Benchmarking CNN Components in EZKL: A Layer-Level Analysis for EVM-Compatible Deployment

George Chidera Akor[1], Love Allen Chijioke Ahakonye[2], Jae Min Lee[1], Dong-Seong Kim[1,*]
[1]Department of IT-Convergence Engineering, Kumoh National Institute of Technology, Gumi, South Korea
[2]ICT Convergence Research Center, Kumoh National Institute of Technology, Gumi, South Korea
*NSLab Co. Ltd., Gumi, South Korea
Email: {georgeakor, loveahakonye, ljmpaul, dskim}@kumoh.ac.kr

***Abstract*—Zero-knowledge machine learning (ZKML) enables verifiable inference on private data, but deploying convolutional neural networks (CNNs) in production remains constrained by a multi-dimensional tradeoff between proof-generation latency, bandwidth consumption, and computational complexity. Existing ZKML frameworks and engineering blogs provide qualitative heuristics, yet practitioners lack systematic, layer-level measurements to guide architecture design under these constraints. This work presents the first systematic, layer-level characterization of CNN component costs in EZKL, a Halo2-based ZKML framework targeting EVM-compatible blockchains. We profile 8 feasible layer types (activations, pooling, normalization, and linear) across two EZKL precision settings (scale 7 and 10), measuring proof-generation time, proof size, circuit complexity, and peak memory in 26 experiments. We reveal critical infrastructure requirements by documenting 10 additional experiments that exceeded hardware limits (Conv2d operations, LayerNorm, and ReLU-based composite CNNs requiring >125GB RAM). Contrary to conventional wisdom, we find that precision configuration has a negligible performance impact (1.002× ratio), and that system RAM, not GPU VRAM, is the primary bottleneck. We release an open-source profiling toolkit and a public dataset that enable practitioners to query expected costs for their architectures and constraints.**

***Index Terms*—Blockchain, Convolutional Neural Networks, EZKL, EVM, Halo2, Zero-Knowledge Machine Learning, zk-SNARK.**

I. Introduction

Zero-knowledge machine learning (ZKML) enables verifiable neural network inference while preserving privacy [1]. When instantiated as zk-SNARKs, these proofs enable deployment in smart contracts and resource-constrained verifiers [2]. Deploying verifiable CNNs requires navigating three constraints: latency (proof generation takes minutes [3]), bandwidth (proof sizes affect transmission costs), and computational complexity (circuit size determines memory requirements). Architects must select activation functions and downsampling strategies under tight resource budgets, but at present have little quantitative, layer-level guidance.

Existing work, including ZKML surveys, framework documentation, and engineering blog posts, primarily offers qualitative recommendations (e.g., "avoid MaxPooling," use polynomial activations) derived from scattered benchmarks and engineering intuition rather than systematic, layer-level profiling under production-realistic constraints [4]–[7]. ZKML surveys and on-chain overviews summarize available frameworks and use cases [4], [8], [9] while benchmarking efforts on ZK proofs for ML inference report end-to-end performance for full models and selected architectures [5], [7], [10]. However, prior evaluations rarely isolate individual CNN layers or systematically map the multi-dimensional cost space spanning proof time, proof size, circuit complexity, and memory under a concrete, production-oriented stack.

This paper provides systematic layer-level benchmarks for EZKL, a Halo2-based ZKML framework targeting EVM-compatible blockchains. Our contributions are:

1) **Layer-level measurements.** We benchmark eight viable CNN layer types in EZKL, covering activations, pooling, normalization, and linear operations, at standard resolution under two precision settings (scale 7 and 10), with an additional scaling study on dense layers. Across three composite CNN architectures, we conduct 26 feasible experiments to measure proof time, proof size, circuit complexity, and peak memory usage.
2) **Multi-objective analysis.** We characterize the multi-dimensional Pareto frontier spanning latency, bandwidth, and circuit complexity, and identify deployment regimes with distinct architectural optima: bandwidth-constrained IoT, latency-critical edge, and resource-limited verification nodes.
3) **Open benchmark.** We release an open-source profiling toolkit and a public dataset of configuration–cost pairs, enabling practitioners to query expected costs for their deployment constraints and serving as a basis for future ZKML-aware architecture search.

The remainder of this paper is organized as follows. Section II reviews background and related work in ZKML and on-chain verification. Section III describes our methodology and experimental design. Section IV presents single-layer and composite architecture results, design principles and deployment guidance. Section V discusses limitations and directions for future work.

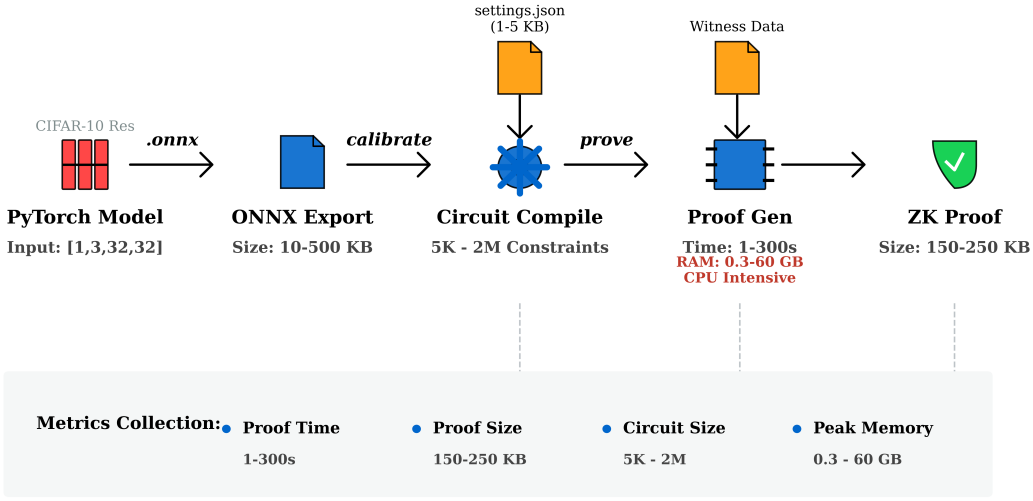

Fig. 1. EZKL CNN benchmarking pipeline with concrete data flow. Input: PyTorch models with CIFAR-10 resolution tensors (e.g., [1, 3, 32, 32] or [1, 64, 64] for activations). ONNX export produces 10–500KB model files. EZKL calibration generates settings.json (1–5KB) with optimized scales. Circuit compilation yields constraints ranging 5K–2M. Proving key generation consumes 0.3–60GB RAM. Final outputs: proof files (150–250KB), circuit size, proof time (1–300s), and peak memory metrics.

II. Background & Related Work

Zero-knowledge proofs enable proving the correct execution of $f(x) = y$ without revealing inputs or intermediate values [11]. Neural networks are compiled to arithmetic circuits over $\mathbb{F}_p$; linear operations map naturally to field arithmetic while non-linear operations require expensive constraint gadgets [4]. Proving time typically scales as $O(|C| \log |C|)$ in constraints $|C|$ [2]. We focus on EZKL [12], an open-source ZKML framework that compiles neural networks to Halo2-based zk-SNARK circuits and has production deployments on EVM-compatible blockchains [12], [13].

Prior ZKML benchmarking spans framework comparisons [14], which evaluated six ZK systems on MLPs; specialised protocols, including zkCNN's sumcheck optimisations (88.3s for VGG16, 1,264× speedup) [10], Mystique's arithmetic-Boolean conversions [5], and pvCNN's QMP-based 13.9× acceleration [15]; and end-to-end benchmarks such as ZKML (52.9s for ResNet-18) [6] and ZKTorch (6× speedup on MLPerf Edge) [16]. However, all prior work benchmarks complete models rather than isolating layer-level costs. No existing study profiles individual CNN layers within a production framework across proof time, proof size, circuit complexity, and peak memory; the granularity practitioners require for architecture selection under resource constraints.

III. Methodology

All experiments use an Intel i9-10940X CPU (14 cores, 28 threads) with 125GB RAM and three NVIDIA RTX 3090 GPUs (24GB VRAM each) running Ubuntu 22.04. Despite GPU availability, EZKL proof generation is CPU and RAM-intensive, with GPU utilization below 5%. This finding suggests practitioners should prioritize RAM (128+ GB) and multi-core CPUs over GPU hardware. The software stack consists of EZKL v23.0.3 [12] and PyTorch v2.5.1. All scripts and configuration files are publicly available. Figure 1 illustrates the end-to-end benchmarking pipeline. Each CNN layer flows through PyTorch definition, ONNX export (typically 10–500KB), EZKL circuit generation (producing settings.json with calibrated scales), proving key generation (1–60GB RAM), and proof generation with comprehensive metrics collection.

We profile 8 layer types: activations (ReLU, SiLU, Tanh, polynomial), pooling (MaxPool2d, AvgPool2d), normalization (BatchNorm2d), and linear (Dense). Input shapes match CIFAR-10 resolution (32×32×3 or variants). Combining these with 2 precision configurations yields 16 core experiments, supplemented by 4 Dense scaling experiments and 6 composite architecture runs, for a total of 26 feasible experiments. An additional 10 experiments (Conv2d variants, LayerNorm, CNN-ReLU composite) exceeded 125GB RAM during key generation.

Each layer is wrapped in a minimal PyTorch module, exported to ONNX, and compiled via EZKL Python bindings with automatic calibration. We evaluate two precision configurations: scale 10 (accuracy mode, high precision) and scale 7 (efficiency mode, lower precision), which control EZKL's `input_scale` and `param_scale` parameters that affect the fixed-point representation. We measure proof generation time, proof size, circuit size (constraints), and peak memory usage. For composite architectures, we train small CNNs on CIFAR-10 (50 epochs, Adam, lr=0.001) to evaluate end-to-end costs. We test four architectures:

- **CNN-ReLU**: Conv – ReLU – MaxPool – Conv – ReLU – MaxPool – Dense
- **CNN-Poly**: Conv – polynomial activation – AvgPool – Conv – polynomial activation – AvgPool – Dense

- **CNN-Mixed**: Conv – polynomial activation – AvgPool – Conv – ReLU – AvgPool – Dense
- **CNN-Strided**: Conv (stride 2) – ReLU – Conv (stride 2) – ReLU – Dense

Each architecture is trained on CIFAR-10 for 50 epochs using Adam optimizer (learning rate 0.001, weight decay 0.0001), then compiled through EZKL to obtain end-to-end proof-generation time, proof size, circuit size, peak memory, and model accuracy.

IV. Results and Discussion

We present empirical findings across 26 experiments, organized into the following categories: precision impact, layer-level performance, scaling behavior, composite architectures, deployment guidance, and infrastructure insights.

A. Precision Impact: A Counter-Intuitive Finding

A central finding of this work is that EZKL's precision configuration (`input_scale` and `param_scale`) has *negligible impact* on proof generation performance across all tested layer types. This contradicts conventional ZKML wisdom that higher precision incurs substantial overhead. Table I presents the measured ratios between high-precision (scale 10) and low-precision (scale 7) configurations.

TABLE I
PRECISION IMPACT ON PERFORMANCE METRICS (SCALE 10 VS SCALE 7 RATIO)

Metric	Mean Ratio	Std Dev	Interpretation
Proof generation time	1.002×	0.08	Negligible (0.2% increase)
Proof size	1.000×	0.00	Identical
Circuit size (constraints)	1.000×	0.00	Identical
Peak memory usage	1.001×	0.05	Negligible

This result contradicts conventional ZKML wisdom that higher precision incurs substantial performance penalties. We attribute this to EZKL's automatic calibration phase, which adjusts internal scales to minimize circuit complexity while maintaining numerical stability. After calibration, both precision modes converge to similar actual scale values, resulting in nearly identical circuits. This finding has immediate practical implications: practitioners can use strict precision settings (scale 10) to ensure high numerical accuracy *without sacrificing efficiency*, eliminating a traditional accuracy-vs-performance tradeoff.

Figure 2 visualizes this counter-intuitive result, comparing proof time, proof size, and circuit complexity across precision settings. The near-identical bars demonstrate that EZKL's calibration effectively neutralizes the trade-off between precision and performance.

B. Layer-Level Performance Characteristics

Having established that precision settings are essentially cost-free, we now examine the performance characteristics of individual layer types. Table II presents detailed metrics for eight feasible CNN layers, averaged across both precision settings. Layers are ordered by proof generation time from fastest to slowest, revealing a 146× performance range.

TABLE II
LAYER-LEVEL PERFORMANCE CHARACTERISTICS (AVERAGED ACROSS SCALE 7 AND 10)

Layer	Proof time (s)	Proof size (KB)	Constraints	Peak RAM (GB)
Fast (<10s): Activation Functions & Linear				
Dense	1.78	74.9	19,072	0.8
Polynomial	4.66	1,257.0	65,536	1.2
Tanh	5.06	1,248.0	65,536	1.2
SiLU	5.16	1,265.9	67,584	1.3
ReLU	5.19	1,216.7	96,256	1.5
Medium (10–100s): Pooling				
AvgPool2d	82.71	24,576.4	1,884,162	26.4
Slow (>100s): Normalization & Pooling				
BatchNorm2d	137.15	39,057.3	2,097,152	42.1
MaxPool2d	260.80	39,059.5	2,310,144	51.8
Excluded (OOM): Conv2d variants & LayerNorm				
Conv2d (stride 1, 2), DepthwiseConv2d, LayerNorm: >125 GB RAM				

Several trends emerge: (1) Activation functions are uniformly fast (1.8–5.2s), with Dense layers being the most efficient, and polynomial activations slightly outperforming ReLU. (2) Pooling operations span a wide performance range, with AvgPool2d being 3.2× faster than MaxPool2d (82.7s vs 260.8s). (3) BatchNorm2d incurs moderate overhead (137.1s), making it viable for applications with relaxed latency budgets. (4) Proof sizes vary dramatically (75 KB to 39 MB), driven primarily by circuit complexity rather than operation type. (5) Memory consumption is the limiting factor for convolutional layers, with all Conv2d variants exceeding our 125 GiB system capacity.

C. Scaling Study: Sub-Linear Cost Growth

The layer-level measurements raise an essential question: how do these costs scale with model size? To answer this, we evaluated Dense layers in two configurations: Small (5,440 constraints) and Large (272,896 constraints), a 50× increase in circuit complexity. Table III summarizes the results.

TABLE III
DENSE LAYER SCALING BEHAVIOR

Configuration	Constraints	Proof time (s)	Peak RAM (GB)
Dense-Small	5,440	1.09	0.3
Dense-Large	272,896	16.76	4.1
Ratio	50.1×	15.4×	13.7×

Proof time scales *sub-linearly* with circuit size: a 50× increase in constraints yields only a 15.4× increase in proof time. Across all 26 experiments, power-law fit yields exponent 0.89 ± 0.03 (R^2=0.975, $p<10^{-20}$), confirming sub-linear scaling. This favorable behavior suggests larger models may be more efficient per parameter. Memory consumption exhibits similar sub-linear scaling (13.7×).

Figure 3 presents the scaling behavior across all 26 experiments on a log-log plot. The power-law fit reveals sub-linear scaling (exponent 0.89 ± 0.03, R^2=0.975), indicating that EZKL proof generation benefits from economies of scale as circuit complexity increases.

D. Attribution of Performance Characteristics

An important methodological considteration is whether our benchmarked costs reflect properties of the EZKL Library or

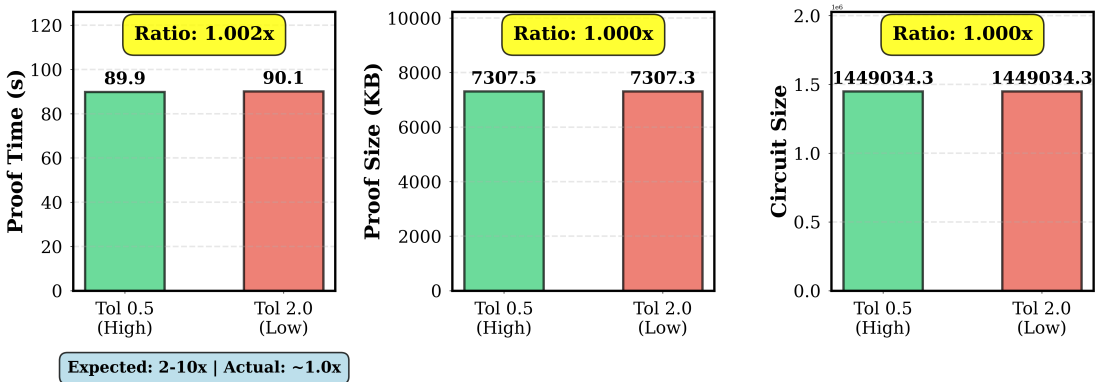

Fig. 2. Precision impact comparison showing negligible performance differences between scale 10 (high precision) and scale 7 (low precision). The 1.002× ratio contradicts expectations of substantial overhead for higher precision.

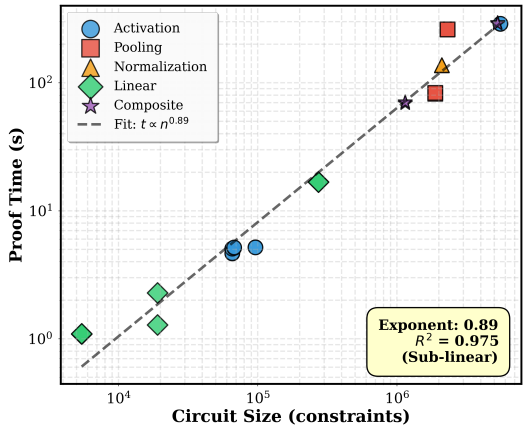

Fig. 3. Scaling behavior across all experiments showing sub-linear relationship (exponent 0.89, R^2=0.975) between circuit size and proof time. Different markers denote layer categories; the dashed line shows a power-law fit.

artifacts of Pytorch. We argue that the measured performance characteristics are predominantly attributable to EZKl and the underlying Halo2 proving system, as EZKL's architecture sets a clear separation between model definition and proof generation. Pytorch is only involved in the initial model definition and ONNX export phases, which contribute negligibly to the total benchmarked time. The Python bindings serve merely as an interface to invoke EZKL's compiled Rust routines; the computationally intensive operations (polynomial commitment, constraint satisfaction, and proof construction) execute entirely within the native Halo2 prover.

Nevertheless, we acknowledge two potential sources of framework influence:

1) ONNX export fidelity: Different frameworks may produce subtly different ONNX graphs for equivalent operations, potentially affecting circuit structure; and
2) Quantisation calibration: the sample inputs provided during EZKL's calibration phase derive from PyTorch tensors, which could marginally affect the optimised scale parameters.

Still, we consider these effects to be second-order and unlikely to alter our primary findings, though future work could validate this by comparing ONNX exports from TensorFlow or other frameworks.

E. Composite Architecture Comparison

With layer-level costs and scaling behavior characterized, we now examine how these components compose in end-to-end CNN architectures. Table IV presents results for three successfully evaluated composite architectures on CIFAR-10, alongside one failed architecture that exceeded our hardware capacity. CNN-ReLU could not be assessed due to out-of-memory errors during proving key generation, even on our 125 GiB RAM system, indicating that ReLU-based composite architectures require substantially more memory than alternatives using polynomial or mixed activations.

TABLE IV
COMPOSITE CNN ARCHITECTURES ON CIFAR-10 (AVERAGED ACROSS 2 PRECISION SETTINGS)

Architecture	Proof time (s)	Proof size (KB)	Constraints	Peak memory (GB)
CNN-Strided	69.8	478.9	1,144,660	11.2
CNN-Mixed	291.1	478.7	5,277,528	58.3
CNN-Poly	288.8	478.9	5,531,480	59.1
CNN-ReLU	OOM error (4 failed attempts; requires >125 GB RAM)			

CNN-Strided emerges as the clear winner for latency-critical applications, achieving 4.2× faster proof generation than CNN-Mixed and CNN-Poly while requiring only 11.2 GiB peak memory. This architecture eliminates pooling layers in favor of strided convolutions, resulting in substantially smaller circuit complexity (1.14M vs 5.3M constraints). CNN-Mixed and CNN-Poly exhibit similar performance characteristics, with proof times near 290 seconds and peak memory usage around 58 GiB. Both architectures prioritize ZK-friendly operations (polynomial activations, average pooling) but generate significantly larger circuits than CNN-Strided. The composite models exhibit a 3.4× overhead compared to the average of isolated core layers (216.6s vs 62.8s), demonstrating that layer-level costs compose approximately linearly in end-to-end architectures.

F. The Conv2d Paradox

Isolated Conv2d layers (32×32 input) consistently exceeded 125GB RAM during proving key generation, yet composite CNNs containing convolutional layers successfully generated proofs with <60GB peak memory. This discrepancy arises from three factors: (1) isolated layers are wrapped in identity I/O circuits for testing, creating larger end-to-end circuits; (2) composite architectures downsample inputs progressively: CNN-Strided uses stride-2 convolutions (effective 16×16 resolution), while CNN-Mixed and CNN-Poly use AvgPool after the first conv, reducing to 16×16 spatial dimensions; and (3) EZKL's compiler applies more aggressive optimizations to multi-layer graphs than isolated operations. This finding implies Conv2d is viable at ≤16×16 resolution on 125GB systems; practitioners should use strided convolutions or early pooling to manage memory.

G. Deployment Regime Recommendations

Based on our measurements, we provide concrete architectural guidance for three deployment regimes prioritizing different constraints. Table V summarizes recommendations.

Bandwidth-limited regime. For IoT and mobile scenarios where network transmission cost dominates, Dense layers and polynomial activations offer the best proof size efficiency (75–1,300 KB). BatchNorm2d should be avoided due to its 39 MB proof size, while AvgPool2d (24.6 MB) may be acceptable for moderate bandwidth budgets.

Latency-critical regime. Real-time applications (autonomous vehicles, robotics) should prioritize fast layers: Dense (1.78s), activation functions (4.7–5.2s), and the CNN-Strided architecture (69.8s). Critically, MaxPool2d (260.8s) and BatchNorm2d (137.1s) are prohibitively slow for interactive workloads.

Memory-constrained regime. Embedded systems with limited RAM should use activation-only or small Dense-layer models, avoiding pooling (26–52 GB) and normalization (42 GB). Conv2d is currently infeasible on commodity hardware (>125 GB).

Beyond these regimes, we anticipate that the dataset will enable automated model search, with the cost model incorporating ZK-specific metrics, complementing existing Neural Architecture Search (NAS) methods [17], [18].

H. Lessons Learned: Infrastructure & Design Principles

Our systematic evaluation reveals several counterintuitive findings and practical insights:

a) System RAM, not GPU VRAM, is the bottleneck.: Despite using three NVIDIA RTX 3090 GPUs (24 GiB VRAM each), GPU utilization remained below 5% across all experiments, with VRAM consumption under 1 GiB. Proof generation in EZKL is CPU- and system-RAM-intensive, with CPU usage reaching 1,500–1,600% (15–16 cores) and RAM consumption up to 60 GiB per experiment. This finding has direct implications for infrastructure planning: practitioners should prioritize high-capacity system RAM (128+ GB) and multi-core CPUs over expensive GPU hardware. Cloud deployments should favor memory-optimized instances (e.g., AWS r6i family) rather than GPU instances (p3/p4).

b) Precision is "free.": The negligible performance difference between scale 7 and scale 10 (1.002× ratio) contradicts common assumptions in ZKML that higher precision incurs steep costs. This enables practitioners to use strict accuracy requirements without performance penalties, eliminating a traditional tradeoff.

c) Pooling-free architectures win for latency.: CNN-Strided (69.8s) outperforms CNN-Mixed and CNN-Poly (290s) by 4.2× through eliminating pooling layers. Strided convolutions provide downsampling without the circuit complexity of MaxPool2d (260.8s) or AvgPool2d (82.7s).

d) ReLU composite architectures are problematic.: While standalone ReLU layers work well (5.19s), CNN-ReLU composite architectures failed due to OOM errors. Polynomial and mixed-activation architectures (CNN-Poly, CNN-Mixed) successfully generate proofs, suggesting activation function choice significantly impacts memory requirements in multi-layer contexts.

e) Sub-linear scaling offers hope.: The observed 0.89 scaling exponent indicates that proof time grows slower than circuit size, making moderately-sized models (1–5M constraints) more viable than naive linear extrapolation would suggest.

I. Limitations

Several limitations of this study are worth highlighting:

- **Single framework.** Our measurements focus on EZKL and Halo2; results may not directly generalize to other proving systems or ZKML stacks (e.g., Circom/Groth16, Risc0, Plonky2) [13].
- **On-chain gas measurement.** While our infrastructure generates Solidity verifier contracts suitable for EVM deployment, we encountered Solidity compiler stack depth limitations when deploying verifiers for complex circuits. This is a known constraint of zk-SNARK verifiers on current EVM implementations [12]. Future work may explore gas estimation using alternative compilation strategies (e.g., `--via-ir`) or simplified circuit subsets.
- **Model scale.** We target CNNs with parameter counts suitable for edge and on-chain inference. Extending to large-scale architectures (e.g., ResNet-50 or transformers) will likely require recursive proving or folding schemes [19].
- **Hardware configuration.** We evaluate a specific GPU instance; results may vary for larger prover clusters or specialized ZK accelerators.
- **Reproducibility.** Our experiments are run on a single high-end workstation with an i9-10940X CPU and RTX 3090 GPU. While this configuration may not be identical to all deployment environments, we mitigate this by releasing all scripts, configuration files, and measurement pipelines so that results can be replicated on comparable hardware. and by reporting mean ± standard deviation over multiple runs for each configuration.

TABLE V
ARCHITECTURE SELECTION GUIDE BY DEPLOYMENT CONSTRAINT

Constraint priority	Recommended layers / patterns	Cost characteristics	Use cases
Bandwidth-limited (IoT)	Dense, polynomial activations, AvgPool2d; avoid BatchNorm2d	Proof size: 75–1,300 KB; Time: 1–83s	Wireless sensors, satellite links, mobile edge
Latency-critical (edge)	Dense, SiLU/ReLU/Poly activations, CNN-Strided architecture	Time: 1–70s; Size: 75–479 KB; Constraints <1.2M	Real-time vision (vehicles, robotics, AR/VR)
Memory-constrained	Activation-only models, small Dense layers; avoid pooling & normalization	Peak RAM: 0.3–1.5 GB; Constraints: 5K–96K	Embedded systems, resource-limited nodes

V. Conclusion & Future Work

This paper presents a systematic benchmark of CNN components in EZKL for verifiable inference on EVM-compatible blockchains. We provide quantitative, layer-level measurements to navigate tradeoffs between proof latency, bandwidth, circuit complexity, and memory in edge deployments. Our study profiles 8 feasible layer types across two precision settings (scales 7 and 10), including scaling studies and composite architectures, and captures proof time, proof size, circuit size, and peak memory across 26 successful experiments. Ten additional configurations exceeded hardware limits: Conv2d and ReLU-based composite models require more than 125 GiB RAM, which is a barrier for commodity hardware.

Key findings are that (1) precision configuration has a negligible impact (1.002× time ratio), (2) RAM, not GPU, is the main bottleneck, (3) scaling is sub-linear (exponent 0.89 ± 0.03), and (4) pooling-free architectures reach about 4× speedup. We release an open source toolkit and dataset for cost queries and ZKML-aware architecture search, and outline future work on larger models with recursive proofs, transformer components for verifiable language models, and comparisons with alternative ZKML stacks.

Acknowledgment

This work was partly supported by the Innovative Human Resource Development for Local Intellectualization program through the IITP grant funded by the Korea government (MSIT) (IITP-2025-RS-2020-II201612, 25%), by the Priority Research Centers Program through the NRF funded by the MEST (2018R1A6A1A03024003, 25%), and by the MSIT, Korea, under the ITRC support program (IITP-2025-RS-2024-00438430, 25%) and by the Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education (RS-2025-25431637, 25%).

References

[1] M. B. Aziz, A. S. Naushad, M. Siddiqui, and J. A. Shamsi, "Zkvml: Zero-knowledge verifiable machine learning," in *International Conference on Asia Pacific Advanced Network*. Springer, 2024, pp. 220–239.

[2] J. Groth, "On the size of pairing-based non-interactive arguments," in *Advances in Cryptology – EUROCRYPT 2016*, ser. Lecture Notes in Computer Science, vol. 9616. Springer, 2016, pp. 305–326.

[3] zkonduit, "Benchmarking ezkl and zkml frameworks," 2023.

[4] A. Sathe, V. Saxena, P. A. Bharadwaj, and S. Sandosh, "State of the art in zero-knowledge machine learning: A comprehensive survey," in *International Conference on Advancements in Smart Computing and Information Security (ASCIS)*. Springer, 2023, pp. 98–110.

[5] C. Weng, K. Yang, X. Xie, J. Katz, and X. Wang, "Mystique: Efficient conversions for zero-knowledge proofs with applications to machine learning," in *Proceedings of the 30th USENIX Security Symposium*. USENIX Association, 2021, pp. 501–518.

[6] B.-J. Chen, S. Waiwitlikhit, I. Stoica, and D. Kang, "Zkml: An optimizing system for ml inference in zero-knowledge proofs," in *Proceedings of the 19th European Conference on Computer Systems (EuroSys 2024)*. Association for Computing Machinery, 2024, pp. 560–574.

[7] M. Hao, H. Chen, H. Li, C. Weng, Y. Zhang, H. Yang, and T. Zhang, "Scalable zero-knowledge proofs for non-linear functions in machine learning," in *Proceedings of the 33rd USENIX Security Symposium*. USENIX Association, 2024, pp. 3819–3836.

[8] V. Keršič, S. Karakatič, and M. Turkanović, "On-chain zero-knowledge machine learning: An overview and comparison," *Journal of King Saud University – Computer and Information Sciences*, vol. 36, no. 9, p. 102207, 2024.

[9] Y. Peng *et al.*, "A survey of zero-knowledge proof based verifiable machine learning," *arXiv preprint*, vol. arXiv:2502.18535, 2025.

[10] T. Liu, X. Xue, and Y. Zhang, "zkcnn: Zero knowledge proofs for convolutional neural network predictions and accuracy," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery, 2021, pp. 2968–2985.

[11] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM Journal on Computing*, vol. 18, no. 1, pp. 186–208, 1989.

[12] zkonduit, "Ezkl documentation."

[13] F. H. Soureshjani, M. Hall-Andersen, M. Jahanara, J. Kam, J. Gorzny, and M. Ahmadvand, "Automated analysis of halo2 circuits," in *Proceedings of the 21st International Workshop on Satisfiability Modulo Theories (SMT 2023)*. CEUR-WS.org, 2023, pp. 3–17.

[14] Modulus Labs, "The cost of intelligence: Proving machine learning inference with zero-knowledge," 2023.

[15] J. Weng, J. Weng, G. Tang, A. Yang, M. Li, and J.-N. Liu, "pvcnn: Privacy-preserving and verifiable convolutional neural network testing," *Trans. Info. For. Sec.*, vol. 18, p. 2218–2233, Jan. 2023. [Online]. Available: https://doi.org/10.1109/TIFS.2023.3262932

[16] B.-J. Chen, L. Tang, and D. Kang, "Zktorch: Compiling ml inference to zero-knowledge proofs via parallel proof accumulation," 2025. [Online]. Available: https://arxiv.org/abs/2507.07031

[17] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search," in *Automated Machine Learning: Methods, Systems, Challenges*, F. Hutter, L. Kotthoff, and J. Vanschoren, Eds. Springer, 2019, pp. 63–77.

[18] M. Wistuba, A. Rawat, and T. P. S., "A survey on neural architecture search," *arXiv preprint*, vol. arXiv:1905.01392, 2019.

[19] S. Bowe, J. Grigg, and D. Hopwood, "Halo: Recursive proof composition without a trusted setup," in *IACR Cryptology ePrint Archive*, vol. 2019/1021, 2019.

[1] `https://github.com/Jupiter-Plantagenet/ezkl-cnn-benchmark`