standard x86 CPU machine was used for both sklearn and Concrete-ML tests; no GPU acceleration was applied during these particular measurements). We used identical data splits and hyperparameters for both versions of each model. The datasets for each algorithm were chosen to be of moderate size to keep FHE processing feasible (for instance, tens of thousands of samples for simpler models and smaller subsets for more complex models like the neural network, to avoid excessive FHE computation time). In classification tasks, we ensured class balance to avoid skewed accuracy results. The Concrete-ML library imposes some constraints on models (for example, requiring fixed-point quantization of inputs and model weights, since TFHE operates on integers/binary). We used Concrete-ML's default quantization and compilation settings for each model, which typically quantize inputs to a small bit-width (e.g., 8-bit or 16-bit integers) and approximate non-linear functions (such as the sigmoid in logistic regression) with lookup tables or low-degree polynomials suitable for FHE execution.

By comparing the sklearn (plaintext) and Concrete-ML (FHE) versions of each algorithm, we can quantify the overhead introduced by homomorphic encryption. We summarize these comparisons in terms of relative slowdowns (factor increase in time) and any changes in accuracy. Additionally, we document any practical hurdles encountered during implementation (such as limitations of the FHE library for certain model types or memory usage issues) as part of assessing the technology's current maturity.

# Experimental Results

Our experimental results highlight a substantial performance penalty when using FHE for ML, though model accuracy remained largely intact. We summarize the findings for each metric below.

Training Time: For most models, training is done in plaintext even for the FHE workflow (since Concrete-ML compiles the model after training). As a result, training times for those models were approximately 10 to 100 times slower than their scikit-learn counterparts. The FHE-enabled logistic regression training was dramatically slower. In fact, training logistic regression on encrypted data was the slowest operation by far, taking on the order of 100,000× longer than training the same model on plaintext data. (For context, what took milliseconds in plaintext could take hours under FHE for logistic regression training.) This five-orders-of-magnitude slowdown is due to the overhead of performing each gradient descent update homomorphically, where even basic arithmetic on encrypted numbers is costly. Most other algorithms do not yet support encrypted training in current libraries – and based on this result, it is evident that significant optimizations would be needed before homomorphic training of complex models becomes practical.

Even when training on plaintext, we observed some overhead in the overall FHE workflow due to the compilation step. Concrete-ML must compile the trained model into an FHE circuit, which can take from a few seconds up to several minutes depending on the model complexity

(this one-time cost is not present in a plaintext scenario). For example, compiling a complex model like the fully connected neural network or XGBoost model to an FHE circuit took several minutes. This compilation time is a preprocessing cost and not needed for every inference, so it's usually acceptable, but it does add to the deployment latency for FHE models.

Inference Time: The latency for making predictions on encrypted data was 3–5 orders of magnitude longer compared to traditional plaintext inference across the board. In other words, operations that take microseconds or milliseconds on plaintext might take seconds (or more) on ciphertexts. For instance, evaluating a single decision tree or linear model on one data point took on the order of tens to hundreds of milliseconds in FHE, versus microseconds in plaintext - roughly a 1,000× slowdown. More complex models like XGBoost (which ensembles many trees) or the neural network incurred even larger slowdowns, sometimes tens of thousands of times slower than their plaintext counterparts. Concretely, an XGBoost classification that took ~1ms per sample in plaintext could take on the order of ~10s in the FHE version.

One interesting observation was that the inference time did not depend on whether the model itself was trained on plaintext or encrypted data. For example, the logistic regression model trained homomorphically had virtually the same prediction latency as the one trained normally (plaintext) and then used for FHE inference. This is expected, because once the model parameters are set, the homomorphic execution of the logistic regression has the same complexity regardless of how the weights were obtained. It underscores that the primary performance cost lies in *inference on encrypted inputs*, not in how the model was trained.

Another observation is that certain operations are particularly expensive under FHE. Concrete-ML (using TFHE) performs all operations as boolean or binary arithmetic circuits. Operations like addition and multiplication of small integers are reasonably fast (on the order of tens of milliseconds), but operations that require heavy binary circuit implementation, such as division or comparison, can be extremely slow. We noted, for instance, that any algorithm requiring division (e.g., computing a mean or a probability) would incur a large penalty. Table 2 illustrates the latency of various basic operations under TFHE encryption (based on an independent benchmark by Zama):

Table 2. Performance of basic TFHE operations on encrypted integers (by bit-width)

| **Operation \ size (bits)** | **8** | **16** | **32** | **64** | **128** | **256** |
|---|---|---|---|---|---|---|
| **Negation (-)** | 49.3 ms | 57 ms | 79.5 ms | 104 ms | 167 ms | 184 ms |
| **Add, Sub (+,-)** | 58.4 ms | 62 ms | 85.3 ms | 113 ms | 183 ms | 195 ms |
| **Mul (*)** | 103 ms | 148 ms | 219 ms | 404 ms | 1.14 s | 3.89 s |
| **Equal, Not Equal (eq, ne)** | 33.6 ms | 56.4 ms | 58.5 ms | 81.4 ms | 83.4 ms | 107 ms |
| **Max, Min (max, min)** | 78.7 ms | 99.3 ms | 124 ms | 151 ms | 197 ms | 254 ms |

| | | | | | | |
|---|---|---|---|---|---|---|
| **Div / Rem (/, %)** | 728 ms | 1.61 s | 3.66 s | 8.65 s | 21.8 s | 57 s |

As seen in Table 2, even modest-size multiplications can take hundreds of milliseconds, and division or remainder operations can take seconds when operating on typical 32-64 bit numeric ranges. These lower-level timings help explain the higher-level model delays we measured. Models that can avoid expensive non-linear operations (e.g., linear models, which mostly use additions and multiplications) perform relatively better under FHE than those that require divisions or many comparisons (e.g., a deep decision tree or complex activation functions).

Accuracy: Despite the massive slowdowns in runtime, a promising finding is that model accuracy under FHE was nearly the same as without FHE. The encryption itself is "noise-free" in terms of correctness (FHE provides exact or appropriately rounded results), so any accuracy differences stem from quantization or approximation needed to make the model FHE-friendly. In our tests, the majority of FHE models' accuracies were within **<1%** of the plaintext models' accuracies. In several cases, the FHE model achieved *identical* accuracy to the plaintext model. Figure 1 illustrates the accuracy difference between FHE (Concrete-ML) and plaintext (sklearn) implementations for each algorithm. Most bars are barely visible, indicating minimal loss in accuracy. The worst-case accuracy degradation we observed was still under 5 percentage points. For instance, the FHE version of the XGBoost regressor had a small drop in $R^2$ score compared to plaintext, and the FHE SVM classifier was a couple of percentage points lower in accuracy than the plaintext SVM on our test data. On the other hand, several models such as logistic regression, Poisson regression, the decision tree classifier, and even the fully connected neural network showed essentially no difference in accuracy between the FHE and non-FHE implementations. This demonstrates that Concrete-ML's quantization and approximations did not significantly compromise the model's predictive performance in our experiments.

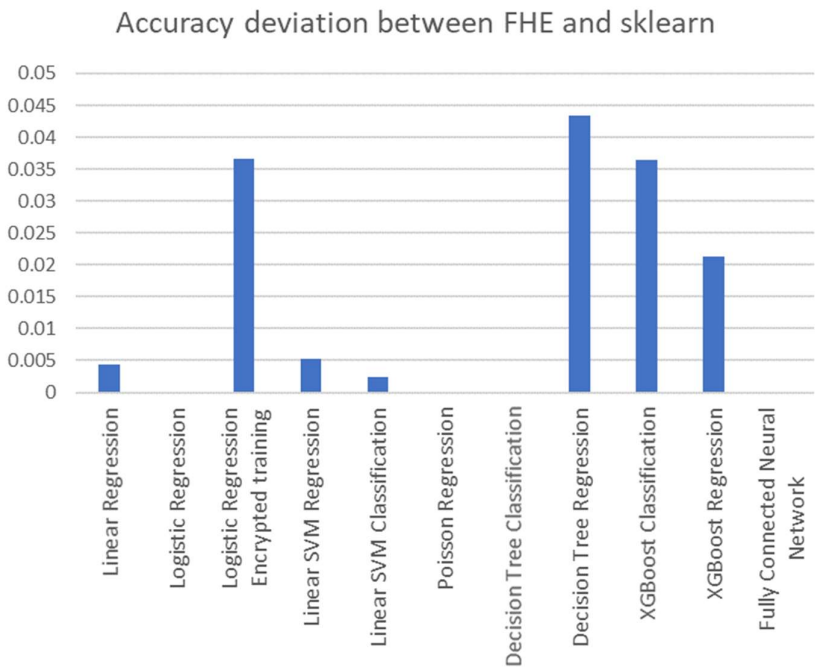


*Figure 1. Accuracy difference between Concrete-ML (FHE) and plaintext (sklearn) models. Bars represent percentage point decrease in accuracy for the FHE model relative to plaintext.*

Overall, our results indicate that, from a **security and privacy** standpoint, FHE integration succeeds in maintaining model fidelity - an encrypted ML model can achieve nearly the same outcomes as a normal model, which is crucial for viability. However, from a **performance** standpoint, the current cost of using FHE is exceptionally high. In practical terms, this would limit the use of FHE-enabled ML to scenarios where data confidentiality is so critical that one is willing to accept substantial computation latency and resource usage. Examples might include off-line or batch analysis of sensitive data (where results are needed hours later, not in real-time), or extremely sensitive computations where even a slow answer is preferable to exposing data. For applications that demand real-time or high-throughput processing, today's FHE technology would struggle to meet requirements.

These findings underscore the importance of ongoing research to optimize FHE. Some optimizations are already on the horizon (e.g., GPU acceleration for FHE). But before exploring future improvements, we first consider how FHE aligns with regulatory frameworks and what limitations exist beyond just raw performance numbers.

# Pilot Solution

The TruPS pilot solution demonstrates the practical application of Fully Homomorphic Encryption (FHE) for secure, privacy-preserving machine learning. Built around Zama's Concrete-ML framework and the TFHE scheme, the pilot enables encrypted inference workflows for four machine learning models:

- Linear regression
- Logistic regression
- Decision tree regressor
- Decision tree classifier

For details see:

https://github.com/RandomRedLtd/trups-public/blob/main/README.md

# Future Work

Our exploration of FHE for machine learning, while affirming its potential, also exposes several areas where further research and development are necessary. Here we outline some future work and improvements that could make FHE more practical for real-world ML applications:

1. Performance Optimization: The most glaring need is to speed up FHE computations. There are multiple avenues for this:

- *Algorithmic improvements:* Cryptographers are continually refining FHE algorithms. Recent papers have improved bootstrapping in both TFHE and CKKS schemes. For example, new bootstrapping techniques or better parameter selection can cut down the noise and thereby reduce the overhead. The community is also exploring optimized homomorphic operations (e.g., faster homomorphic multiplication for high-precision numbers).
- *Parallelization and Hardware Acceleration:* FHE is highly parallelizable since operations on different parts of ciphertexts can often be done independently. Utilizing GPUs and FPGAs can offer huge speedups. We already see multi-GPU support in TFHE-rs v0.7 which achieved unprecedented performance gains for certain operations. Future work will likely optimize specific kernels (like the FFT-based polynomial multiplications at the heart of homomorphic operations) for GPUs and specialized hardware. There is also interest in developing ASICs (custom chips) for FHE; companies and academia have started designing hardware blocks specifically for accelerating homomorphic encryption tasks. If these materialize, we could see speedups of one or two orders of magnitude, which when combined with algorithmic improvements, might bring FHE from 1000× slower than plaintext to maybe just 10× or so for some tasks - a threshold at which adoption becomes much more feasible.
- *Compression and Memory:* Homomorphic ciphertexts are large and memory bandwidth and storage can become bottlenecks. Techniques to compress ciphertexts

(as mentioned in the TFHE-rs v0.7 update) are in development. Reducing ciphertext size or enabling faster memory movement will improve overall throughput in ML inference where many ciphertexts (e.g., a batch of inputs) need to be processed.

2. Support for Complex Models: Currently, the set of models that can be run under FHE is limited. Expanding this is important for broader ML adoption. For example, deep learning models like convolutional neural networks (CNNs) are not yet practical with FHE at non-trivial scale. Future work might explore:

- *Efficient homomorphic alternatives for neural network layers:* There is ongoing research on new activation functions or network architectures that are FHE-friendly. For instance, replacing expensive operations (like non-linear activations or softmax) with polynomial or comparators that work well under encryption.
- *Transfer learning and inference on large models:* One idea is to take a pre-trained large model (like a deep CNN on images) and create a smaller FHE-friendly model (through knowledge distillation or model compression) that can run on encrypted data with acceptable accuracy loss. This way, we leverage powerful models but only deploy a simplified version for the encrypted domain.
- *Hybrid approaches:* Perhaps not all parts of a pipeline need to be FHE. Combining FHE with other privacy tech (like secure multiparty computation or trusted execution environments) might allow more complex workflows. For example, sensitive parts of a model can be computed homomorphically while others run in a secure enclave, balancing speed and security. Research into such hybrid privacy-preserving ML could open up use cases that pure FHE cannot yet handle.

3. Encrypted Training Techniques: So far, FHE has been mostly used for *inference* on encrypted data, with model training assumed to be on plaintext. Enabling more algorithms to train on encrypted data would be revolutionary - it would allow collaborative model training without exposing raw data (useful in multi-party scenarios where data is siloed due to privacy). Our experiment with logistic regression encrypted training is a small step in this direction. Future work needs to address:

- *Efficiency of encrypted training:* Homomorphic gradient descent is extremely slow as we saw. One possible improvement is using simpler or approximate training methods that converge in fewer iterations. Another is leveraging parallelism - e.g., multiple data owners compute gradients on their encrypted data in parallel and aggregate them (there is overlap here with federated learning and MPC).
- *Regularization and other training components:* Training isn't just multiplication and additions; it involves things like randomness (for initialization, or shuffling data) and non-linear operations (activation functions in neural nets). Homomorphic analogues for these (e.g., ways to generate random noise in FHE, or alternative loss functions that are easier to compute under encryption) are areas of active research.
- *Hyperparameter tuning under encryption:* Even if we manage to train a model under FHE, tuning it (trying different hyperparameters) could be prohibitively slow. Methods to efficiently explore model configurations without full retraining each time, or using automated tuning that is encryption-aware, would be useful.

4. Usability and Abstraction: For FHE to be adopted by the wider ML community, it must become easier to use. This means higher-level abstractions where data scientists can apply encryption with minimal cryptography knowledge. Concrete-ML is a good start, but future improvements could include:

- *Integrations with popular ML frameworks:* We might envision FHE support integrated into libraries like scikit-learn or TensorFlow. For instance, a future TensorFlow Privacy module could allow one to declare certain tensors as "encrypted" and behind the scenes use FHE for those computations. This requires significant engineering but would put FHE in the hands of ML practitioners directly.
- *Automated parameter selection:* Choosing encryption parameters (polynomial moduli, security parameters, etc.) is complex. Tools are being developed to automate this based on desired security level and expected operations (Concrete does some of this already). Eventually, a user might simply specify "encrypt this data with 128-bit security" and the library picks all needed parameters optimally.
- *Debugging and Visualization:* Developing with FHE can be challenging because you can't easily inspect intermediate values (they're encrypted). Future developer tools might simulate FHE computations (to approximate what noise levels are, or where an operation might fail) and give feedback, or visualize the "circuit" that a computation will create. This would help optimize and understand performance bottlenecks in a complex encrypted computation.

5. Standardization and Interoperability: As multiple libraries emerge, standards like the HomomorphicEncryption.org's schemes and noise metrics are important so that results and methods can be compared apples-to-apples. Ongoing efforts will likely yield standard formats for FHE-encrypted data (so one system's output could be another's input), and perhaps standardized APIs. This will encourage a healthy ecosystem where improvements in one library can benefit others.

6. Security and Compliance Research: While FHE is theoretically secure, implementing and using it correctly is non-trivial. Future work includes formal verification of FHE implementations (to ensure no side-channel leaks, etc.), and building audit tools for systems that use FHE. Also, demonstrating compliance (as touched on in the regulatory section) could become a field of study: e.g., developing a framework to certify that a cloud service truly never sees plaintext when using FHE, which auditors can trust. This might involve open-sourcing critical components and enabling reproducible builds to ensure no backdoors.

In summary, the road ahead for FHE in machine learning is challenging but exciting. Progress is being made on many fronts, and there's a clear *momentum* in the community (bolstered by competitions, hackathons, and increasing funding for privacy tech). Each year has brought an order-of-magnitude improvement in some aspect of FHE. If this trend continues, within a few years we might have FHE toolkits that make privacy-preserving machine learning not just a niche novelty, but a default option for sensitive data applications. Our work, and that of others in the field, is moving us closer to a future where individuals and organizations can reap the benefits of data-driven ML/AI solutions without surrendering privacy or trust.

# Conclusion

Fully Homomorphic Encryption holds the promise of revolutionizing data security by allowing computations to be performed on encrypted data. In this white paper, we presented a security overview of FHE and evaluated its performance in the context of machine learning. Our investigation using Concrete-ML and several representative ML models showed that, at present, FHE entails a substantial performance overhead - often rendering computations thousands of times slower - yet it can achieve almost the same accuracy and outcomes as traditional processing. This dichotomy underscores the current state of FHE: technically feasible and secure, but not yet efficient enough for broad, real-time use across different AI/ML models.

From a security standpoint, the ability to keep data encrypted throughout processing is a game-changer. It provides strong guarantees that even if a computational environment is untrusted or compromised, the data remains confidential. This capability directly addresses rising concerns over data privacy and regulatory compliance, enabling new ways to share and analyze sensitive information without exposing it. The inherent post-quantum resilience of lattice-based FHE schemes also means solutions built today are safeguarded against tomorrow's cryptographic threats, offering long-term value.

From a practical standpoint, our performance assessment makes it clear that FHE in 2025 is mostly suitable for niche scenarios - such as processing highly sensitive data in batch mode, or applications where security trumps all other considerations. The overhead in training and inference time, while gradually improving, is still a relevant barrier. For instance, training even a simple model like logistic regression on encrypted data required an inordinate amount of time in our experiments, and running inference on encrypted inputs, though faster than before, is far from real-time for anything but the simplest tasks. These limitations mean that for now, FHE will complement rather than replace conventional approaches. Hybrid architectures may emerge (using FHE for the most sensitive parts and plaintext processing elsewhere), until such time that FHE becomes more optimized.

Our comparative study of FHE libraries illustrated a vibrant landscape of tools, each advancing the state of the art. This competition and diversity are healthy for the field, pushing improvements in different directions - whether it's better algorithms (e.g., CKKS bootstrapping techniques), better software engineering (easy-to-use APIs, integration like Concrete-ML), or leveraging hardware (GPU/FPGA acceleration). It is reasonable to expect that FHE performance will continue to improve by leaps and bounds in coming years. As it does, more applications of FHE in machine learning will move from research demos to deployed systems. In particular, if the community can achieve even one or two more orders of magnitude improvement in speed, many online inference tasks could become viable under FHE, unlocking use cases in healthcare diagnostics, finance (e.g., fraud detection on encrypted transaction data), and government (secure analytics on census or tax data), to name a few.

In conclusion, the work presented demonstrates that FHE for machine learning is not just an academic idea but a working reality - albeit one that comes with trade-offs. Organizations should begin evaluating FHE for their highest-value privacy-sensitive use cases, keeping an eye on the rapid progress in this domain. Meanwhile, researchers and engineers must tackle the

remaining challenges: making FHE faster, more accessible, and integrating it seamlessly into the data science workflow. The convergence of encryption and ML/AI via FHE embodies the ideal of "privacy-preserving innovation," where we no longer have to sacrifice data privacy to benefit from data insights. As FHE technology matures, it has the potential to become a foundational layer in secure computing, ensuring that the future of machine learning is not only intelligent but also trustworthy and secure.

# References

1. A. Al Badawi and Y. Polyakov, "Demystifying Bootstrapping in Fully Homomorphic Encryption," Cryptology ePrint Archive, Report 2023/149. (Explains bootstrapping techniques and performance improvements from 30 minutes to practical times)
2. C. Gentry, "Fully homomorphic encryption using ideal lattices," *41st ACM Symposium on Theory of Computing (STOC)*, pp. 169-178, 2009. (Seminal paper introducing the first FHE scheme)
3. I. Chillotti *et al.*, "TFHE: Fast Fully Homomorphic Encryption over the Torus," Cryptology ePrint Archive, Report 2018/421. (Introduces the TFHE scheme used in Concrete)
4. Changmin Lee et al. „Faster TFHE Bootstrapping with Block Binary Keys", https://eprint.iacr.org/2023/958.pdf
5. J. H. Cheon *et al.*, "Homomorphic Encryption for Arithmetic of Approximate Numbers," *ASIACRYPT 2017*, pp. 409-437, 2017. (Proposes the CKKS scheme for approximate arithmetic on encrypted data)
6. Zama, "Benchmarks of homomorphic operations using TFHE-rs," Zama Documentation, https://docs.zama.ai/tfhe-rs/get-started/benchmarks, accessed Feb. 22. (Benchmark data for TFHE-rs operations; source for Table 2)
7. J.-B. Orfila *et al.*, "TFHE-rs v0.7: Ciphertext Compression, Multi-GPU Support and More," Zama Blog, https://www.zama.ai/post/tfhe-rs-v0-7-ciphertext-compression-multi-gpu-support-and-more, July 5, 2024 (Announces GPU support and performance improvements in TFHE-rs)
8. Zama, "Concrete-ML Documentation: Training Logistic Regression on Encrypted Data," https://docs.zama.ai/concrete-ml/built-in-models/training, accessed Feb. 22, 2025. (Describes how Concrete-ML implements SGD for logistic regression on encrypted data)
9. I. Uglik *et al.*, "Analysis of available open source FHE libraries," NGI Sargasso TruPS Project Deliverable A1.4, Jan. 2025. (Internal report comparing FHE libraries; basis for Comparative Analysis section)
10. I. Uglik *et al.*, "Identifying technical limitations related to FHE," NGI Sargasso TruPS Project Deliverable A2.1, Feb. 2025. (Internal report on FHE performance and usability limitations; provided data for performance discussion)
11. Zama, "Concrete-ML GitHub Repository," https://github.com/zama-ai/concrete-ml, accessed May 2025. (Zama's ML framework GitHub repo)
12. Zama, "Concrete Core Library GitHub Repository," https://github.com/zama-ai/concrete, accessed May 2025. (Zama's core FHE library repo)
13. Zama, "Concrete-ML Release Notes v0.7," https://docs.zama.ai/concrete-ml/release_notes/0.7, accessed May 2025. (Release notes detailing GPU and performance improvements)

14. RandomRed Ltd., "TruPS Project Overview," GitHub README, https://github.com/RandomRedLtd/trups-public/blob/main/README.md, accessed May 2025. (Overview of the TruPS project objectives and structure)
15. D. Akinyele and O. Godwin, Regulatory compliance for homomorphic encryption in network traffic analysis, ResearchGate, October 2024. https://www.researchgate.net/publication/384729322_Regulatory_Compliance_for_Homomorphic_Encryption_in_Network_Traffic_Analysis
16. Anishkumar Dhablia, The role of cybersecurity policies in ensuring compliance with U.S. federal and state network security laws, Network Security, 2024(7), 14–21, 2024. https://networksecuritypub.com/index.php/journal/article/view/45
17. A. Atadoga, O. A. Farayola, B. S. Ayinla, O. O. Amoo, T. O. Abrahams, and F. Osasona, A comparative review of data encryption methods in the USA and Europe, Computer Science & IT Research Journal, 5(2), 447–460, 2024. https://doi.org/10.51594/csitrj.v5i2.815
18. C. Crane, 10 data privacy and encryption laws every business needs to know, Hashed Out by The SSL StoreTM, March 11, 2021. https://www.thesslstore.com/blog/10-data-privacy-and-encryption-laws-every-business-needs-to-know/#7-healthcare-insurance-portability-and-accountability-act-%E2%80%94-united-states
19. M. J. Haber, B. Chappell, and C. Hills, Regulatory compliance, APress Media, LLC, pages 297–373, 2022. https://doi.org/10.1007/978-1-4842-8236-6_8
20. G. Macon, A narrative review of advantageous cybersecurity frameworks and regulations in the United States healthcare system, ProQuest, 2023. https://www.proquest.com/openview/33932e4bd19bc866ba862d48ad7ed4f8/1?pq-origsite=gscholar&cbl=18750&diss=y
21. K. Munjal and R. Bhatia, A systematic review of homomorphic encryption and its contributions in healthcare industry, Complex & Intelligent Systems, vol. 9, 2022. https://doi.org/10.1007/s40747-022-00756-z
22. N. Rico and J. Luis, Holistic business approach for the protection of sensitive data: Study of legal requirements and regulatory compliance at international level to define and implement data protection measures using encryption techniques, Openaccess.uoc.edu, 2021. https://openaccess.uoc.edu/handle/10609/90727
23. R. Sion and M. Winslett, Regulatory compliance, in Handbook of Financial Cryptography and Security, Taylor & Francis Group, 2020. https://www.taylorfrancis.com/chapters/edit/10.1201/9781420059823-27/regulatory-compliance-radu-sion-marianne-winslett
24. Srinivas Katkuri, Need of encryption legislation: Protecting India's digital realm and beyond, Indian Journal of Public Administration, 70(3), 562–578, 2024. https://doi.org/10.1177/00195561241271590
25. Davor Vinko, Kruno Miličević, Ivan Uglik, Adrijan Đurin, Madhurima Ray, Richard Lomotey, "Security overview and experimental performance assessment of using Fully Homomorphic Encryption (FHE) on machine learning algorithms", scientific paper submitted in a peer-review journal (under review)
26. Madhurima Ray, Richard Lomotey, Davor Vinko, Kruno Miličević, Ivan Uglik, Adrijan Đurin, "Evaluation of Sector-Specific EU and US Regulations for Fully Homomorphic Encryption in Data Transfer, Processing, and Storage", scientific paper submitted in a peer-review journal (under review)