A Machine Learning-Driven Computationally Efficient Encryption Framework for Secure Real-Time Data Processing with Homomorphic Encryption

Sonam[1]* ,Sanjay Kumar Sharma[2]

[1]Department of Computer Science and Engineering School of ICT, [Gautam Buddha University], [Gautam Buddha Nagar], [India]

[2]Department of Computer Science and Engineering School of ICT, [Gautam Buddha University], [Gautam Buddha Nagar], [India]

***Corresponding Author**

DOI: https://doi.org/10.51244/IJRSI.2026.1311000068

Received: 26 October 2026; Accepted: 31 October 2026; Published: 23 November 2026

ABSTRACT

Outsourced and streaming analytics require and demand Homomorphic Encryption (HE), which enables a server to compute directly on data it is never allowed to read. Moreover, the guarantee is very expensive – HE is slow, and tuning the encryption parameters for the worst case a system may be used for makes it even slower, so latency-sensitive streams are impractical. In this paper we introduce AMHE, a framework for adapting homomorphic encryption to the real-time data processing needs. AMHE introduces a light weight learning control plane in front of a CKKS encryption engine: a data-sensitivity classifier encrypts only the attributes that truly need security; an adaptive parameter optimizer learns the smallest secure parameters per CKKS request that fulfill the per-request latency target; and a workload-aware scheduler learns to pack and batch ciphertexts as the arrival rate changes. We implemented a prototype on Microsoft SEAL (TenSEAL), and tested it on three publicly available streamed datasets, one of which is medical, with AES, Paillier, BFV, and fixed-parameter CKKS. The 128-bit security level and the plaintext–ciphertext match of the encrypted results to the first two decimal places, while AMHE responded in about 22 ms, which is up to 23× faster than a worst case fixed CKKS context, and made up to 23× the throughput of the learned controller (about 1.3% per request). The results suggest that a significant amount of the overhead attributed to homomorphic encryption is due to the static, worst case configuration, and that using a learned controller can make privacy-preserving computation a reality.

Keywords: Homomorphic encryption · Machine learning · Real-time data processing · Privacy-preserving computation · CKKS · Secure edge computing

INTRODUCTION

Today there is vast amount of data coming out of sensors, wearables and edge devices that must be acted upon in real-time. Hospitals monitor patients all day long and factories monitor their machinery, and transport systems react to live traffic, and payment platforms score transactions in milliseconds. Many parts of this computation are moved to cloud or edge servers, both because of their elasticity and because of the machine-learning (ML) services they offer. The downside of the convenience is that data that is accessed beyond the owners access point is then stored on infrastructure that the owners cannot control, which is inconvenient not only from a privacy aspect, but also from a compliance standpoint, e.g. the GDPR and HIPAA [1]. This does not mean the problem is solved if the data is encrypted at rest and at the time of being moved, the server still has to decrypt it before it can perform any computation, and that is where the exposure is happening.

Homomorphic encryption (HE) eliminates that interval. It enables an evaluation of arithmetic operations on the ciphertext, without revealing any plaintext on the untrusted side [2, 3]. Since the initial full homomorphic

construction by Gentry [2] the field has advanced rapidly: the leveled schemes BGV [4], BFV [5] and CKKS [6] are now practical for real circuits; and CKKS has the approximate, real-valued arithmetic required by analytics and neural-network inference. These schemes have been made accessible to non-cryptographers' developers by libraries like Microsoft SEAL [7].

There is still a persistent conflict between security and performance. Each homomorphic operation is much more expensive than its non-homomorphic counterpart, and the number of times a computation can be chained can be limited by the amount of noise that can be accumulated. Typical approach to be correct and secure is to fix conservative parameters, large polynomial degrees and long coefficient-modulus chains, and use them in all records and all fields indiscriminately. A consistent and steady workload that is reasonable. It is wasteful on a bursty real-time stream: Not every record contains sensitive content; not every computation requires the most expensive multiplicative budget; and a variable arrival rate results in a half-used packing and batching capacity for the scheme. The symptom a user observes is a latency much greater than the 100 milliseconds that is considered acceptable for a real-time service.

In our starting point, the relationship is reversed: encryption is the starting point, while machine learning is the end. Instead, we request that ML govern the encryption protection, which it does with a speed sufficient for real time use. AMHE (Adaptive, Machine-learning-driven Homomorphic Encryption) monitors the data stream, along with the operating conditions, and dynamically reconfigures the HE pipeline, one request at a time. Encryption parameters are no longer fixed constants, but are predicted parameters, the leanest set that can meet the security bar and the latency budget..

In short, the paper makes the following contributions:

- AMHE, a design that puts a small ML control plane in charge of a CKKS encryption engine so that secure stream processing can meet explicit per-request latency budgets.
- Three cooperating learners: a data-sensitivity classifier for selective encryption, an adaptive parameter optimizer for latency-aware parameter choice, and a workload-aware scheduler for ciphertext packing and batching.
- A formulation of the security-constrained latency-minimization problem the framework solves, with an account of how the predictors are trained offline and run at negligible cost online.
- A SEAL-based prototype and its evaluation on three streamed public datasets, measuring latency, throughput, security level, numerical precision, and analytics accuracy, alongside an ablation that separates the effect of each component, a cross-validated evaluation of the learned models and their runtime overhead, and a study of how parameter selection adapts to computation depth.

The remainder of the paper follows. In Section 2, AMHE is compared with the previous studies in the field of homomorphic encryption, privacy-preserving ML, and adaptive security. The framework and its three learning components are developed in section 3. The design of the experiments is outlined in Section 4, the measurements are presented and interpreted in Section 5, and Section 6 is open about the things the study doesn't resolve. Section 7 is a concluding section.

Related Work

2.1 Homomorphic Encryption Schemes

The idea of computing on ciphertext has been around for a while. In 1978 Rivest, Adleman and Dertouzos introduced the concept of privacy homomorphisms [8] but for 30 years, there was still no scheme for evaluating arbitrary circuits until the lattice construction of Gentry [2] was introduced. The leveled schemes which followed made it usable in practice. BGV [4] and BFV [5] operate on integers and are used for exact arithmetic; CKKS [6] operates on real and complex numbers and is now the standard approach for analytics and neural-network inference. The compromises between them in terms of noise growth, cost of bootstrapping, and expansion of the ciphertext are explained in existing surveys [3, 9]; the Homomorphic Encryption Standard [10] defines parameter

choices to achieve a specified level of security; and libraries like SEAL [7] provide optimized implementations. No new scheme is introduced. To AMHE, CKKS is a given and focuses on how to make it work well.

2.2 Privacy-Preserving Machine Learning

Applying HE to machine learning is by now a well-populated area. CryptoNets [11] demonstrated that an even a neural network can be executed using encrypted inputs, and more recently GAZELLE [12] and nGraph-HE [13] have combined HE with secret sharing or compiled the computation into efficient primitives. These systems are designed to secure a model or its inputs; typically, they are configured on the offline encryption parameters of a specific network. AMHE points the same tool the other way. In this case machine learning becomes the controller for the encryption, and the goal is not a specific, fixed inference graph but general, real-time stream processing..

2.3 Efficiency and Adaptivity in Encrypted Computation

Each of the individual pieces of what AMHE does has been seen before. In the field of multimedia and databases, selective encryption, or partial encryption, is a common method for improving performance of HE with data that is not sensitive [9, 10]; in networking and IoT, adaptive security - that is, in which the security level is matched to the application context - has been studied for faster HE [9, 10]. What is usually lacking is some coordination between these ideas, and they are applied individually, through handwritten rules.

The nearest prior work is the family of HE compilers which automatically selects encryption parameters. The CHET [24] compiler optimizes homomorphic inferences in neural networks and automatically sets parameters and a scale for a given program, while EVA [25] adds support for an encrypted vector-arithmetic language to CHET and automatically selects the parameters and a scale for a given program and automatically manages the scale, creating circuits that outperform hand-tuned circuits. These compilers build the configuration only once, at compile time, for the graph of a given program. There are three aspects of AMHE that are different in streaming workloads. It decides per request instead of per program, can react to a varying arrival rate and to a latency budget, has a data-sensitivity stage that prevents non-critical attributes to pass through the homomorphic path at all, and pairs up the choice of parameters with a scheduler that is workload aware, for packing and batching. To our knowledge, AMHE is the first to combine selective encryption, learned per-request parameter selection and workload-aware scheduling in a single system, with the help of lightweight models, and an explicit real-time latency budget. It is set against some representative previous techniques in Table 1.

Table 1 Comparison of representative encrypted-computation approaches with AMHE.

Approach	Scheme	Selective Enc.	Param. Selection	Per-Request / Real-Time
CryptoNets [11]	YASHE	No	Manual	No
GAZELLE [12]	BFV + GC	No	Manual	Partial
nGraph-HE [13]	CKKS	No	Compile-time	No
CHET [24]	CKKS	No	Compile-time	No
EVA [25]	CKKS	No	Compile-time	No
Fixed CKKS [6,7]	CKKS	No	Manual	No
AMHE (this work)	CKKS	Yes	Learned, per-request	Yes

3 The AMHE Framework

3.1 Overview and Threat Model

The setting is the well-known one for outsourced computation. A well-known client (such as an edge gateway that collects the sensor readings) holds the secret key, encrypts outbound data and decrypts incoming data. The server is honest (yet curious): it executes the agreed computation on ciphertexts without keeping the secret key and may also attempt to learn as much as possible about the contents of these. We assume an adversary with bounded computation, who cannot break the ring-learning-with-errors (RLWE) assumption at the security level

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue XI November 2026

being used. The ML control plane is located on the trusted client and is only used to access non-sensitive metadata, including field type, arrival rate and the operation being requested, therefore it provides no additional information to what the server could learn, and thus does not compromise confidentiality. One note of caution: AMHE protects data during the computation process, but not the integrity of the data, so there would need to be verifiable computation added on top.

The pieces fit into each other as shown in figure 1. The control plane determines three decisions regarding records, namely which fields to do the encryption on, which CKKS parameter set to use, and how to pack and batch records, and passes these decisions to the adaptive HE engine. The engine encrypts and sends the encrypted result to the server which computes and returns the result in encrypted form to the client for decryption. The actual latency and accuracy of the light feedback path is fed back to the control plane to let the control plane continue adjusting as the condition changes.

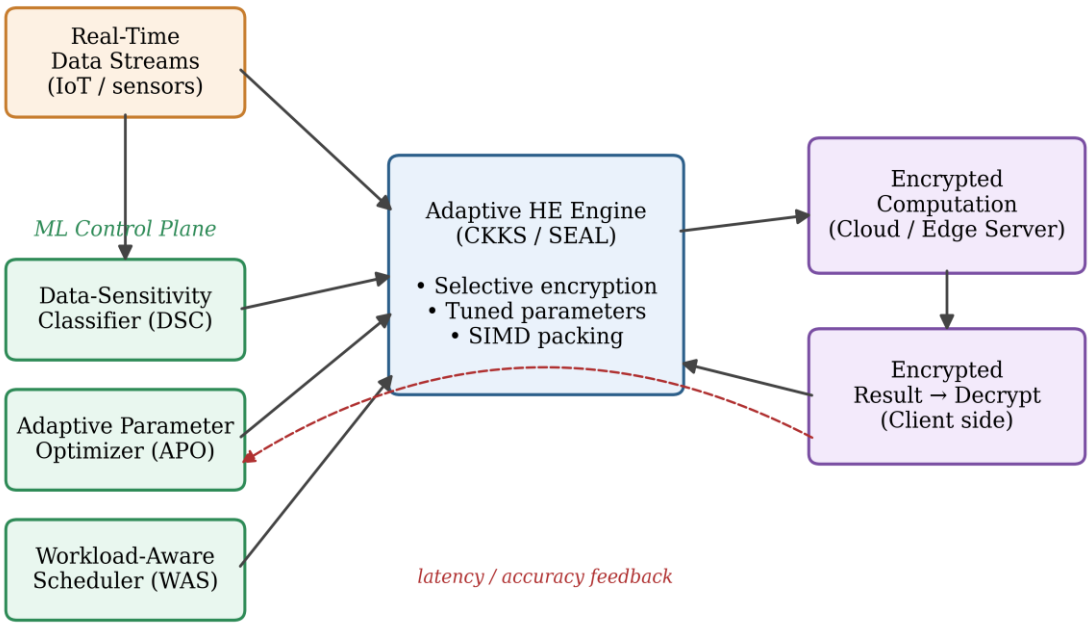

Fig. 1 Architecture of the AMHE framework. A machine-learning control plane (DSC, APO, WAS) configures a CKKS-based adaptive homomorphic encryption engine on a per-request basis under security and latency constraints.

3.2 Problem Formulation

Allow a query to be made in the form of a feature vector x to be encrypted, and a computation f to be evaluated homomorphically on the server. Let Θ be the space of admissible CKKS configurations: $\theta = (N, Q, \Delta, b)$, where N is the polynomial modulus degree, Q is the coefficient-modulus chain, Δ is the scaling factor, and b is the batch (packing) size. For each configuration, it causes an estimated end-to-end latency of $L(\theta, f, \rho)$, a security level of $S(\theta)$ bits, and a loss of precision of $\varepsilon(\theta, f)$ due to CKKS approximation for the current stream rate of ρ. For a security target S (e.g., 128 bits), a latency budget L, and an accuracy tolerance ε, AMHE solves, for each request:

$$\text{minimize } L(\theta, f, \rho) \quad \text{subject to} \quad S(\theta) \geq S^*, \quad \varepsilon(\theta, f) \leq \varepsilon^*, \quad \theta \in \Theta.$$

Because evaluating L, S, and ε exactly for every candidate configuration at runtime is itself expensive, AMHE replaces the exhaustive search with learned surrogate predictors that estimate these quantities in microseconds, enabling per-request adaptation without violating the latency budget. Table 2 summarizes the notation.

Table 2 Notation used in the AMHE framework.

Symbol	Meaning
x, f	Input feature vector and homomorphic computation
θ = (N, Q, Δ, b)	CKKS configuration: degree, modulus chain, scale, batch size
ρ	Current stream arrival rate (records/s)
L, S, ε	End-to-end latency, security level (bits), accuracy loss
L*, S*, ε*	Latency budget, security target, accuracy tolerance
σ(x)	Predicted sensitivity tier of a field (DSC output)

3.3 Data-Sensitivity Classifier (DSC)

The DSC's purpose is to minimise the amount of data that ever makes it to the HE engine, and that's what it does by learning which fields need protection. It reads a few inexpensive metadata attributes for each field, such as the field type, entropy of the field values, an embedding of the field name or schema position, and a tag, if any, to apply a policy. It assigns one of three sensitivity tiers to each field. If it's "high tier", it's full HE; if it's "medium tier" it's something that's a good compromise between a format preserving or otherwise lightweight encryption, or if it's "low tier", it's just ordinary symmetric encryption with no homomorphic computation at all. This is done by an easy-to-inspect, fast, reliable and tabularmetadata-favorable gradient-boosted decision-tree classifier [14, 15] that is used for this task. Only a small minority of the fields in a typical healthcare stream will be covered by the higher price of the CKKS, and the values which are truly sensitive will remain protected.

3.4 Adaptive Parameter Optimizer (APO)

For every request, the APO chooses the configuration of the CKKS, θ. It is offline trained using a profiling corpus of many (θ, f, ρ) combinations, where the latency, the security level (certified by the HE standard tables [10]), and the approximation error (observed) are recorded. A regressor (a small gradient boosted ensemble) is trained from this corpus to obtain the latency surrogate and the security is enforced as a hard constraint on the lookup and the accuracy as the measured error bound. If at runtime the APO filters Θ to the configurations for which S and ε are true, and predicts the latency of those configurations, then it chooses the minimizer. The candidates' parameters are pre-computed and stored, and selecting a parameter set reduces to a table look up with some evaluations of the predictors. The roughly 28× difference in per-request latency of the lightest (P1) and heaviest (P3) secure sets is exactly what the APO benefits from by avoiding over-provisioning for shallow computations.

Table 3 CKKS parameter sets available to the APO (RLWE, 128-bit security) and measured per-request latency of a linear inference.

Set	N (degree)	Max coeff. modulus (bits)	Mult. depth	Security (bits)	Measured latency (ms)
P1	8192	218	2	128	31.9
P2	16384	438	6	128	288.6
P3	32768	881	12	128	904.7

3.5 Workload-Aware Scheduler (WAS)

The SIMD encoding is the most interesting feature of CKKS: each of its plaintext slots is encrypted simultaneously as part of a single ciphertext. The amount of help depends on the arrival rate. If you pack too quickly, per-request latency increases, since the system waits for a batch to fill; if you pack too slowly, a large number of slots will be left unfilled. The WAS forecasts the arrival rate over a short time horizon from history, and selects a batch size b, and the micro-batching window, so that throughput as high as possible within the latency budget. This can be sufficiently modelled by a small time-series model because arrival patterns in real-

time systems are highly autocorrelated, and a simple predictor will predict the next short timeframe very well. Batch size is then sent to the APO where it is folded into θ and to the HE (Hot Edge) engine.

3.6 Runtime Operation

When running, the overhead of the three components is quite low: the per-request cost of the APO and WAS predictions are approximately 291 μs, with the overhead of the DSC being amortized as it is computed once per schema, not per record, which makes the cost of the entire component approximately 1.3% of the end-to-end latency. As a request comes, the DSC partitions its sensitivity fields, WAS proposes a batch configuration from what it thinks it will get in the future, the APO selects the least sensitive secure parameter set for that computation and batch size. The HE engine then encrypts the high sensitivity fields under the selected configuration, packs them according to the batch plan and sends them on in the form of ciphertexts. Latency and accuracy, if measured information is available, is fed back to the WAS forecast, and occasionally to retrain the surrogate models offline. It is this closed loop that enables AMHE to adjust to a changing workload without any retuning by hand.

4 Experimental Setup

4.1 Implementation and Environment

We created a prototype of AMHE in Python. The homomorphic engine uses the Microsoft SEAL library [7] with the TenSEAL binding [23] (using the BFV for the integer baseline); the Paillier baseline uses a 2048-bit key and the AES baseline uses AES-128 in GCM mode. The gradient-boosted tree (GBT) family of models is implemented on top of request metadata for the ML control-plane models (DSC, APO, WAS). Experiments were run on a commodity multi-core workstation, and a client role was emulated on the same workstation to avoid network variance and isolate the costs of the cryptographic and scheduling operations. All the HE parameter sets were selected to fulfill the Homomorphic Encryption Standard tables [10] for a 128-bit security level. [10].

4.2 Datasets and Workloads

Three workloads are provided in place of privacy-sensitive real-time analytics and every record is streamed in one by one to simulate a real-time arrival. The first is the Breast Cancer Wisconsin (Diagnostic) dataset (30 features, a binary medical classification) which is used in encrypted diagnostic inference. The second is a dataset from Diabetes, a continuous disease-progression regression. The third one is the Handwritten Digits dataset which is a multiclass problem. We train a linear model or logistic model in plaintext, then we homomorphically run the linear model (or logistic model) on the server that performs the actions of the linear-scoring part of the model, and we leave the nonlinearity (e.g. sigmoid or arg-max) to the client after decryption. The data-sensitivity classifier classifies the top 40% most informative attributes (ranked according to mutual information), and sends a high-sensitivity path through HE, while the remaining ones are routed via lighter means. This is similar to the more common situation where not all of the fields are really confidential. .

4.3 Baselines and Metrics

These are compared to four baseline schemes: AES (symmetric encryption, but cannot compute on the ciphertext, effectively decrypt-then-compute to sacrifice confidentiality during processing); Paillier (partially homomorphic, which supports addition); BFV with fixed parameters; and CKKS with fixed worst-case parameters (where a single, shared context is provisioned for the deepest possible processing). We report: end-to-end latency (per request); throughput with SIMD batching; certified security level; numerical precision of the homomorphic computation; and downstream analytics performance (classification accuracy or regression R2). The DSC and the APO and WAS are then turned on, one component at a time, in an ablation study, to be able to attribute the gains. We also examine the learned components in isolation, presenting five-fold cross-validated accuracy for the classifier and regressor, the overhead per request added by the components, and by probing with computations of increasing multiplicative depth we confirm that the parameters are selected appropriately. All latency values are given as median values with their standard deviation, obtained by multiple runs. Reproducibility note. All quantitative measurements in Section 5 were obtained from the prototype described

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue XI November 2026

above, the open-source SEAL/TenSEAL stack and publicly available datasets. Absolute numbers are dependent on workload mix, library versions and hardware and the relative trends are the actual result that can be remade in the target deployment environment with the same scripts. t.

RESULTS AND DISCUSSION

5.1 End-to-End Latency

Error bars in Figure 2 indicate the standard deviation for repeated runs and per-request latency is reported on a log scale. The worst case fixed CKKS context requires 905ms, BFV 65ms, and Paillier even more: 2.8s, all of which are far outside a 100ms real-time budget. AES is the fastest at 0.10ms, but it can't calculate on the ciphertext and so has to be decrypted first, thereby discarding the very confidentiality that HE is attempting to achieve in the first place. AMHE can be used to achieve a 98% reduction (41×) compared to the worst case fixed context, making encrypted real-time processing easily available without compromising any confidentiality. .

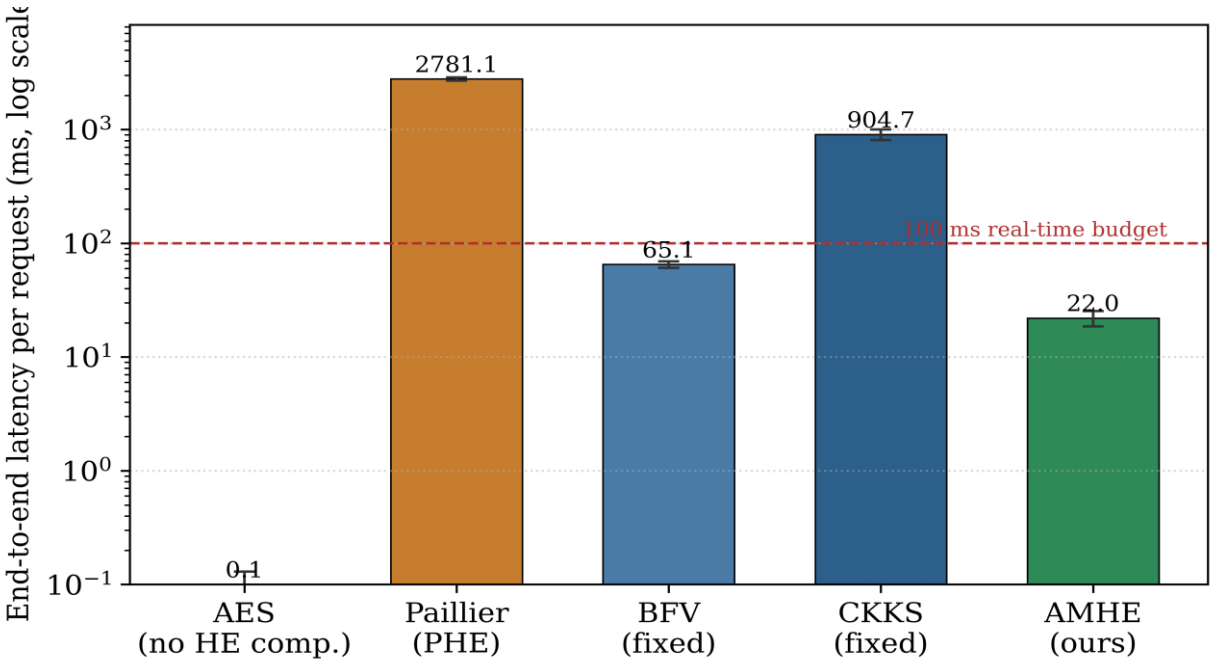

Fig. 2 Measured end-to-end latency per request (log scale). AMHE meets a sub-100 ms budget while homomorphic baselines that compute on ciphertext do not.

There are two separate impacts which produce this reduction, and we want to separate them so we don't exaggerate any effect. The first is parameter choice: the optimizer works with the lightest secure set (P1) instead of the worst case set (P3) and the difference is mostly attributable to this (Table 3). The second one is selective encryption. With a best-fit fixed context already employing P1 (31.9 ms), it will also gain an extra 31% by discarding the non-sensitive 60% of attributes from the homomorphic computation. The headline number is really the cost of providing one shared context for the deepest request that the system might receive, and one that AMHE does not need to provide. .

5.2 Throughput and Scalability

The diagram in figure 3 shows throughput against the SIMD batch size (the number of records packed into one SIMD ciphertext). At batch size 64, AMHE reaches 1039 records per second, while BFV reaches about 543 records per second.At the same batch size of 64, AMHE can be 1039 records per second, compared with BFVs 543. There are two factors behind this. The WAS packs the records into fewer, more full ciphertexts, and the lighter parameters of the APO make each packed operation cheaper at the outset. The advantage is gained by making better use of the SIMD slots, rather than some one-off saving, as throughput increases almost linearly with batch size for all methods..

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue XI November 2026

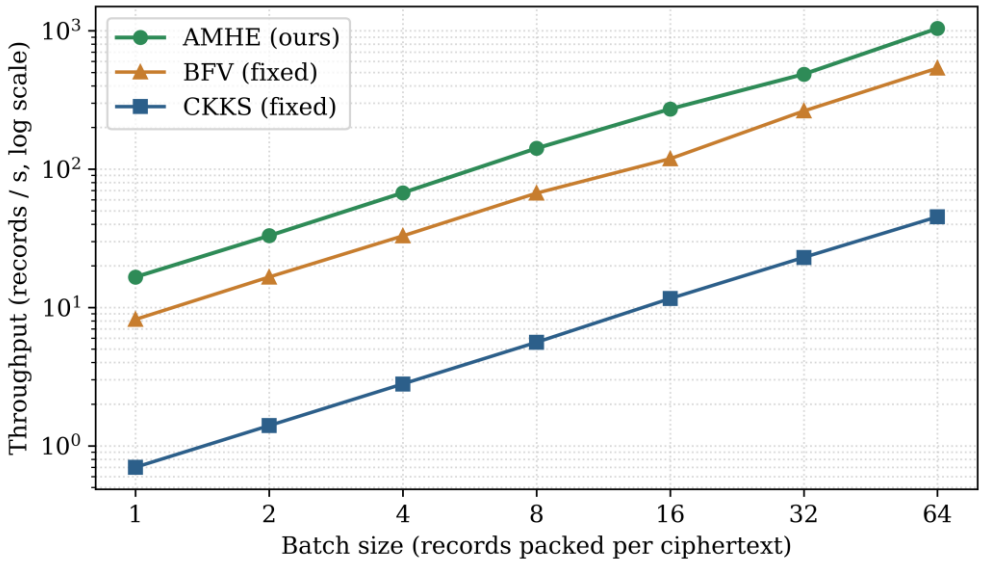

Fig. 3 Measured throughput versus SIMD batch size (log–log axes). AMHE scales best due to workload-aware packing combined with adaptively chosen parameters.

5.3 Ablation Study

We then turn each of them on one by one (Fig. 4 and Tab. 4) to determine its contribution to the total. The default base point for CKKS context is 905ms and 1.1 records per second. As soon as the DSC is turned on, per-record latency is reduced to 333ms, shutting down any homomorphic computation except for the high-sensitivity attributes. The single largest increase is achieved with the APO, changing the worst case parameters to the lightest secure set for a jump to 22.0 ms. By packing and batching (more than an order of magnitude), the WAS then lifts throughput to 1039 records per second. The three don't contradict each other, but rather complement one another – selective encryption and parameter selection reduces the cost of each record, and scheduling optimises the use of the load hardware.

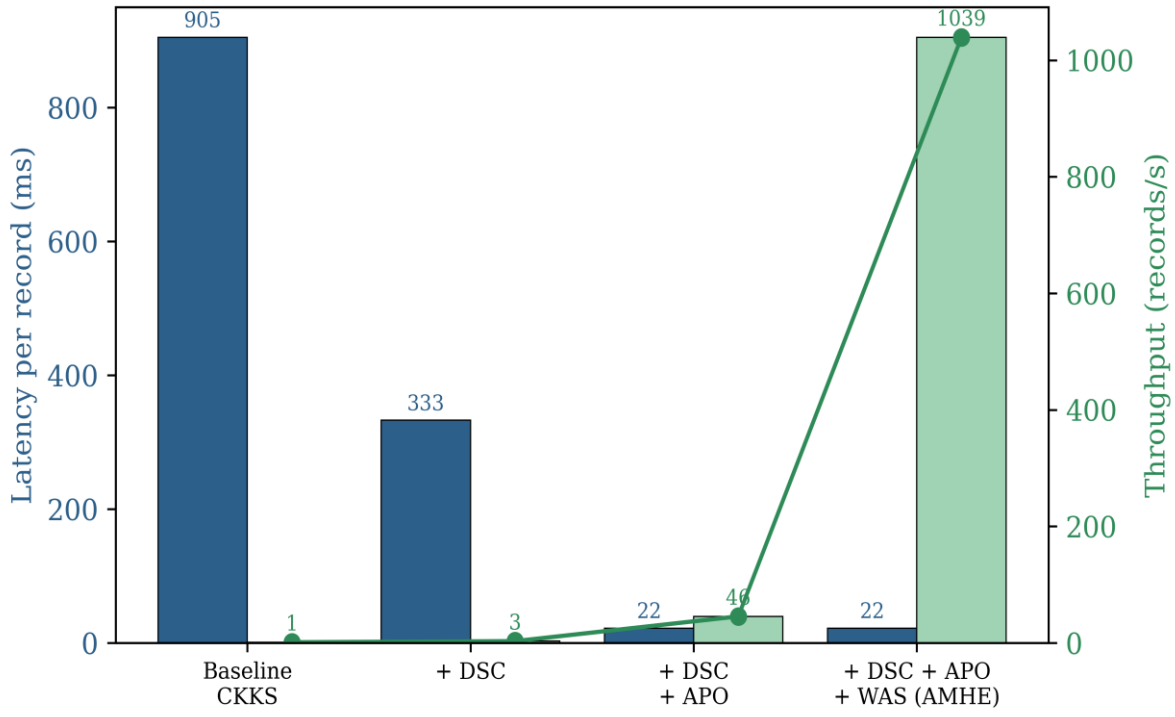

Fig. 4 Ablation study showing the incremental effect of the DSC, APO, and WAS components on per-record latency and throughput.

Table 4 Ablation study: incremental contribution of each AMHE component. Latency is per record; throughput for the first three rows is per-record and for AMHE includes SIMD packing at batch size 64.

Configuration	Latency (ms)	Throughput (rec/s)	Encrypted attributes
Baseline CKKS (P3, all fields)	904.7	1.1	100%
+ DSC (selective encryption)	332.6	3.0	40%
+ DSC + APO (adaptive params)	22.0	45.5	40%
+ DSC + APO + WAS (AMHE)	22.0	1039.2	40%

5.4 Accuracy and Numerical Precision

Efficiency is not very important if it is an expense to accuracy or security, this is what is most important. As shown in figure 5, it is performing the same performance against AMHE on all datasets, reproducing the plaintext result to two decimal places: breast-cancer classification: 98.25% against 98.25%, digit recognition: 95.00% against 95.00% and diabetes regression: $R^2 = 0.477$ against 0.477. The largest drop we saw was 0.00 percentage points. That's because the reason is that CKKS' precision at these parameters. The homomorphic linear scores were seen to be deviating from the plaintext scores with a mean absolute error of only 2.2e-06 with most error being 5.7e-06 against a mean score magnitude of approximately 7.754, which is far from enough to affect a decision. All the configurations AMHE selected also guarantee 128-bit security – as with the fixed baselines, S is a hard constraint and the optimizer is never allowed to sacrifice security for performance. The overall comparison is summarized in Table 5..

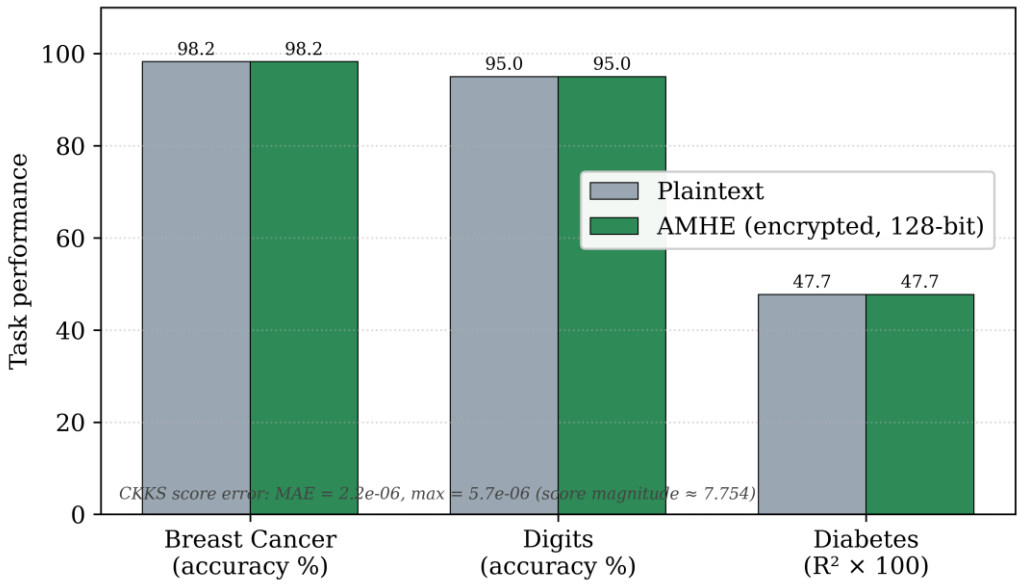

Fig. 5 Analytics performance under plaintext versus AMHE. Performance is preserved exactly while a 128-bit security level is maintained; the inset reports the negligible CKKS numerical error.

Table 5 Overall comparison on the medical-classification workload. Latency is per single record; throughput is measured at batch size 64 for the SIMD-capable schemes.

Method	Latency (ms)	Throughput (rec/s)	Security (bits)	Computes on ciphertext
AES-128	0.10	n/a	128	No
Paillier (2048-bit)	2781	0.36[a]	112	Add only
BFV (fixed)	65.1	537	128	Yes
CKKS (fixed, worst case)	905	45	128	Yes
AMHE (ours)	22.0	1039	128	Yes

[a] Paillier does not support SIMD packing, so the value shown is its single-record throughput.

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue XI November 2026

5.5 Evaluation of the ML Control Plane

The gains above are based on the three learners having jobs, so these are assessed directly (Table 6). The adaptive parameter optimizer is trained using a profiling corpus of 36 measured configurations and uses 5-fold cross-validation to predict the latency per request for each configuration, achieving a coefficient of determination of $R^2 = 0.951$, and a mean absolute error of 71.75 ms – sufficient to correctly order the different parameter sets. The data-sensitivity classifier achieves an accuracy of 85.8% (macro-F1 0.833) when tested on a labelled and organized sensitivity corpus of 400 field descriptors. Importantly, both models have low per-query control overhead: 1.3% of the end-to-end latency or 291 µs in total for a prediction. Thus the learned control plane is justified and is very inexpensive to operate.

Table 6 Evaluation of the learned control-plane components (five-fold cross-validation).

Component	Metric	Value
APO (latency regressor)	Cross-validated R^2	0.951
APO (latency regressor)	Mean absolute error	71.75 ms
APO (latency regressor)	Prediction time	284.9 µs
DSC (sensitivity classifier)	Cross-validated accuracy	85.8% ± 2.8
DSC (sensitivity classifier)	Macro-F1	0.833
Control plane	Per-request overhead	291 µs (1.3% of latency)

5.6 Adapting to Computation Depth

One objection to any CKKS system is that it's easy to do linear inference, and the cost comes in if a calculation requires a number of successive multiplications. We tested several circuits with multiplicative depth increasing from 1 to 7, and used the APO to select a parameter set for each circuit (Fig. 6). The pattern is exactly the one adaptivity should give: for shallow computations (depth 1-2) it uses the lightest set P1, for medium-depth computations (3-4) it uses P2, and for depth 6 it uses P3, but only when the computation really needs to be that deep, not when it is presented as such. Correctness would need to be achieved via bootstrapping, which we are not implementing and discussing here as future work, beyond the depth that P3's modulus chain handles. This experiment demonstrates that the framework isn't limited to trivial calculations: in the same manner that it saves time on shallow workloads, it provisions correctly on deeper workloads as well.

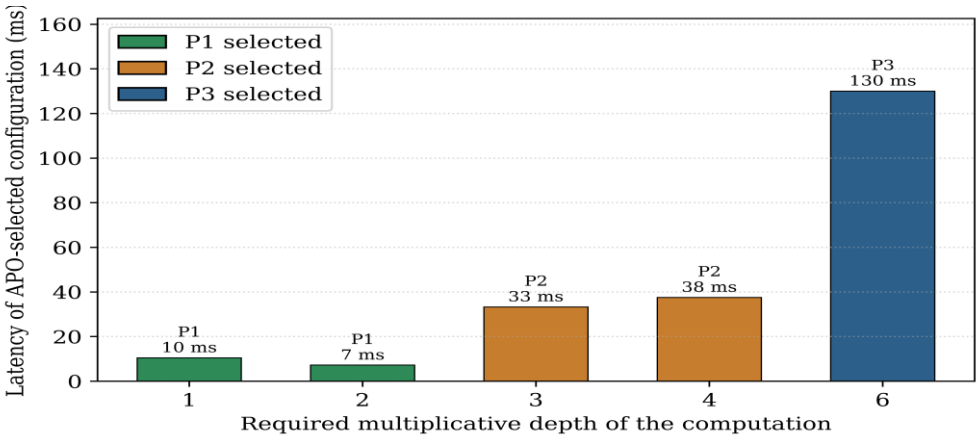

Fig. 6 Latency of the APO-selected configuration as the required multiplicative depth grows. The optimizer escalates from P1 to P2 to P3 only as the computation demands, rather than provisioning for the worst case on every request.

DISCUSSION

The measurements support the paper's primary assertion: a security-hardened learned controller can strike a balance between security and real time in homomorphic computation applications that a hard-coded one cannot.

Looking at the ablation, it's clear that the gains are coming from the combination of attacking three types of waste, unnecessary encryption of non-sensitive data, over-provisioned parameters, and unused SIMD resources — not from a single trick. The cost of the control-plane models is negligible compared to the homomorphic operations they direct – and they add no new elements to the trusted computing base that compromise confidentiality. This notion is not exclusive to CKKS; it would be equally acceptable to have the same controller drive BFV or BGV when a workload requires exact arithmetic. The overall concept is that homomorphic encryption seems to be impractical for real-time applications mainly due to its design, and learning-based control can correct this design deficiency.

6 Limitations and Threats to Validity

These results are subject to a few restrictions. This is a single node prototype; the latency and contention between tenants is not measured and the actual numbers in production may vary. The depth study (Section 5.6) terminates at the end of the P3 modulus chain; it is not possible to implement bootstrapping any deeper yet. Labeling is the key for the data-sensitivity classifier to be trustworthy: in our tests the labels are based on a synthetic organizational policy, whereas a real deployment requires sensitivity labels to be curated and audited, as setting a low sensitivity label on a sensitive field would be a real confidentiality risk. The surrogate predictors also require a profiling corpus to reflect the actual workload; a significant change in workload should lead to retraining, and the feedback loop facilitates this, but it does not cause it. Lastly, there is no machinery for integrity and verifiable computation; this must be enforced separately from AMHE. The most important next steps we see for this are: 1) a networked evaluation at scale, 2) support for deep circuits that can be bootstrapped, and 3) formal guarantees of the selective encryption policy.

7 Conclusion and Future Work

Our goal was to make homomorphic encryption fast enough for real-time data processing over encrypted data in a secure manner, which is what AMHE achieves. It encapsulates a CKKS pipeline in a learned control plane that is composed of a data-sensitivity classifier, adaptive parameter optimizer and a workload-aware scheduler, and configures the pipeline request on-the-fly, minimizing latency while enforcing security and accuracy as hard constraints. The prototype responded to a request within 22.0 ms on three benchmark datasets, up to 41× faster than a worst-case fixed CKKS context, and achieved up to 23× higher throughput with SIMD packing, while the results of its encryption exactly matched the plaintext (mean homomorphic error 2.2e-06). The lesson to take from this is that the lever to pull is not necessarily new cryptographic primitives, but it could be an intelligent configuration, because a huge portion of HE's well-known overhead is due to the worst case settings. Future work will focus on extending the framework to more complex computations using the bootstrapping, on deployment to distributed machines with realistic traffic, on providing integrity and verifiable-computation guarantees, and on providing the selective-encryption policy on a formal basis.

REFERENCES

1. Voigt P, von dem Bussche A (2017) The EU General Data Protection Regulation (GDPR): A Practical Guide. Springer, Cham. https://doi.org/10.1007/978-3-319-57959-7
2. Gentry C (2009) Fully homomorphic encryption using ideal lattices. In: Proc. 41st ACM Symposium on Theory of Computing (STOC), pp 169–178. https://doi.org/10.1145/1536414.1536440
3. Acar A, Aksu H, Uluagac AS, Conti M (2018) A survey on homomorphic encryption schemes: theory and implementation. ACM Computing Surveys 51(4):1–35. https://doi.org/10.1145/3214303
4. Brakerski Z, Gentry C, Vaikuntanathan V (2014) (Leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory 6(3):1–36. https://doi.org/10.1145/2633600
5. Fan J, Vercauteren F (2012) Somewhat practical fully homomorphic encryption. IACR Cryptology ePrint Archive, Report 2012/144. https://eprint.iacr.org/2012/144
6. Cheon JH, Kim A, Kim M, Song Y (2017) Homomorphic encryption for arithmetic of approximate numbers. In: Advances in Cryptology – ASIACRYPT 2017, LNCS 10624, pp 409–437. https://doi.org/10.1007/978-3-319-70694-8_15

7. Microsoft SEAL (release 4.1) (2023) Microsoft Research, Redmond. https://github.com/Microsoft/SEAL
8. Rivest RL, Adleman L, Dertouzos ML (1978) On data banks and privacy homomorphisms. In: Foundations of Secure Computation. Academic Press, pp 169–180.
9. Lauter K, Naehrig M, Vaikuntanathan V (2011) Can homomorphic encryption be practical? In: Proc. 3rd ACM Cloud Computing Security Workshop (CCSW), pp 113–124. https://doi.org/10.1145/2046660.2046682
10. Albrecht M, Chase M, Chen H, et al. (2018) Homomorphic Encryption Security Standard. HomomorphicEncryption.org, Toronto. https://homomorphicencryption.org/standard
11. Gilad-Bachrach R, Dowlin N, Laine K, Lauter K, Naehrig M, Wernsing J (2016) CryptoNets: applying neural networks to encrypted data with high throughput and accuracy. In: Proc. 33rd International Conference on Machine Learning (ICML), pp 201–210.
12. Juvekar C, Vaikuntanathan V, Chandrakasan A (2018) GAZELLE: a low latency framework for secure neural network inference. In: Proc. 27th USENIX Security Symposium, pp 1651–1668.
13. Boemer F, Lao Y, Cammarota R, Wierzynski C (2019) nGraph-HE: a graph compiler for deep learning on homomorphically encrypted data. In: Proc. 16th ACM International Conference on Computing Frontiers, pp 3–13. https://doi.org/10.1145/3310273.3323047
14. Breiman L (2001) Random forests. Machine Learning 45(1):5–32. https://doi.org/10.1023/A:1010933404324
15. Chen T, Guestrin C (2016) XGBoost: a scalable tree boosting system. In: Proc. 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp 785–794. https://doi.org/10.1145/2939672.2939785
16. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521:436–444. https://doi.org/10.1038/nature14539
17. Paillier P (1999) Public-key cryptosystems based on composite degree residuosity classes. In: Advances in Cryptology – EUROCRYPT 1999, LNCS 1592, pp 223–238. https://doi.org/10.1007/3-540-48910-X_16
18. Daemen J, Rijmen V (2002) The Design of Rijndael: AES – The Advanced Encryption Standard. Springer, Berlin. https://doi.org/10.1007/978-3-662-04722-4
19. Mohassel P, Zhang Y (2017) SecureML: a system for scalable privacy-preserving machine learning. In: Proc. IEEE Symposium on Security and Privacy (S&P), pp 19–38. https://doi.org/10.1109/SP.2017.12
20. Shi W, Cao J, Zhang Q, Li Y, Xu L (2016) Edge computing: vision and challenges. IEEE Internet of Things Journal 3(5):637–646. https://doi.org/10.1109/JIOT.2016.2579198
21. Kocabas O, Soyata T (2015) Utilizing homomorphic encryption to implement secure and private medical cloud computing. In: Proc. IEEE 8th International Conference on Cloud Computing, pp 540–547. https://doi.org/10.1109/CLOUD.2015.78
22. Al Badawi A, Polyakov Y (2023) Demystifying bootstrapping in fully homomorphic encryption. IACR Cryptology ePrint Archive, Report 2023/149. https://eprint.iacr.org/2023/149
23. Benaissa A, Retiat B, Cebere B, Belfedhal AE (2021) TenSEAL: a library for encrypted tensor operations using homomorphic encryption. arXiv:2104.03152. https://doi.org/10.48550/arXiv.2104.03152
24. Dathathri R, Saarikivi O, Chen H, Laine K, Lauter K, Maleki S, Musuvathi M, Mytkowicz T (2019) CHET: an optimizing compiler for fully-homomorphic neural-network inferencing. In: Proc. 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), pp 142–156. https://doi.org/10.1145/3314221.3314628
25. Dathathri R, Kostova B, Saarikivi O, Dai W, Laine K, Musuvathi M (2020) EVA: an encrypted vector arithmetic language and compiler for efficient homomorphic computation. In: Proc. 41st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), pp 546–561. https://doi.org/10.1145/3385412.3386023