zkMPNN: Query-Local Zero-Knowledge Proofs of Complete Message-Passing Inference

Anonymous authors
Paper under double-blind review

Abstract

Message passing neural networks (MPNNs) are a powerful class of deep learning models for graph-structured data. To our knowledge, no existing approaches to verifiable machine learning establish both the correctness of the computation and the completeness of the required neighbors and intermediate states for a prediction. This paper formalizes this requirement as complete GNN verification and proposes zkMPNN, a modular framework for complete, query-local graph neural network (GNN) inference over committed models and graphs using zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs). Our evaluation demonstrates end-to-end proving for all 48 selected cases across four GNN families on node-classification and link-prediction tasks. Compressed proofs are 14.2–18.4 KiB and verify in 3.2–11.1 s with the public parameters and verifier key available. Controlled synthetic experiments show that proving costs depend jointly on query dependencies and authentication-circuit shape, rather than total graph size alone.

1 Introduction

Graph-based predictions are used in a wide range of applications, including fraud detection (Cheng et al., 2025) and item recommendation (Ying et al., 2018). In these tasks, graph neural networks (GNNs) exploit relationships among entities, such as accounts, transactions, and items, to make accurate predictions. Message-passing neural networks (MPNNs) are a widely used class of GNNs (Gilmer et al., 2017). In an MPNN, each node updates its state at each layer by aggregating messages from its neighbors. A prediction can depend not only on information associated with the queried node itself, but also on neighboring nodes and relationships in the graph.

Published cloud workflows demonstrate that graph-based classifiers can be served remotely to third parties (Shen et al., 2022). When GNNs are provided in such services, some of the graph information required for prediction may not be owned by the querying party, while the model itself may constitute valuable proprietary information that the service provider does not wish to disclose. A customer, auditor, or business partner may need to verify that a returned prediction was actually produced by the agreed model on the agreed graph snapshot. Recomputing the prediction would require the operator's parameters and other customers' records; publishing the model discloses proprietary parameters, while publishing the graph can expose identities (Narayanan & Shmatikov, 2009). Even prediction-only access can permit model extraction (Oliynyk et al., 2023).

This motivates a mechanism for verifying predictions without revealing the underlying model or graph data. Zero-knowledge proofs (ZKPs) (Goldwasser et al., 1985; Groth, 2016) offer a natural approach to this problem: the operator can prove that the prediction was computed using the agreed model and graph snapshot, while the verifier checks the result without gaining access to the private model parameters or graph data.

For a GNN query over a graph, proving the arithmetic correctness of the supplied inputs via ZKPs is not sufficient. A prover could otherwise select a favorable subgraph and produce a valid arithmetic proof for a result that differs from full-graph inference. The required guarantee is *complete GNN verification*: the query result must be derived from all dependencies required by the committed graph and model and match full-graph inference under the same numerical semantics.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

A straightforward way to meet this completeness requirement is to prove full-graph inference and the subsequent queries over its outputs. For very large graphs, constructing such a proof may exceed practical time or memory budgets. Even when feasible, this upfront cost may be difficult to amortize when few queries are served on the same graph snapshot and model, especially if the graph changes frequently. Much of the proved computation may then contribute nothing to any requested prediction. This motivates query-local proof generation as a cost-reduction strategy, while retaining the same completeness guarantee.

Technical barriers. Localizing the computation must not weaken this guarantee. The required neighborhoods differ in size, and multiple dependency paths may reach the same node-layer state. Authenticating each supplied neighbor does not establish that the list is complete, and recomputing normalization from the retained subgraph can change the prediction. Likewise, authenticating a cached state does not establish its computational provenance; shared states may be reused only after their complete computation has been verified within the query proof.

To organize these checks, the required computation can be expressed as a sequence of bounded operations that consume neighbor entries, update aggregation state, and read or write node-layer states. For a fixed public configuration, the same operation definitions can then represent different queries through different execution traces. The key is therefore to equip these operations with authenticated state and composition rules, so that their local checks jointly establish complete query inference over the same committed graph and model.

Our contributions. To our knowledge, we are the first to formalize the *complete GNN verification problem*: given commitments to a graph and a model, prove that the output for a specific query is derived from all dependencies required by GNN inference over the committed graph, without revealing the private graph or model.

We propose zkMPNN, a framework for complete query-local verification of MPNN inference over private graphs and models.[1] The proved relation binds each prediction to the complete query-dependent computation induced by the registered graph: required neighborhoods and original degrees are authenticated, intermediate node-layer states are accepted only after their computation has been verified, and verified states can be safely reused across overlapping dependency paths. Together, these mechanisms establish recursive dependency completeness while allowing proof execution to remain query-local, avoiding MPNN evaluation over unrelated parts of the graph while preserving full-graph inference semantics. We instantiate this relation using generic SNARKs and Nova's folding machinery (Kothapalli et al., 2021).

We evaluate zkMPNN across 16 configurations on real-world graphs, spanning four GNN families and four tasks. We demonstrate zkMPNN achieves end-to-end verification and observe lower prover cost and peak memory than an EZKL-based full-graph baseline. Controlled synthetic experiments characterize how graph size, query dependencies, depth, and circuit shape affect proving cost.

Our contributions are as follows:

- **Problem formulation.** We formalize the complete GNN verification problem, which requires proving that a query result is derived from all graph dependencies required to compute that query under the committed GNN without revealing the private graph or model.
- **Framework.** We introduce a complete query-local verification framework for finite-depth MPNNs that combines authenticated neighborhoods and features, verified intermediate-state reuse, and task-specific readout.
- **Theoretical analysis.** We formalize the proved relation and establish conditional correctness, soundness, and zero-knowledge guarantees.
- **Empirical evaluation.** We evaluate end-to-end proving costs across four GNN model families, analyze the effects of query dependencies and circuit shape on proving cost, quantify numerical effects on held-out tasks, and compare query-local verification with an EZKL-based full-graph baseline.

[1] Anonymous source code: `https://anonymous.4open.science/r/zkMPNN`.

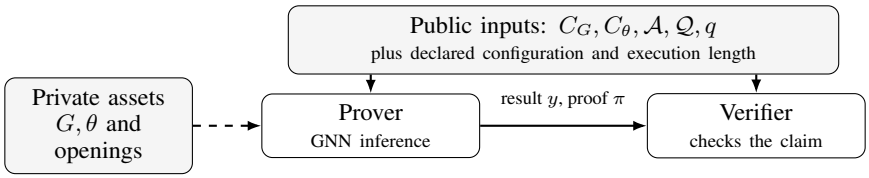


Figure 1: **Complete GNN verification problem.** The prover holds the registered graph and model privately and returns a query result with a proof of Equation (3). The verifier checks the claim against the public inputs without receiving the graph, parameters, or intermediate states. Dashed arrows carry private inputs; solid arrows carry public data.

2 Related Works

For a broader overview of verifiable machine learning, see Lyu et al. (2026). Many works have realized zero-knowledge proof verification of neural network inference. Some target specific types of neural network models, such as zkCNN and ZKML, which certify neural-network inference (Liu et al., 2021); others attempt to be compatible with universal neural network models, such as zkML (Chen et al., 2024) and EZKL (Zkonduit, 2026). However, none of them take into account the requirement of local inference completeness in GNNs. zkGCN is the first work to attempt to bring GCN into this field (Ding et al., 2026), but its approach takes pre-normalized and pre-aggregated node features as inputs, which makes inputs and outputs public while keeping model parameters private. This differs from proving raw features, complete neighbor lists, and original degrees to a private registered graph without publishing intermediate vectors.

Beyond neural-network inference, zkUCB applies zk-SNARKs to privacy-preserving verification of UCB decisions in multi-armed bandit problems (Jiang et al., 2024). This concerns a different verification target from complete message passing over a committed graph. Appendix F gives the detailed related-work comparison and clarifies the motivating application.

3 Problem definition

Graph inference. Let $G = (V, E, X)$ be a registered graph with $N = |V|$ nodes and node features $X = (x_v)_{v \in V}$. For each node v, let $R(v) = [u_0, \ldots, u_{\delta_v - 1}]$ contain every registered incoming neighbor and exactly one self entry, with $\delta_v = |R(v)|$. The public architecture $\mathcal{A}$ fixes a finite depth L and the layer operators. A separate public numerical contract $\mathcal{Q}$ specifies integer encodings, rounding, approximations, and admissible ranges. With model parameters θ, embedding $h_v^{(\ell)}$ for node v at layer $\ell = 1, \ldots, L$ is defined by

$$\begin{aligned} h_v^{(0)} &= x_v, \\ m_{uv}^{(\ell)} &= \mathsf{Msg}_{\ell,\theta}^{\mathcal{Q}}\left(h_u^{(\ell-1)}, h_v^{(\ell-1)}, \delta_u, \delta_v\right), \\ h_v^{(\ell)} &= \mathsf{Upd}_{\ell,\theta}^{\mathcal{Q}}\left(h_v^{(\ell-1)}, \mathsf{Agg}_{\ell,\theta}^{\mathcal{Q}}\left([m_{uv}^{(\ell)}]_{u \in R(v)}\right)\right). \end{aligned} \tag{1}$$

Every required aggregation uses the entire registered neighbor list in its registered order and the original degrees wherever normalization requires them. The operators are local; graph-wide normalization and unsupported edge attributes lie outside the implemented model class.

A node-classification (NC) query $q = v$ returns the logit vector $h_v^{(L)}$. A link-prediction (LP) query $q = (u, v)$, with $0 \le u < v < N$, returns $\langle h_u^{(L)}, h_v^{(L)} \rangle$ without a sigmoid. We write $\mathsf{Eval}_{\mathcal{A},\theta}^{\mathcal{Q}}(G, q)$ for this readout under $\mathcal{Q}$. A *complete query inference* recursively determines all states required by the readout from the registered features and neighborhoods. Its correctness is exact relative to $\mathcal{Q}$; agreement with a floating-point checkpoint is a separate empirical question.

Query-locality. $S_\ell(q)$ indexes the node states required at network layer ℓ to answer query q. Let $S_L(q)$ contain the queried node for NC or both endpoints for LP. The required states are identified

recursively by

$$S_{\ell-1}(q) = \bigcup_{v \in S_\ell(q)} R(v), \qquad \ell = L, \ldots, 1. \tag{2}$$

The resulting node-layer states, together with their required feature and original-degree records, form the query's *dependency closure*. Graph inference should process this closure rather than evaluate unrelated node-layer states. Cost still depends on the architecture and authentication overhead, and the closure may span the entire graph. Section 5 defines the associated workload counts and accounts for setup and graph/model loading separately.

Complete GNN verification problem. Figure 1 illustrates our problem of complete GNN verification. The graph and model owners publish commitments C_G and C_θ through externally accepted, correctly encoded registrations. A prover holds the private graph G, model parameters θ, and secret commitment randomizers r_G, r_θ; together, these constitute the commitment-opening information. A verifier knows the commitments, $\mathcal{A}$, $\mathcal{Q}$, and q, but not the graph or parameters. The task is to return a result y and a proof π for the claim

$$\exists\, G, \theta, r_G, r_\theta : \quad C_G = \mathsf{Com}_G(G; r_G), \quad C_\theta = \mathsf{Com}_\theta(\mathcal{A}, \theta; r_\theta), \quad y = \mathsf{Eval}^{\mathcal{Q}}_{\mathcal{A},\theta}(G, q). \tag{3}$$

The result and all declared metadata, including configuration and execution length, are also public; the commitment-opening information and intermediate states remain private. The proof system must accept honest in-range executions, prevent efficient provers from obtaining acceptance for false claims except with negligible probability, and be zero knowledge relative to this public view. Section 5 states the security assumptions and scope.

4 Methodology

4.1 Construction overview and proof obligations

We employ the committed inference relation in Equation (3) using authenticated graph records, verified state reuse, and incremental proofs. The prover holds the registered graph and model together with their secret commitment randomizers; a query proof uses the required records and authentication data, private parameters, and intermediate states.

The construction enforces four obligations: (i) every coefficient used in arithmetic is the committed one, at its committed layer, role, index, and dimension; (ii) every required node-layer update consumes its complete registered neighbor list, using original degrees wherever normalization requires them; (iii) every non-input state used by the readout is produced by a verified update from authenticated predecessors, so no unverified embedding or frontier can be substituted; and (iv) the published result is the declared query's completed readout, not a computation prefix. Sections 4.2–4.4 realize these obligations.

Proposition 1 (Complete local evaluation)**.** *For a valid registered graph and the local deterministic operators in Equation* (1)*, evaluating the dependency closure with original neighbor lists, features, degrees, and the same numerical contract gives the same query output as full-graph evaluation.*

The proof is induction on the layer: every predecessor needed by a required state is included and uses identical arithmetic. A bare induced neighborhood is insufficient if it replaces original degrees at boundary nodes; Appendix D gives the induction.

4.2 Registration and authenticated neighborhood streams

To instantiate the graph commitment and authenticate the records required by Equation (3), the graph owner first constructs an authenticated representation reusable across queries. Using domain-separated hash functions, define a backward neighbor-list chain and a node leaf:

$$c_{v,\delta_v} = 0, \qquad c_{v,j} = \mathsf{H}_{\mathrm{nbr}}(v, j, u_j, c_{v,j+1}), \qquad a_v = \mathsf{H}_{\mathrm{node}}(v, \delta_v, c_{v,0}, \mathsf{H}_{\mathrm{feat}}(x_v)). \tag{4}$$

A Merkle tree (Merkle, 1988) over the node leaves yields an internal root R_G with height $D_g = \lceil \log_2 N \rceil$. Canonical feature encoding and domain-separated empty leaves make padding unambiguous. The registered graph metadata specifies the feature type, dimension, and scale. To hide the deterministic graph root and model fingerprint, zkMPNN publishes randomized commitments

$$C_\theta = \mathsf{H}_\theta(r_\theta, \mathsf{H}_{\mathrm{arch}}(\mathcal{A}), P, \mathsf{pack}(\theta)), \qquad C_G = \mathsf{H}_G(r_G, R_G, N, D_g, \mathsf{inputSpec}), \tag{5}$$

where P is the parameter count, inputSpec is the registered feature specification, and the two randomizers are independent, uniform, secret field elements. Parameters have one architecture-determined ordering; each is range-checked as a signed 32-bit integer before canonical packing. The implementation uses Poseidon-based hashing (Grassi et al., 2021). Computational hiding requires a secret-prefix/random-oracle assumption in addition to collision resistance; a deterministic hash or a public salt would not provide this argument.

For a node-layer update, a start instruction authenticates a_v and initializes a cursor and chain head. Each edge instruction authenticates the source node, checks the current chain link, advances the cursor, and accumulates its message; an update is allowed only when the cursor equals δ_v and the head reaches zero. Omitting an entry, changing its source, or ending early therefore cannot produce a completed neighbor list without violating authentication. Authenticating δ_u as well as δ_v preserves the original degree normalization rather than recomputing degrees on an extracted subgraph.

4.3 Complete layers through a verified-write-only memo

To enforce the intermediate-state dependencies of Equation (1), an authenticated state memo stores values at node-layer addresses. It begins at a canonical EMPTY root, a value can be inserted only after its full update has been verified and only into an empty position. Its authenticated leaf binds the layer, node identifier, and complete vector, so later layers read at the exact predecessor address and a value from the wrong layer or another node is unusable even if its entries look plausible. The honest planner computes the dependency closure backward, emits updates in increasing layer order, and computes shared node-layer dependencies once. Addresses can be organized as $\ell 2^{D_g} + v$, with memo height $D_m = D_g + \lceil \log_2(L+1) \rceil$, supporting finite depth.

Two policies handle the first-layer raw-feature projection. *Inline* authenticates and projects raw features when needed; *cached* first proves each distinct required raw projection and stores it at stage zero. Neither accepts an unverified embedding; caching reduces repeated work but adds authenticated writes. A minimal schedule is not required because extra verified states are harmless but cost work.

4.4 Model operators, private blocks, and task completion

Model and attention operators. The architecture $\mathcal{A}$ specifies state widths $d_0, \ldots, d_L$, per-branch projection widths $p_1, \ldots, p_L$, and activations. Our operator implementations instantiate Equation (1) as graph convolutional networks (GCN) with degree-normalized aggregation (Kipf & Welling, 2017), GraphSAGE's non-self mean and separate root projection (Hamilton et al., 2017), graph isomorphism networks (GIN) with two-affine updates (Xu et al., 2019), and graph attention networks (GAT) (Veličković et al., 2018), with biases and activations constrained under $\mathcal{Q}$ (Appendix C). The GAT-specific execution schedule and workload are detailed in Appendix E.2.2.

Block folding and readout. Each block contains B fixed-shape instructions with constrained instruction and layer selection. It privately opens C_θ over all P range-checked coefficients and uses those same variables in arithmetic, incurring $\kappa_\theta(P)$ per block; it also privately opens C_G at cost κ_G. All candidate layer branches are compiled, so the per-instruction constraint cost C_μ includes inactive branches, yielding Equation (8). Graph roots and parameter digests remain private. To enforce the readouts specified in Section 3, NC authenticates $h_q^{(L)}$ and publishes its logits; LP authenticates both final-layer endpoints from the same memo and publishes $s = \langle h_u^{(L)}, h_v^{(L)} \rangle$ for $0 \le u < v < N$, without a sigmoid. Readout enters canonical DONE, clearing witness-dependent state; padding is permitted only thereafter. Nova folds the blocks into a compressed proof (Kothapalli et al., 2021), and verification checks the public initial state, declared block count, and entire canonical final state, thereby enforcing completion of Equation (3).

5 Security and cost analysis

Theorem 1 (Conditional complete-query guarantees)**.** *Fix correctly encoded, externally accepted registrations and a valid public configuration. Assume canonical encodings and correct compilation of bounded integer arithmetic and state transitions. Then: (i) backend completeness gives an accepted proof of Equation* (3) *for honest in-range execution; (ii) binding commitments and authenticated structures, together with backend knowledge soundness, prevent efficient provers from obtaining*

acceptance for a different result except with negligible probability; and (iii) backend zero knowledge and the randomized commitments' computational hiding protect witness information beyond the declared public view.

Hiding requires the secret-prefix/random-oracle assumption, not collision resistance alone. The public view includes configuration, query, result, and execution length.

Proof sketch. Correctness follows by induction over the closure: Proposition 1 equates local and full-graph evaluation, and the layer order supplies every predecessor before use. Soundness combines binding of C_θ and C_G, which fixes the model and graph; neighbor-list termination, which prevents omitted or shortened lists; the verified-write-only memo, which admits a state only after its update and therefore excludes unverified stale or injected states; and the constrained readout with its canonical DONE state, which forces the published result to be the query readout. Backend knowledge soundness then yields an extractor for these conditions. Zero knowledge follows from the backend simulator and the randomized commitments: the public statement contains the commitments, the declared public configuration and execution length, the query, and the result, and the terminal state clears all remaining witness-dependent fields. Appendix D gives the full arguments.

Query workload and circuit shape. For the dependency sets in Equation (2), define

$$K_\ell = |S_\ell(q)| \quad (0 \le \ell \le L), \qquad A_q = \sum_{\ell=1}^{L} K_\ell, \qquad U = \sum_{\ell=1}^{L} \sum_{v \in S_\ell(q)} \delta_v. \tag{6}$$

Here A_q counts node-layer updates, whereas U counts neighbor-list entries with multiplicity across layers, not distinct edges. These counts describe the honest planner; the circuit need not reject a longer schedule containing additional correctly verified states. We define the *query workload* T as the exact number of active (non-padding) instructions emitted by the honest planner; let $c \in \{0, 1\}$ indicate cached projection. For GCN, GraphSAGE, and GIN (see Appendix E.2.2 for GAT),

$$T = cK_0 + U + 2A_q + 1, \qquad T_{\text{pad}} = B \cdot \lceil T/B \rceil. \tag{7}$$

Thus topology affects the number of blocks through dependencies and traversals, not merely through N. Let $C_\mu(\mathcal{A}, d_0, D_g, D_m)$ denote the constraints for a fixed-shape local instruction, including all candidate layer arithmetic and authentication, and let $\kappa_\theta(P)$ and κ_G be the model- and graph-opening costs per block. Then

$$C_B = \kappa_\theta(P) + \kappa_G + BC_\mu, \qquad W_{\text{constraints}} = \lceil T/B \rceil (\kappa_\theta(P) + \kappa_G + BC_\mu). \tag{8}$$

κ_θ includes P signed range checks and hashing $\lceil P/7 \rceil$ packed words. Increasing B amortizes that authentication across more instructions but enlarges the block circuit and its keys. C_μ depends on widths, depth, and graph/memo authentication heights, including inactive candidate branches; it is not the cost of the selected arithmetic branch alone.

Time and space. We distinguish honest per-query proving from registration and reusable preprocessing, graph/model loading, and configuration-specific setup. If M_B is the backend-augmented block size and s_z its state arity, proving decomposes into witness construction, $\lceil T/B \rceil$ folding steps for query workload T, and final compression, $\mathsf{Witness} + \lceil T/B \rceil \mathsf{Fold}(M_B, s_z) + \mathsf{Compress}(M_B, s_z, \lceil T/B \rceil)$, with setup separate; we do not assume constant-size keys. For fixed $\mathcal{A}$ and query dependencies, appending disconnected graph regions outside the query's dependency closure leaves T unchanged while increasing authentication height logarithmically. For canonical neighbor lists, graph storage is $O(F_X + N + |E|)$ and preprocessing time is $O(F_X + W_X + N + |E|)$, where F_X counts stored feature entries and W_X counts encoded feature words hashed (Appendix E). The current prover materializes the Bt-instruction padded trace and maintains a query memo, so small compressed proofs do not imply small total memory. Appendices E and A.2 give detailed bounds and measurement scopes.

6 Experiments and evaluation

In this experiment, we answer four questions: **RQ1** evaluates real-data feasibility, prover/verifier costs, and the full-graph comparison; **RQ2** measures numerical fidelity and task preservation; **RQ3**

studies graph size at fixed query dependencies, density, and dependency overlap; **RQ4** studies depth, input/hidden width, and block size. Appendix A and B report detailed settings and full results for **RQ1/RQ2** and **RQ3/RQ4**, respectively.

6.1 Experimental design

Real-data evaluation (RQ1–RQ2). The real-data evaluation experiment is designed to evaluate GCN, GraphSAGE, GIN, and GAT on Cora (Yang et al., 2016) and ogbn-arxiv (Hu et al., 2020) for NC, and Cora and ogbl-collab for LP. Each configuration proves minimum-, lower-median-, and maximum-T test queries three times, with independent verification. All four GNN families are instantiated as two-layer MPNNs (hidden width 32 on Cora and 64 on ogbn-arxiv/ogbl-collab). Both studies use four-thread Nova/Spartan for zkMPNN. Proof time excludes loading, setup, and verification. Reported verification assumes the public key is available, so it needn't be regenerated every time. RSS is the process-lifetime peak.

For comparison, we implemented a full-graph GCN inference baseline on Cora based on EZKL 23.0.5[2] with the same NC and LP tasks and the same three queries. It is compared with three repeat-index sums of the original private zkMPNN query costs, including fresh setup per proof on both sides. The baseline's public topology/support, deterministic digests, numerical semantics, and resource limits differ. RQ2 compares outputs with the integer reference under $\mathcal{Q}$ and floating-point checkpoints, and measures full-test-set task quality without per-sample proofs.

Controlled synthetic evaluation (RQ3–RQ4). This evaluation uses controlled synthetic graphs generated with the Barabási–Albert model (Albert & Barabási, 2002), using the same four GNN models and two tasks. Every axis is a single-variable intervention: exactly one quantity is varied while the graph, model, query, and all remaining settings stay fixed; shared baseline levels are reused across axes rather than re-measured.

6.2 RQ1: End-to-end verification costs

Real-data feasibility. All 144 zkMPNN runs produce independently verified proofs: three repeats for each of 48 selected model–query cases across four GNN families and four dataset–task settings. This establishes end-to-end execution of complete query-local inference proofs for the evaluated workloads, including the OGB graphs.

Prover and verifier costs. Table 1 reports both sides of the protocol. At lower-median workload, mean proving times are 187.7–645.3 s and mean process-peak RSS is 6.4–20.7 GiB. Query costs vary substantially even on the same graph: for ogbn-arxiv GCN, T increases from 1,427 to 106,434 between the lower-median and maximum-workload queries, while mean proving time rises from 338.5 to 14,205.5 s. In contrast, lower-median proofs occupy 14.2–18.4 KiB and pure verification takes 3.2–11.1 s with parameters and keys available. The implementation thus places most per-query work at the prover while keeping proof transfer and verification comparatively small.

Comparison with GCN(EZKL). For the same three selected Cora queries, the EZKL observations are respectively $2.41\times$ and $3.19\times$ the native mean cold costs. Peak RSS over the contributing native runs is 10.67/11.23 GiB (NC/LP), versus 255.05/327.14 GiB for EZKL. These measurements show lower prover cost and peak memory for zkMPNN on the tested workloads.

6.3 RQ2: Numerical fidelity and task preservation

Every proved output equals this integer reference exactly for the 48 model–query cases; agreement with the floating-point checkpoint is a separate empirical relation.

Table 2 summarizes the output deviations for all 48 cases and the full-test-set numerical results. The largest error, 0.5424 in decoded units, occurs for the maximum-T ogbl-collab GIN query; it is a reference-versus-float comparison, while the proved output equals the integer reference exactly. For the full-test evaluation, *decision agreement* is the fraction of examples on which the floating-point checkpoint and the integer reference under $\mathcal{Q}$ produce the same discrete prediction: the argmax class for NC, or whether the raw LP score satisfies $s \geq 0$ (zero is positive). This measures agreement

[2] https://github.com/zkonduit/ezkl/tree/v23.0.5

Table 1: Real-data costs (NC: node classification; LP: link prediction). For ZKMPNN model rows, exact T pairs with Prove (mean $\pm$ sample SD of three independently verified process repeats per workload rank). RSS, Proof, and Verify are repeat means at lower-median T; Verify time excludes key generation. GCN(EZKL) does not depend on queries since it uses the entire graph; therefore, the proof time is the same for any queries. GCN(EZKL) est. rows indicate the estimated values from ZKMPNN's measured max-workload prove-time ratios in Cora.

Data/task	Model/system	$T_{\min}$	$\text{Prove}_{\min}$ (s)	T_{lmed}	$\text{Prove}_{\text{lmed}}$ (s)	$T_{\max}$	$\text{Prove}_{\max}$ (s)	RSS (GiB)	Proof (KiB)	Verify (s)
Cora/NC	GCN(EZKL)				$\text{Prove}_{\text{any}}$ (s): 4,134.5			255.05	6.03	12.5
	GCN	13	$384.3_{\pm 116.6}$	48	$321.6_{\pm 0.3}$	596	$474.8_{\pm 121.9}$	10.5	14.4	5.6
	GraphSAGE	13	$629.9_{\pm 2.4}$	48	$639.3_{\pm 1.3}$	596	$775.9_{\pm 0.8}$	19.8	15.7	10.9
	GIN	13	$317.6_{\pm 0.5}$	48	$324.7_{\pm 0.3}$	596	$409.8_{\pm 0.1}$	10.9	14.4	5.5
	GAT	40	$322.3_{\pm 0.5}$	126	$336.5_{\pm 1.8}$	1,718	$582.9_{\pm 0.5}$	10.9	14.8	5.5
Cora/LP	GCN(EZKL)				$\text{Prove}_{\text{any}}$ (s): 5,289.8			327.14	8.22	15.1
	GCN	13	$317.4_{\pm 0.3}$	58	$326.5_{\pm 0.9}$	1,020	$478.2_{\pm 0.2}$	11.2	14.2	5.6
	GraphSAGE	13	$631.8_{\pm 0.5}$	58	$645.3_{\pm 0.3}$	1,020	$900.7_{\pm 1.6}$	20.7	15.5	11.1
	GIN	13	$319.4_{\pm 0.2}$	58	$327.1_{\pm 0.6}$	1,020	$486.2_{\pm 0.5}$	11.6	14.2	5.5
	GAT	39	$324.8_{\pm 0.4}$	165	$345.4_{\pm 0.5}$	2,997	$816.0_{\pm 0.9}$	11.6	14.6	5.6
ogbn-arxiv/NC	GCN(EZKL) est.				$\text{Prove}_{\text{any}}$ (s): 123,709 (est.)			7,631 (est.)	—	—
	GCN	14	$160.6_{\pm 0.2}$	1,427	$338.5_{\pm 0.3}$	106,434	$14,205.5_{\pm 259.7}$	6.4	16.1	3.2
	GraphSAGE	14	$315.6_{\pm 0.6}$	1,427	$577.4_{\pm 0.8}$	106,434	$20,883.2_{\pm 210.4}$	10.6	18.4	5.5
	GIN	14	$312.8_{\pm 0.2}$	1,427	$515.5_{\pm 0.3}$	106,434	$16,443.0_{\pm 279.7}$	9.4	16.4	5.3
	GAT	58	$167.7_{\pm 0.1}$	3,064	$562.9_{\pm 2.4}$	214,882	$31,167.9_{\pm 566.6}$	6.9	16.4	3.3
ogbl-collab/LP	GCN(EZKL) est.				$\text{Prove}_{\text{any}}$ (s): 45,159 (est.)			2,793 (est.)	—	—
	GCN	13	$157.4_{\pm 0.1}$	225	$187.7_{\pm 0.2}$	28,448	$4,082.5_{\pm 1.7}$	7.1	14.9	3.2
	GraphSAGE	13	$315.7_{\pm 0.9}$	225	$360.5_{\pm 0.4}$	28,448	$6,274.7_{\pm 36.3}$	11.5	17.2	5.4
	GIN	13	$306.6_{\pm 0.1}$	225	$341.7_{\pm 0.9}$	28,448	$5,002.8_{\pm 1.6}$	10.1	15.2	5.3
	GAT	55	$168.8_{\pm 0.4}$	649	$258.4_{\pm 1.1}$	63,053	$9,878.4_{\pm 19.2}$	7.4	15.1	3.3

Table 2: Selected-query output error (decoded units) and held-out task preservation across four settings (NC: node classification; LP: link prediction). Each row summarizes 12 query-level MAEs (four models, three workload ranks) as mean $\pm$ sample SD, with the largest absolute coordinate error; SD describes these deterministic model/query points, not repeated runs or a random-sample confidence interval. Native score is accuracy (NC) or average precision (LP), shown as the range over four models. Agreement is the worst model's decision agreement; $|\Delta\text{metric}|$ is the largest absolute task-metric change in percentage points, computed before rounding.

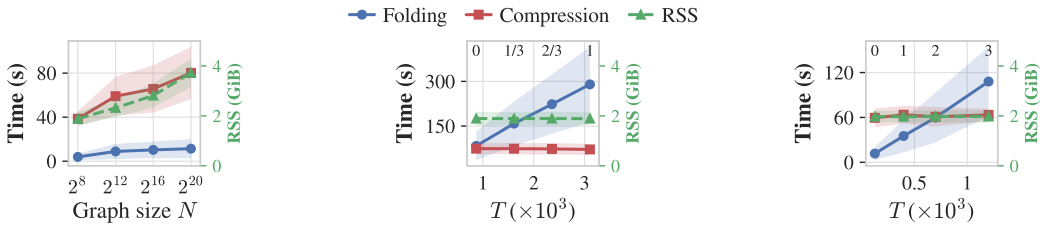
Data/task	Selected-query MAE	Max absolute error	Native score range	Worst agreement (%)	Max $\lvert\Delta\text{metric}\rvert$ (pp)
Cora/NC	$4.82_{\pm 7.97} \times 10^{-4}$	4.280×10^{-3}	0.6190–0.8280	100.000	0.000
Cora/LP	$3.27_{\pm 7.89} \times 10^{-3}$	2.696×10^{-2}	0.7570–0.8081	100.000	0.002
ogbn-arxiv/NC	$3.14_{\pm 6.19} \times 10^{-3}$	5.248×10^{-2}	0.6055–0.6740	99.669	0.021
ogbl-collab/LP	$4.67_{\pm 15.61} \times 10^{-2}$	5.424×10^{-1}	0.8892–0.9656	99.962	0.006

between implementations, not accuracy against ground-truth labels. Across the 16 configurations, 1,022,172 model–example comparisons yield 309 changed decisions and a pooled agreement of 99.97%. The lowest agreement is 99.669% for ogbn-arxiv GIN (161 changes among 48,603 test nodes). These full-test evaluations use host-side reference inference without per-example proofs. The results indicate that the declared numerical semantics preserve the vast majority of decisions for these evaluated checkpoints.

6.4 RQ3: Graph size and query dependencies

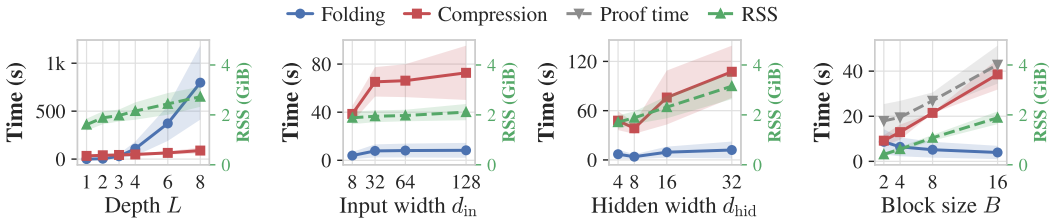
Graph size has limited influence at fixed local workload. With query dependencies and T fixed, proof-generation time and peak RSS grow much more slowly than N (Figure 2(a)). Query-local execution avoids proving work proportional to unrelated graph growth, although authentication and whole-graph storage still contribute overhead.

Density primarily affects traversal work. Adding edges within fixed dependency sets raises U and T, increasing folding and total proving time. Compression time and peak RSS remain nearly unchanged over this scan (Figure 2(b)).



(a) Graph size at fixed query workload (b) Density at fixed dependency sets (c) Degree-preserving rewiring

Figure 2: Structural controls. Lines show means over eight model–task groups, each averaged over three process repeats; shaded bands show ± one sample SD across the group means.



(a) Depth (b) Input width (c) Hidden width (d) Block size

Figure 3: Architectural controls. Lines show means over eight model–task groups, each averaged over three process repeats; shaded bands show ± one sample SD across the group means.

Dependency overlap matters even at fixed global statistics. Degree-preserving rewiring changes the query closure and T without changing N, edge count, or model depth. Larger query workloads increase folding, while compression and peak RSS remain approximately stable (Figure 2(c)).

6.5 RQ4: Architecture and Proof Configuration

Depth increases both workload and circuit cost. Deeper models increase T and folding time; compression time and peak RSS also rise (Figure 3(a)). Folding dominates at greater depths. Unlike fixed-depth rewiring, this intervention changes both execution length and circuit shape.

Width affects cost even when instruction counts change little. Input width increases proving time at fixed T, mainly through compression, with a modest RSS increase (Figure 3(b)). Hidden width increases peak RSS and generally raises compression cost, but proving time is not monotonic at the narrowest settings (Figure 3(c)). Thus T alone is insufficient to predict cost.

Larger blocks trade folding for compression and memory. At fixed active T, increasing B reduces folding time but raises compression time and peak RSS (Figure 3(d)). Over the tested range, compression costs outweigh folding savings and total proving time increases: fewer folds need not mean cheaper proofs.

7 Conclusion and Limitations

We formalized the complete GNN verification problem, and then presented zkMPNN, a framework for complete GNN inference verification over committed graphs and models. Our evaluation demonstrated end-to-end verification across four GNN families and showed that proving cost is governed by both query dependencies and circuit shape. These results showed the feasibility of complete query-local verification for GNN inference and identified proof efficiency as an important direction for future work.

Limitations. zkMPNN has several limitations. First, our experiments cover selected queries, fixed checkpoints, and limited graph instances. Second, Theorem 1 is conditional on registration, bounded arithmetic, correct compilation, commitment security, and backend assumptions; the proofs certify inference but not data provenance or training. Finally, proving time remains impractical for large graphs and queries with high workload.

AI USE STATEMENT

Generative AI tools assisted manuscript drafting, automated consistency checks, and table/figure generation code. The authors review and take full responsibility for the contents of this manuscript.

ETHICS STATEMENT

The experiments use public citation-network and collaboration benchmarks and synthetic graphs rather than newly collected personal data. Public benchmark artifacts cannot be made private retroactively. A proof establishes execution of a specified model, not fairness, appropriate deployment, predictive validity, or lawful access to the underlying data. Real private deployments require access control, authenticated registration, and safeguards against inference through outputs and operational metadata.

REPRODUCIBILITY STATEMENT

Appendices A and B give the two experimental protocols and full results; Appendices C–E give numerical semantics and conditional analysis. The measurement snapshots underlying the reported numbers and the scripts that produce the paper's tables and figures accompany the submission (`scripts/build_final_assets.py` and `scripts/eval_task_quality.py`); they require no private openings. The implementation is available in our anonymous repository at `https://anonymous.4open.science/r/zkMPNN`, and its `docs/REPRODUCE.md` documents `scripts/exp1_main.py` and `scripts/exp2_controls.py` for preparation, proving, and independent verification. Re-running the experiments requires the implementation, prepared graph data, checkpoints, and configuration inputs, which are not part of the accompanying artifact. Appendix A.1.1 documents the full-graph comparison with the original experiment-1 runs; `data/external_baseline.json` and `scripts/build_external_baseline_assets.py` reproduce its tables without rerunning proofs.

REFERENCES

Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47–97, 2002. doi: 10.1103/RevModPhys.74.47.

Bing-Jyue Chen, Suppakit Waiwitlikhit, Ion Stoica, and Daniel Kang. ZKML: An optimizing system for ML inference in zero-knowledge proofs. In *Proceedings of the Nineteenth European Conference on Computer Systems*, pp. 560–574, 2024. doi: 10.1145/3627703.3650088.

Dawei Cheng, Yao Zou, Sheng Xiang, and Changjun Jiang. Graph neural networks for financial fraud detection: A review. *Frontiers of Computer Science*, 19(9):199609, 2025. doi: 10.1007/s11704-024-40474-y.

Hongfa Ding, Linjiang Wu, Yingxuan Luo, and Heling Jiang. ZkGCN: Zero-knowledge based verifiable inference for graph convolutional networks. In *Algorithms and Architectures for Parallel Processing*, volume 16384 of *Lecture Notes in Computer Science*, pp. 377–394. Springer Nature Singapore, 2026. doi: 10.1007/978-981-95-8408-6_20. ICA3PP 2025.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pp. 1263–1272, 2017. URL `https://proceedings.mlr.press/v70/gilmer17a.html`.

Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, pp. 291–304, 1985. doi: 10.1145/22145.22178.

Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In *30th USENIX Security Symposium*, pp. 519–535, 2021. URL `https://www.usenix.org/conference/usenixsecurity21/presentation/grassi`.

Jens Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology—EUROCRYPT 2016*, volume 9666 of *Lecture Notes in Computer Science*, pp. 305–326. Springer Berlin Heidelberg, 2016. doi: 10.1007/978-3-662-49896-5_11.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *Advances in Neural Information Processing Systems*, volume 33, pp. 22118–22133, 2020. URL https://arxiv.org/abs/2005.00687.

Xikun Jiang, He Lyu, Chenhao Ying, Yibin Xu, Boris Düdder, and Yuan Luo. Privacy-preserving UCB decision process verification via zk-SNARKs. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pp. 5900–5908. International Joint Conferences on Artificial Intelligence Organization, 2024. doi: 10.24963/ijcai.2024/652. URL https://www.ijcai.org/proceedings/2024/652.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. Cryptology ePrint Archive, Paper 2021/370, 2021. URL https://eprint.iacr.org/2021/370.

Tianyi Liu, Xiang Xie, and Yupeng Zhang. zkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pp. 2968–2985, 2021. doi: 10.1145/3460120.3485379.

He Lyu, Xikun Jiang, Boris Düdder, and Yuya Sasaki. Survey on verifiable machine learning. *ACM Computing Surveys*, 2026.

Ralph C. Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology—CRYPTO '87*, pp. 369–378, 1988. doi: 10.1007/3-540-48184-2_32.

Arvind Narayanan and Vitaly Shmatikov. De-anonymizing social networks. In *IEEE Symposium on Security and Privacy*, pp. 173–187, 2009. doi: 10.1109/SP.2009.22.

Daryna Oliynyk, Rudolf Mayer, and Andreas Rauber. I know what you trained last summer: A survey on stealing machine learning models and defences. *ACM Computing Surveys*, 55(14s):1–41, 2023. doi: 10.1145/3595292.

Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. Cryptology ePrint Archive, Paper 2019/550, 2019. URL https://eprint.iacr.org/2019/550.

Xiaoli Shen, Vedant Jain, and Xin Huang. Detect financial transaction fraud using a graph neural network with Amazon SageMaker. AWS Machine Learning Blog, June 2022. URL https://aws.amazon.com/blogs/machine-learning/detect-financial-transaction-fraud-using-a-graph-neural-network-with-amazon-sagemaker/.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Bang Wu, Xingliang Yuan, Shuo Wang, Qi Li, Minhui Xue, and Shirui Pan. Securing graph neural networks in MLaaS: A comprehensive realization of query-based integrity verification. In *IEEE Symposium on Security and Privacy*, pp. 2534–2552, 2024. doi: 10.1109/SP54263.2024.00110.

Hao Wu, Changzheng Wei, Yanhao Wang, Li Lin, Yilong Leng, Shiyu He, Minghao Zhao, Hanghang Wu, Ying Yan, and Aoying Zhou. Zero-Knowledge Verifiable Graph Query Evaluation via Expansion-Centric Operator Decomposition. In *2026 IEEE 42nd International Conference on Data Engineering (ICDE)*, pp. 461–473, 2026. doi: 10.1109/ICDE65706.2026.00041.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km.

Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48, pp. 40–48, 2016. URL https://proceedings.mlr.press/v48/yanga16.html.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 974–983, 2018. doi: 10.1145/3219819.3219890.

Zkonduit. zkonduit/ezkl, January 2026. URL https://github.com/zkonduit/ezkl. original-date: 2022-07-05T19:54:03Z.

Table 3: Real-data configurations. Encoder edges exclude self entries and, for link prediction, all supervised positive edges; test units are nodes for node classification or pairs for link prediction. T (lmed) is the query workload of the lower-median selected query, shown as the range over the four model families.

Benchmark	Task	N	Encoder edges	Input width	Test units	T (lmed)
Cora	Node classification	2,708	5,278	1,433	1,000	48–126
Cora	Link prediction	2,708	3,169	1,433	1,054	58–165
ogbn-arxiv	Node classification	169,343	1,157,799	128	48,603	1427–3064
ogbl-collab	Link prediction	235,868	614,662	128	204,886	225–649

Appendix guide. Appendices A and B contain the experimental protocols and complete records. Appendix C specifies the integer operators; Appendix D gives the conditional security arguments; and Appendix E details execution schedules and resource accounting, including the GAT-specific workload in Appendix E.2.2. Appendix F provides the detailed related-work comparison and the scope of the motivating application.

A Experiment 1: Real-data evaluation (RQ1–RQ2)

A.1 Experimental design

Table 3 summarizes graph sizes, input dimensions, and test-pool sizes. The following protocol supports both RQ1 and RQ2.

Graph data and features. The real-data study uses Cora, ogbn-arxiv, and ogbl-collab, with node counts and input widths listed in Table 3. The ogbn-arxiv encoder is explicitly symmetrized, giving 1,157,799 non-self undirected edges. Continuous features (ogbn-arxiv and ogbl-collab) are quantized to signed 16-bit integers at scale 2^{14} by round-to-nearest-even; the largest observed quantization error is 3.05176×10^{-5}. Cora retains its binary features.

Test-set construction. For NC, we retain the dataset-provided splits: Cora's Planetoid split has 140 training, 500 validation, and 1,000 test nodes; ogbn-arxiv uses the official time split (training through 2017, validation in 2018, testing from 2019), with 48,603 test nodes. These test pools are fixed before workload ranking.

For LP, a common static edge-holdout protocol keeps the split and negative-sampling conventions consistent across datasets and separates supervised pairs from the message-passing graph. Source positive edges are merged into unique non-self undirected pairs. With m source pairs, a Fisher–Yates stream (seed 1702) assigns $\lfloor 0.10m \rfloor$, $\lfloor 0.05m \rfloor$, and $\lfloor 0.25m \rfloor$ to test, validation, and training supervision; the rest form the encoder graph. All supervised positive edges, including training positives, are excluded from that graph. Negatives are drawn 1:1 from the complement of the complete source positive-edge set (seed 1703), excluding self pairs, reversed duplicates, and cross-split reuse. The test pool therefore has $2\lfloor 0.10m \rfloor$ pairs. Cora has $m = 5{,}278$, giving 527/263/1,319 test/validation/training positive pairs, 3,169 encoder edges, and $527+527 = 1{,}054$ test pairs. Merging ogbl-collab's three official splits yields $m = 1{,}024{,}434$ unique pairs, giving 102,443/51,221/256,108 supervision positives, 614,662 encoder pairs, and $102{,}443 + 102{,}443 = 204{,}886$ test pairs. Its encoder has maximum degree 280 and 1,465,192 directed neighbor-list entries including self. This custom LP split is not the official OGB temporal evaluation; we do not report official metrics such as Hits@50.

Model architectures. We instantiate GCN, GraphSAGE, GIN, and GAT on each of the four dataset–task settings, giving sixteen two-layer MPNNs. Input widths are in Table 3; hidden width is 32 on Cora and 64 on ogbn-arxiv/ogbl-collab. NC output widths are seven for Cora and 40 for ogbn-arxiv; LP embeddings have width 16. GAT splits its total hidden width across four heads on Cora and two on the OGB graphs, with one final head. GIN uses a two-affine multilayer perceptron (MLP) with $\epsilon = 0$; training batch normalization, where used, is folded into the affine parameters before export.

Training and checkpoint selection. Each model is trained once with seed 1701 for 400 epochs using Adam and weight decay 5×10^{-4}. Learning rates are 0.01 for GCN, GraphSAGE, and GIN and 0.005 for GAT. Checkpoints are selected by validation accuracy for NC and by average precision, then area under the ROC curve (AUC), then loss for LP. These single-seed checkpoints do not characterize training-seed variability.

Numerical contract. An integer z at scale σ represents z/σ; Qk denotes scale 2^k. Weights and hidden states use $\sigma = 2^{14}$; binary inputs have scale one, and continuous inputs are quantized at $\sigma_x = 2^{14}$. Signed division uses floor, including on negative inputs. Changing projection, aggregation, or activation rounding order changes the proved function even if real-arithmetic expressions look equivalent.

Each model first obtains a layer-specific projection of the authenticated raw or preceding-layer state. GCN uses a Q16 approximation of the reciprocal square root of the product of the original endpoint degrees; the integer coefficient's prescribed rounding is constrained, and aggregated weighted sums are divided only at the declared normalization point. GIN distributes its first linear transform across neighbor messages, then applies its first bias, ReLU, second affine map and bias, and any non-final ReLU, with the specified projection rounding retained. It has $\epsilon = 0$, so self contributes exactly once. GraphSAGE separates non-self neighbor and self/root projections; the constrained equality of source and target determines inclusion. The non-self sum is divided by $\delta_v - 1$ when nonzero, with zero mean for isolates.

GAT uses multi-head attention in non-final layers, a single final head, and no omitted exponential linear unit (ELU) coordinates. Its exponential/ELU approximation uses clipping at magnitude 16, a $1/1024$ grid, and Q24 bit-factor arithmetic. One grid transition has a one-Q24-unit non-monotonic tick; no exact-exponential or global monotonicity claim is made. The proof establishes this declared approximation, not IEEE floating-point execution. NC publishes Q14 logits. LP publishes the raw Q28 dot product and does not prove a sigmoid.

The current admissible bounds include $N \leq 2^{32} - 1$, $D_g \leq 32$, neighbor-list length and cursor at most 65,535, input width at most 4,096, per-branch projection width p_ℓ and state width d_ℓ at most 64, and $D_m \leq 63$. GraphSAGE combines two projected branches, so its combined projection buffer can have up to 128 coordinates. Coefficients, projections, hidden states, and NC logits are signed 32-bit; aggregation prefixes are signed 64-bit; the LP dot product is signed 70-bit. An honest computation that exceeds these bounds is outside the completeness claim, even if its final output would be small. Invalid values are rejected rather than interpreted modulo the proof field.

Query selection and repeats. Each configuration uses the complete predefined test pool in Table 3: test nodes for NC, and both positive and negative test pairs for LP, not all graph nodes or all possible node pairs. We compute query workload T structurally from the registered neighbor lists and public architecture using Equations (7) and (18), without numerical inference or proof generation. Candidates are sorted by $(T,$ canonical query identifier$)$, fixing ties, and selected at zero-based indices 0, $\lfloor (M-1)/2 \rfloor$, and $M-1$ among M candidates. The middle selection is the lower median (one-based rank $\lceil M/2 \rceil$); unlike averaging the two central workloads for even M, it always identifies an actual query. No separate arithmetic-median or mean-T query is selected. These are workload ranks, not latency ranks; ranking uses neither observed timing nor prediction quality.

Four dataset–task settings $\times$ four models $\times$ three ranks give 48 model–query cases, with three independently verified process repeats each (144 runs). Query identifiers may recur across models. Repeats measure process variability for a fixed query, not additional queries, graphs, or training seeds; the selected ranks are not random samples of the query distribution. Reported SDs use denominator $n - 1 = 2$. Table 4 records coverage and Table 5 gives the selected identifiers and workloads.

Execution protocol. Each process uses inline projection and $B = 16$, loads its graph and model, and rebuilds public parameters (PP), compression prover key (PK), and verifier key (VK) from scratch. Parameters and graph witnesses remain hidden behind randomized commitments. A run is counted only after an independent verifier accepts the proof; this verifier receives only the public registration, claim, and proof, not model weights, graph caches, or randomizers, and derives its own key from the public architecture rather than trusting a prover-supplied key. The operating-system page cache is not flushed and the host is shared, so timings are descriptive rather than controlled.

Hardware and software. The measured environment uses Python 3.13.5, PyTorch 2.14.0+cpu, PyTorch Geometric 2.8.0.post1, and Rust 1.97.1 on a server with two AMD EPYC 7542 processors, 64 available cores, and 2,003.81 GiB RAM; proofs use Nova 0.76.0 over the Pallas/Vesta cycle with an inner-product-argument commitment backend and Spartan compression (Kothapalli et al., 2021; Setty, 2019). Each prover uses four threads, a 128 GiB address-space limit, and a budget of up to 36,000 s, and the independent verifier uses 7,200 s.

Measurement definitions. *Proof time* is trace construction plus recursive initialization, folding, compression, and serialization; it excludes graph and model loading, PP/PK/VK setup, instrumentation, and verification. *Cold wall* is the whole child process, which additionally includes loading, setup, and instrumentation. Peak *RSS* is the whole-child maximum over its lifetime; *witness* is the application witness; *CRS* is the serialized public parameters; and *verify* is pure verification time, excluding verifier-key reconstruction. Application witness and key sizes are counted with an explicit counting writer; separately serialized PP, PK, and VK values are not deduplicated memory.

Implementation checks. Two implementation properties complement the verification protocol. First, the internal relation is exercised against adversarial witnesses: dropped operations, truncated traces, wrong layer selection, forged outputs or queries, altered memo paths and values, omitted neighbor-list entries, altered authenticated degrees, and incomplete GAT passes are rejected, and a small-scale compressed proof rejects a mutated witness while reusing one PP/VK across two weight tensors; these are implementation tests rather than cryptographic evidence. Second, each run verifies the compressed proof against deliberately mutated public statements (altered commitments, output, query, or length); all are rejected. The public parameters and verifier keys depend solely on the public architecture and configuration—task, policy, block size B, graph size N, and authentication height D_g—and remain invariant to private weight values or graph topology.

A.1.1 FULL-GRAPH COMPARATOR

Protocol and registration. EZKL 23.0.5 evaluates the original two-layer Cora GCN with 2,708 nodes, all 1,433 input features, hidden width 32, both graph propagations, and the intermediate ReLU. Each task keeps its original encoder graph and trained checkpoint. One proof serves the original minimum-, maximum-, and lower-median-T queries: nodes 2255, 1986, and 1923 for NC; pairs $(234, 2591)$, $(645, 1358)$, and $(472, 559)$ for LP. Runtime selectors prevent replacing full-graph inference with a query-local subgraph. Only the selected prediction outputs are published: 21 NC logits or three LP scores, with the LP dot products inside the SNARK rather than computed from public embeddings.

Raw feature values and the four model tensors are private inputs bound to deterministic Poseidon digests. Topology, feature support, node order, and numerical configuration are fixed and public. The registration harness pins the canonical graph/model, settings, compiled circuit, and fresh verification key before verification. An independent verifier checks the registered key and actual public instances without reading features, weights, witnesses, or the proving key. Deterministic digests do not provide the hiding of ZKMPNN's randomized commitments. EZKL uses parameter scale 11, output scale 22, rebase multiplier two, three decomposition legs, and Freivalds optimizations, with $k = 23$ and two/three inner columns for NC/LP. Its numerical function differs from ZKMPNN's Q14/Q16 arithmetic and Q28 LP scores; cross-system bit equality and privacy equivalence are not claimed.

Timing, resources, and sample scope. The authorized baseline scope is one repeat per task, hence two full-graph proofs. The recorded scope reduction excludes interrupted subsequent attempts and unrun repetitions; no baseline sample SD is available. We reuse all 18 original private Cora GCN runs (three workload ranks, three repeats, two tasks), without rerunning them. At each repeat index, their three cold times are summed; the three sums provide the mean and sample SD. These are aggregates of separate historical runs, not measured shared-setup batches. Each native sum includes three fresh setups; the baseline includes one. No full-graph cost is tripled or counted as three observations.

Baseline cold cost is the sum of witness, setup, and proof-generation phase walls, including key I/O. Native cold cost is whole-child wall time, including loading, setup, instrumentation, and internal checks. Compilation, registration, preprocessing, SRS acquisition, and independent verification are outside the comparison. Both systems use four threads, but the baseline permits a 512 GiB address space versus the original 128 GiB; its observed RSS exceeds 128 GiB. RSS is the maximum

across the baseline's three prover phases or the nine contributing native query runs per task. The shared-host measurements were taken in different campaigns, so ratios are descriptive, not controlled or equal-budget speedups.

A.2 RQ1: VERIFICATION COSTS AND FULL-GRAPH COMPARISON

Independent verifier timing evidence. Fresh-verifier whole-process times were recorded for the same 144 verified runs. They range from 78.9436 to 295.2475 s and include public-input loading, public-parameter and verifier-key reconstruction, and verification itself; they are not sums inferred from prover setup timings. Per-run records with measurement paths and digests accompany the artifact (`data/verifier_timings.json`).

Table 6 lists every selected query with its proof time and proof size. Table 7 decomposes trace construction, folding, compression, public-parameter setup, compression setup, and whole-child cold wall. Tables 4 and 5 give verified-repeat coverage, query identifiers, and workload counts. Setup is part of cold wall but excluded from proof time, and is charged once per process. Compression dominates small-workload Cora queries; folding grows with the number of blocks on the larger graphs.

The application witness and key accounting follows the definitions of the measurement scope: the padded trace and the shared parameter/opening representation are counted once; separately serialized PP, PK, and VK are distinct objects whose sum is not deduplicated memory; and the compressed proof is small (14–18 KiB) even though the prover materializes a large trace and large keys. The timing evaluation excludes first-deployment registration and preprocessing; cache loading and opening checks are included in measurement setup. Registration constructs authentication data reusable across queries on a fixed snapshot; deployment costs and practical amortization lie outside this evaluation's scope.

Run-to-run variability is small when the host is quiet and visible when it is not: for most fixed queries the proof-time sample standard deviation is below a few seconds (0.03–2.4 s on Cora). The two exceptions are the min- and max-rank Cora GCN NC queries, whose first repeats were 202 s and 211 s slower than their other two repeats, giving sample standard deviations of 116 s and 122 s; shared-host contention is a possible explanation; the timing records alone do not identify the cause. Repeats of a fixed query are not independent queries, and the three deterministic ranks are workload probes rather than a performance sample, so they do not support population estimates.

Table 4: Query workloads T and independently verified repeats per configuration and selected query rank.

			Active instructions T			Verified runs		
Data	Task	Model	min	lmed	max	min	lmed	max
Cora	Node classification	GCN	13	48	596	3/3	3/3	3/3
Cora	Node classification	GraphSAGE	13	48	596	3/3	3/3	3/3
Cora	Node classification	GIN	13	48	596	3/3	3/3	3/3
Cora	Node classification	GAT	40	126	1718	3/3	3/3	3/3
Cora	Link prediction	GCN	13	58	1020	3/3	3/3	3/3
Cora	Link prediction	GraphSAGE	13	58	1020	3/3	3/3	3/3
Cora	Link prediction	GIN	13	58	1020	3/3	3/3	3/3
Cora	Link prediction	GAT	39	165	2997	3/3	3/3	3/3
ogbn-arxiv	Node classification	GCN	14	1427	106434	3/3	3/3	3/3
ogbn-arxiv	Node classification	GraphSAGE	14	1427	106434	3/3	3/3	3/3
ogbn-arxiv	Node classification	GIN	14	1427	106434	3/3	3/3	3/3
ogbn-arxiv	Node classification	GAT	58	3064	214882	3/3	3/3	3/3
ogbl-collab	Link prediction	GCN	13	225	28448	3/3	3/3	3/3
ogbl-collab	Link prediction	GraphSAGE	13	225	28448	3/3	3/3	3/3
ogbl-collab	Link prediction	GIN	13	225	28448	3/3	3/3	3/3
ogbl-collab	Link prediction	GAT	55	649	63053	3/3	3/3	3/3

Full-graph comparison results. Both authorized proofs passed independent verification and rejection checks for changed feature/model/selector/constant digests, altered outputs, and reordered queries. Table 10 gives their phase walls and proof sizes; Table 1 combines them with experiment 1. The snapshot retains source hashes, original-plan exclusions, verification outcomes, and all native observations, whose summaries match the unchanged experiment-1 snapshot.

Table 5: Workload denominators of the 48 model–query cases: query workload T (active instruction count), first-layer dependency size K_0, deeper dependency sets K_1, K_2, node-layer updates A_q, neighbor-list entries with layer multiplicity U, and K_0/N as a percentage of registered nodes.

Data	Task	Model	Rank	Query	T	K_0	K_1	K_2	A_q	U	K_0/N (%)
Cora	Node classification	GCN	min	[2255]	13	2	2	1	3	6	0.07
Cora	Node classification	GCN	lmed	[1923]	48	30	3	1	4	39	1.11
Cora	Node classification	GCN	max	[1986]	596	195	66	1	67	461	7.20
		GraphSAGE	min	[2255]	13	2	2	1	3	6	0.07
		GraphSAGE	lmed	[1923]	48	30	3	1	4	39	1.11
		GraphSAGE	max	[1986]	596	195	66	1	67	461	7.20
		GIN	min	[2255]	13	2	2	1	3	6	0.07
		GIN	lmed	[1923]	48	30	3	1	4	39	1.11
		GIN	max	[1986]	596	195	66	1	67	461	7.20
		GAT	min	[2255]	40	2	2	1	3	6	0.07
		GAT	lmed	[2381]	126	24	4	1	5	37	0.89
		GAT	max	[1986]	1718	195	66	1	67	461	7.20
Cora	Link prediction	GCN	min	[234, 2591]	13	2	2	2	4	4	0.07
Cora	Link prediction	GCN	lmed	[472, 559]	58	12	7	2	9	39	0.44
Cora	Link prediction	GCN	max	[645, 1358]	1020	292	120	2	122	775	10.78
		GraphSAGE	min	[234, 2591]	13	2	2	2	4	4	0.07
		GraphSAGE	lmed	[472, 559]	58	12	7	2	9	39	0.44
		GraphSAGE	max	[645, 1358]	1020	292	120	2	122	775	10.78
		GIN	min	[234, 2591]	13	2	2	2	4	4	0.07
		GIN	lmed	[472, 559]	58	12	7	2	9	39	0.44
		GIN	max	[645, 1358]	1020	292	120	2	122	775	10.78
		GAT	min	[234, 2591]	39	2	2	2	4	4	0.07
		GAT	lmed	[1265, 2291]	165	19	7	2	9	37	0.70
		GAT	max	[645, 1358]	2997	292	120	2	122	775	10.78
ogbn-arxiv	Node classification	GCN	min	[1144]	14	3	2	1	3	7	0.00
ogbn-arxiv	Node classification	GCN	lmed	[80609]	1427	1,260	9	1	10	1,406	0.74
ogbn-arxiv	Node classification	GCN	max	[27646]	106434	39,465	126	1	127	106,179	23.30
		GraphSAGE	min	[1144]	14	3	2	1	3	7	0.00
		GraphSAGE	lmed	[80609]	1427	1,260	9	1	10	1,406	0.74
		GraphSAGE	max	[27646]	106434	39,465	126	1	127	106,179	23.30
		GIN	min	[1144]	14	3	2	1	3	7	0.00
		GIN	lmed	[80609]	1427	1,260	9	1	10	1,406	0.74
		GIN	max	[27646]	106434	39,465	126	1	127	106,179	23.30
		GAT	min	[1144]	58	3	2	1	3	7	0.00
		GAT	lmed	[127679]	3064	1,283	12	1	13	1,410	0.76
		GAT	max	[27646]	214882	39,465	126	1	127	106,179	23.30
ogbl-collab	Link prediction	GCN	min	[133, 165036]	13	2	2	2	4	4	0.00
ogbl-collab	Link prediction	GCN	lmed	[122210, 135134]	225	72	14	2	16	192	0.03
ogbl-collab	Link prediction	GCN	max	[48290, 64204]	28448	4,697	385	2	387	27,673	1.99
		GraphSAGE	min	[133, 165036]	13	2	2	2	4	4	0.00
		GraphSAGE	lmed	[122210, 135134]	225	72	14	2	16	192	0.03
		GraphSAGE	max	[48290, 64204]	28448	4,697	385	2	387	27,673	1.99
		GIN	min	[133, 165036]	13	2	2	2	4	4	0.00
		GIN	lmed	[122210, 135134]	225	72	14	2	16	192	0.03
		GIN	max	[48290, 64204]	28448	4,697	385	2	387	27,673	1.99
		GAT	min	[133, 165036]	55	2	2	2	4	4	0.00
		GAT	lmed	[180747, 183478]	649	90	12	2	14	201	0.04
		GAT	max	[48290, 64204]	63053	4,697	385	2	387	27,673	1.99

Table 6: Per-query proof time as $\mathrm{mean}_{\pm \mathrm{SD}}$ over three independently verified process repeats per query, with median and mean proof size.

						Proof time (s)			
Data	Task	Model	Rank	Query	Verified	mean	SD	med	Proof KiB
Cora	Node classification	GCN	min	[2255]	3/3	384.3	116.59	317.3	14.4
Cora	Node classification	GCN	lmed	[1923]	3/3	321.6	0.30	321.6	14.4
Cora	Node classification	GCN	max	[1986]	3/3	474.8	121.86	404.6	14.4
Cora	Node classification	GraphSAGE	min	[2255]	3/3	629.9	2.37	629.5	15.7
Cora	Node classification	GraphSAGE	lmed	[1923]	3/3	639.3	1.30	639.2	15.7
Cora	Node classification	GraphSAGE	max	[1986]	3/3	775.9	0.77	776.0	15.7
Cora	Node classification	GIN	min	[2255]	3/3	317.6	0.54	317.4	14.4
Cora	Node classification	GIN	lmed	[1923]	3/3	324.7	0.32	324.6	14.4
Cora	Node classification	GIN	max	[1986]	3/3	409.8	0.10	409.9	14.4
Cora	Node classification	GAT	min	[2255]	3/3	322.3	0.49	322.5	14.8
Cora	Node classification	GAT	lmed	[2381]	3/3	336.5	1.80	335.4	14.8
Cora	Node classification	GAT	max	[1986]	3/3	582.9	0.53	582.7	14.8
Cora	Link prediction	GCN	min	[234, 2591]	3/3	317.4	0.31	317.4	14.2
Cora	Link prediction	GCN	lmed	[472, 559]	3/3	326.5	0.90	326.1	14.2
Cora	Link prediction	GCN	max	[645, 1358]	3/3	478.2	0.16	478.2	14.2
Cora	Link prediction	GraphSAGE	min	[234, 2591]	3/3	631.8	0.47	631.8	15.5
Cora	Link prediction	GraphSAGE	lmed	[472, 559]	3/3	645.3	0.33	645.2	15.5
Cora	Link prediction	GraphSAGE	max	[645, 1358]	3/3	900.7	1.56	900.4	15.5
Cora	Link prediction	GIN	min	[234, 2591]	3/3	319.4	0.24	319.3	14.2
Cora	Link prediction	GIN	lmed	[472, 559]	3/3	327.1	0.57	327.0	14.2
Cora	Link prediction	GIN	max	[645, 1358]	3/3	486.2	0.52	486.4	14.2
Cora	Link prediction	GAT	min	[234, 2591]	3/3	324.8	0.35	324.7	14.6
Cora	Link prediction	GAT	lmed	[1265, 2291]	3/3	345.4	0.47	345.4	14.6
Cora	Link prediction	GAT	max	[645, 1358]	3/3	816.0	0.86	815.9	14.6
ogbn-arxiv	Node classification	GCN	min	[1144]	3/3	160.6	0.16	160.7	16.1
ogbn-arxiv	Node classification	GCN	lmed	[80609]	3/3	338.5	0.30	338.5	16.1
ogbn-arxiv	Node classification	GCN	max	[27646]	3/3	14205.5	259.68	14056.8	16.1
ogbn-arxiv	Node classification	GraphSAGE	min	[1144]	3/3	315.6	0.57	315.5	18.4
ogbn-arxiv	Node classification	GraphSAGE	lmed	[80609]	3/3	577.4	0.78	577.5	18.4
ogbn-arxiv	Node classification	GraphSAGE	max	[27646]	3/3	20883.2	210.43	20815.1	18.4
ogbn-arxiv	Node classification	GIN	min	[1144]	3/3	312.8	0.16	312.8	16.4
ogbn-arxiv	Node classification	GIN	lmed	[80609]	3/3	515.5	0.33	515.7	16.4
ogbn-arxiv	Node classification	GIN	max	[27646]	3/3	16443.0	279.66	16309.9	16.4
ogbn-arxiv	Node classification	GAT	min	[1144]	3/3	167.7	0.15	167.7	16.4
ogbn-arxiv	Node classification	GAT	lmed	[127679]	3/3	562.9	2.36	562.1	16.4
ogbn-arxiv	Node classification	GAT	max	[27646]	3/3	31167.9	566.59	30926.6	16.4
ogbl-collab	Link prediction	GCN	min	[133, 165036]	3/3	157.4	0.11	157.4	14.9
ogbl-collab	Link prediction	GCN	lmed	[122210, 135134]	3/3	187.7	0.23	187.7	14.9
ogbl-collab	Link prediction	GCN	max	[48290, 64204]	3/3	4082.5	1.68	4082.2	14.9
ogbl-collab	Link prediction	GraphSAGE	min	[133, 165036]	3/3	315.7	0.91	315.2	17.2
ogbl-collab	Link prediction	GraphSAGE	lmed	[122210, 135134]	3/3	360.5	0.42	360.4	17.2
ogbl-collab	Link prediction	GraphSAGE	max	[48290, 64204]	3/3	6274.7	36.29	6257.7	17.2
ogbl-collab	Link prediction	GIN	min	[133, 165036]	3/3	306.6	0.11	306.6	15.2
ogbl-collab	Link prediction	GIN	lmed	[122210, 135134]	3/3	341.7	0.86	341.4	15.2
ogbl-collab	Link prediction	GIN	max	[48290, 64204]	3/3	5002.8	1.57	5003.6	15.2
ogbl-collab	Link prediction	GAT	min	[133, 165036]	3/3	168.8	0.38	168.7	15.1
ogbl-collab	Link prediction	GAT	lmed	[180747, 183478]	3/3	258.4	1.10	258.2	15.1
ogbl-collab	Link prediction	GAT	max	[48290, 64204]	3/3	9878.4	19.21	9888.4	15.1

Table 7: Timing components per configuration, averaged over nine independently verified runs: three selected queries with three process repeats each. Proof time is trace plus recursive initialization, folding, compression, and serialization; PP setup and compression setup are charged per fresh child and are not included in proof time.

Data	Task	Model	Trace (s)	Fold (s)	Compress (s)	PP setup (s)	Comp. setup (s)	Cold wall (s)
Cora	Node classification	GCN	0.33	33.5	358.3	132.4	12.9	571.1
Cora	Node classification	GraphSAGE	0.64	51.6	626.9	251.8	24.2	1021.0
Cora	Node classification	GIN	0.34	32.5	316.4	128.9	13.6	527.7
Cora	Node classification	GAT	0.33	95.6	316.5	128.8	13.5	590.7
Cora	Link prediction	GCN	0.34	55.9	316.2	129.8	14.2	553.4
Cora	Link prediction	GraphSAGE	0.66	94.1	628.3	254.4	25.8	1072.3
Cora	Link prediction	GIN	0.35	58.4	317.1	130.5	14.8	559.5
Cora	Link prediction	GAT	0.34	176.3	317.1	130.6	14.6	677.1
ogbn-arxiv	Node classification	GCN	0.61	4740.9	159.0	67.0	8.1	4997.9
ogbn-arxiv	Node classification	GraphSAGE	1.05	6942.9	313.2	127.8	12.9	7433.5
ogbn-arxiv	Node classification	GIN	0.66	5444.2	311.0	124.7	11.0	5923.1
ogbn-arxiv	Node classification	GAT	0.76	10470.6	160.4	67.8	8.6	10732.1
ogbl-collab	Link prediction	GCN	0.30	1318.3	156.1	67.5	9.1	1575.2
ogbl-collab	Link prediction	GraphSAGE	0.51	2001.1	313.6	130.2	14.7	2498.4
ogbl-collab	Link prediction	GIN	0.36	1576.5	305.4	124.6	12.0	2052.0
ogbl-collab	Link prediction	GAT	0.35	3272.6	161.0	69.4	9.8	3538.7

Table 8: Proving costs per configuration and query rank. Proof time is $\mathrm{mean}_{\pm \mathrm{SD}}$ over the verified process repeats; RSS is the largest verified peak in GiB; proof size is the mean size in KiB; Verify is the range of pure verification times in seconds.

			Proof time (s)			RSS	Proof	Verify
Data	Task	Model	min T	lmed T	max T	(GiB)	(KiB)	(s)
Cora	Node classification	GCN	$384_{\pm 117}$	$322_{\pm 0}$	$475_{\pm 122}$	10.7	14.4	5.4–5.6
Cora	Node classification	GraphSAGE	$630_{\pm 2}$	$639_{\pm 1}$	$776_{\pm 1}$	19.9	15.7	10.8–11.1
Cora	Node classification	GIN	$318_{\pm 1}$	$325_{\pm 0}$	$410_{\pm 0}$	11.0	14.4	5.4–5.6
Cora	Node classification	GAT	$322_{\pm 0}$	$336_{\pm 2}$	$583_{\pm 1}$	11.0	14.8	5.4–5.6
Cora	Link prediction	GCN	$317_{\pm 0}$	$327_{\pm 1}$	$478_{\pm 0}$	11.2	14.2	5.5–5.6
Cora	Link prediction	GraphSAGE	$632_{\pm 0}$	$645_{\pm 0}$	$901_{\pm 2}$	20.8	15.5	10.8–11.1
Cora	Link prediction	GIN	$319_{\pm 0}$	$327_{\pm 1}$	$486_{\pm 1}$	11.6	14.2	5.4–5.6
Cora	Link prediction	GAT	$325_{\pm 0}$	$345_{\pm 0}$	$816_{\pm 1}$	11.6	14.6	5.5–5.6
ogbn-arxiv	Node classification	GCN	$161_{\pm 0}$	$338_{\pm 0}$	$14206_{\pm 260}$	6.9	16.1	3.2–3.2
ogbn-arxiv	Node classification	GraphSAGE	$316_{\pm 1}$	$577_{\pm 1}$	$20883_{\pm 210}$	11.3	18.4	5.3–5.5
ogbn-arxiv	Node classification	GIN	$313_{\pm 0}$	$516_{\pm 0}$	$16443_{\pm 280}$	10.0	16.4	5.3–5.4
ogbn-arxiv	Node classification	GAT	$168_{\pm 0}$	$563_{\pm 2}$	$31168_{\pm 567}$	7.9	16.4	3.2–3.3
ogbl-collab	Link prediction	GCN	$157_{\pm 0}$	$188_{\pm 0}$	$4082_{\pm 2}$	7.3	14.9	3.2–3.2
ogbl-collab	Link prediction	GraphSAGE	$316_{\pm 1}$	$361_{\pm 0}$	$6275_{\pm 36}$	11.9	17.2	5.4–5.5
ogbl-collab	Link prediction	GIN	$307_{\pm 0}$	$342_{\pm 1}$	$5003_{\pm 2}$	10.3	15.2	5.2–5.4
ogbl-collab	Link prediction	GAT	$169_{\pm 0}$	$258_{\pm 1}$	$9878_{\pm 19}$	7.9	15.1	3.2–3.4

Table 9: Detailed resource and setup accounting, means over three independently verified process repeats at each query rank. NC: node classification; LP: link prediction. Witness is application witness size; PP, PK, and VK are separately serialized public parameters, compression prover key, and verifier key, not deduplicated memory. PP and compression-key setup are excluded from proof time but included in whole-process cold wall.

Data/task	Model	Rank	Witness (MiB)	PP (GiB)	PK (GiB)	VK (MiB)	PP setup (s)	Key setup (s)	Cold wall (s)
Cora/NC	GCN	min	0.40	1.34	1.9e-07	1393.91	128.39	12.77	556.48
	GCN	lmed	0.51	1.34	1.9e-07	1393.91	128.28	13.13	496.74
	GCN	max	2.33	1.34	1.9e-07	1393.91	140.43	12.82	659.95
	GraphSAGE	min	0.76	2.44	1.9e-07	2544.26	251.46	24.12	968.15
	GraphSAGE	lmed	0.88	2.44	1.9e-07	2544.26	252.20	24.01	978.75
	GraphSAGE	max	2.98	2.44	1.9e-07	2544.26	251.84	24.41	1116.23
	GIN	min	0.42	1.40	1.9e-07	1450.58	128.68	13.52	493.78
	GIN	lmed	0.52	1.40	1.9e-07	1450.58	129.20	13.67	502.14
	GIN	max	2.35	1.40	1.9e-07	1450.58	128.88	13.61	587.30
	GAT	min	0.51	1.39	1.9e-07	1435.95	128.55	13.44	498.22
	GAT	lmed	0.77	1.39	1.9e-07	1435.95	128.90	13.38	513.63
	GAT	max	5.99	1.39	1.9e-07	1435.95	128.80	13.64	760.15
Cora/LP	GCN	min	0.41	1.46	1.9e-07	1509.03	129.52	14.11	495.78
	GCN	lmed	0.56	1.46	1.9e-07	1509.03	129.90	14.13	506.36
	GCN	max	3.69	1.46	1.9e-07	1509.03	129.88	14.28	658.07
	GraphSAGE	min	0.77	2.60	1.9e-07	2708.39	254.32	25.85	977.32
	GraphSAGE	lmed	0.95	2.60	1.9e-07	2708.39	254.62	25.76	991.82
	GraphSAGE	max	4.55	2.60	1.9e-07	2708.39	254.39	25.75	1247.62
	GIN	min	0.42	1.52	1.9e-07	1564.26	130.61	14.69	500.78
	GIN	lmed	0.58	1.52	1.9e-07	1564.26	130.34	14.87	508.96
	GIN	max	3.71	1.52	1.9e-07	1564.26	130.65	14.69	668.64
	GAT	min	0.51	1.51	1.9e-07	1554.18	130.47	14.61	506.01
	GAT	lmed	0.93	1.51	1.9e-07	1554.18	130.62	14.57	527.27
	GAT	max	10.16	1.51	1.9e-07	1554.18	130.71	14.57	997.87
ogbn-arxiv/NC	GCN	min	0.13	0.83	1.9e-07	863.78	67.05	8.08	256.16
	GCN	lmed	4.66	0.83	1.9e-07	863.78	67.19	8.09	435.10
	GCN	max	338.13	0.83	1.9e-07	863.78	66.86	8.15	14302.59
	GraphSAGE	min	0.23	1.31	1.9e-07	1377.53	127.85	12.95	489.05
	GraphSAGE	lmed	6.14	1.31	1.9e-07	1377.53	127.97	12.97	752.43
	GraphSAGE	max	442.17	1.31	1.9e-07	1377.53	127.66	12.88	21058.91
	GIN	min	0.20	1.09	1.9e-07	1159.47	124.82	10.96	477.95
	GIN	lmed	4.72	1.09	1.9e-07	1159.47	124.83	10.99	681.53
	GIN	max	338.20	1.09	1.9e-07	1159.47	124.52	10.91	16609.69
	GAT	min	0.29	0.90	1.9e-07	922.91	67.65	8.55	265.95
	GAT	lmed	9.84	0.90	1.9e-07	922.91	68.12	8.61	662.20
	GAT	max	682.54	0.90	1.9e-07	922.91	67.72	8.62	31268.16
ogbl-collab/LP	GCN	min	0.12	0.96	1.9e-07	996.88	67.47	9.12	255.99
	GCN	lmed	0.83	0.96	1.9e-07	996.88	67.63	9.17	287.40
	GCN	max	90.41	0.96	1.9e-07	996.88	67.55	9.12	4182.12
	GraphSAGE	min	0.21	1.49	1.9e-07	1557.42	130.45	14.68	496.56
	GraphSAGE	lmed	1.14	1.49	1.9e-07	1557.42	130.20	14.66	542.17
	GraphSAGE	max	118.27	1.49	1.9e-07	1557.42	130.01	14.73	6456.48
	GIN	min	0.19	1.23	1.9e-07	1295.62	124.35	11.98	473.73
	GIN	lmed	0.90	1.23	1.9e-07	1295.62	124.51	11.98	510.05
	GIN	max	90.48	1.23	1.9e-07	1295.62	124.81	12.04	5172.09
	GAT	min	0.28	1.02	1.9e-07	1045.08	69.55	9.72	271.81
	GAT	lmed	2.16	1.02	1.9e-07	1045.08	69.54	9.81	362.01
	GAT	max	200.32	1.02	1.9e-07	1045.08	69.25	9.75	9982.13

Table 10: One verified EZKL observation per task. Verification wall time is excluded from cold prover cost. Raw proof bytes exclude public instances and JSON framing and are not total communication costs.

Task	Witness (s)	Setup (s)	Prove (s)	Verify wall (s)	Proof (bytes)	Public fields
NC	24.3	1970.9	2139.2	12.5	6176	28
LP	27.3	2473.0	2789.4	15.1	8416	11

A.3 RQ2: NUMERICAL FIDELITY AND TASK PRESERVATION

Table 11 reports the selected-query output errors. Tables 12 and 13 evaluate the same checkpoints and integer exports on the full test sets without per-sample proofs. Decision agreement is the fraction of matching floating-point and integer-reference decisions, not accuracy against the ground-truth labels. NC uses the argmax class, with ties broken by the smallest class index; LP is positive exactly when the raw score is nonnegative. A flip is an example whose two decisions differ, so agreement is $1 - \text{flips}/n_{\text{test}}$. The pooled rate weights configurations by their test-set sizes, counting each model–example comparison once. NC margins are in decoded units. These single-seed checkpoints follow Appendix A.1; the LP protocol is not the official OGB evaluation. The values characterize these checkpoints, not training-seed or graph-instance variability. Full-test measurements are in `data/task_quality.json`, produced by `scripts/eval_task_quality.py`.

Table 11: Floating-point versus integer-reference error for all 48 model–query cases, in dequantized output units. "max" is the largest coordinate error for node classification and the single-pair error for link prediction. Each selected query contributes once regardless of its number of verified repeats.

			min		lmed		max	
Data	Task	Model	MAE	max	MAE	max	MAE	max
Cora	Node classification	GCN	1.41×10^{-4}	3.36×10^{-4}	1.26×10^{-4}	2.67×10^{-4}	3.02×10^{-4}	6.07×10^{-4}
Cora	Node classification	GraphSAGE	2.40×10^{-4}	6.32×10^{-4}	1.28×10^{-4}	4.17×10^{-4}	1.32×10^{-4}	3.82×10^{-4}
Cora	Node classification	GIN	3.19×10^{-4}	8.86×10^{-4}	5.91×10^{-4}	9.54×10^{-4}	2.98×10^{-3}	4.28×10^{-3}
Cora	Node classification	GAT	2.85×10^{-4}	4.76×10^{-4}	2.77×10^{-4}	5.85×10^{-4}	2.63×10^{-4}	8.42×10^{-4}
Cora	Link prediction	GCN	4.09×10^{-5}	4.09×10^{-5}	6.35×10^{-4}	6.35×10^{-4}	1.85×10^{-4}	1.85×10^{-4}
Cora	Link prediction	GraphSAGE	7.80×10^{-5}	7.80×10^{-5}	4.52×10^{-4}	4.52×10^{-4}	3.46×10^{-4}	3.46×10^{-4}
Cora	Link prediction	GIN	1.44×10^{-5}	1.44×10^{-5}	9.21×10^{-3}	9.21×10^{-3}	2.70×10^{-2}	2.70×10^{-2}
Cora	Link prediction	GAT	8.33×10^{-4}	8.33×10^{-4}	2.77×10^{-4}	2.77×10^{-4}	1.92×10^{-4}	1.92×10^{-4}
ogbn-arxiv	Node classification	GCN	1.77×10^{-4}	6.30×10^{-4}	1.24×10^{-4}	3.67×10^{-4}	4.48×10^{-4}	1.47×10^{-3}
ogbn-arxiv	Node classification	GraphSAGE	2.05×10^{-4}	8.17×10^{-4}	2.81×10^{-4}	9.86×10^{-4}	2.07×10^{-4}	9.35×10^{-4}
ogbn-arxiv	Node classification	GIN	1.96×10^{-3}	4.95×10^{-3}	1.82×10^{-2}	5.25×10^{-2}	1.43×10^{-2}	3.41×10^{-2}
ogbn-arxiv	Node classification	GAT	7.87×10^{-4}	3.15×10^{-3}	4.49×10^{-4}	1.65×10^{-3}	5.79×10^{-4}	1.93×10^{-3}
ogbl-collab	Link prediction	GCN	3.86×10^{-4}	3.86×10^{-4}	5.09×10^{-5}	5.09×10^{-5}	6.05×10^{-3}	6.05×10^{-3}
ogbl-collab	Link prediction	GraphSAGE	4.52×10^{-5}	4.52×10^{-5}	8.22×10^{-4}	8.22×10^{-4}	7.25×10^{-4}	7.25×10^{-4}
ogbl-collab	Link prediction	GIN	4.22×10^{-5}	4.22×10^{-5}	1.06×10^{-3}	1.06×10^{-3}	5.42×10^{-1}	5.42×10^{-1}
ogbl-collab	Link prediction	GAT	1.73×10^{-3}	1.73×10^{-3}	2.85×10^{-3}	2.85×10^{-3}	4.04×10^{-3}	4.04×10^{-3}

Table 12: Held-out task quality for node classification, native checkpoint versus integer reference, on the full test splits; no proofs are generated. Margin is the mean top-one minus top-two logit gap in decoded units.

Data	Model	n	Float acc.	Ref. acc.	Agreement	Flips	Mean margin float / ref.
Cora	GCN	1,000	0.8080	0.8080	1.00000	0	3.520 / 3.520
Cora	GraphSAGE	1,000	0.8010	0.8010	1.00000	0	4.077 / 4.077
Cora	GIN	1,000	0.6190	0.6190	1.00000	0	3.610 / 3.609
Cora	GAT	1,000	0.8280	0.8280	1.00000	0	1.039 / 1.038
ogbn-arxiv	GCN	48,603	0.6641	0.6641	0.99998	1	1.528 / 1.528
ogbn-arxiv	GraphSAGE	48,603	0.6740	0.6740	0.99996	2	1.698 / 1.698
ogbn-arxiv	GIN	48,603	0.6055	0.6053	0.99669	161	1.470 / 1.476
ogbn-arxiv	GAT	48,603	0.6366	0.6366	0.99988	6	1.122 / 1.121

Table 13: Held-out task quality for link prediction under the fixed protocol of Section 6.1, native checkpoint versus integer reference, on the full test pair sets; no proofs are generated. Agreement compares the thresholded decision at the declared zero-score boundary.

Data	Model	n	Float AP	Ref. AP	Float AUC	Ref. AUC	Agreement	Flips
Cora	GCN	1,054	0.7988	0.7988	0.8073	0.8073	1.00000	0
Cora	GraphSAGE	1,054	0.7656	0.7656	0.7826	0.7826	1.00000	0
Cora	GIN	1,054	0.7570	0.7569	0.7591	0.7591	1.00000	0
Cora	GAT	1,054	0.8081	0.8080	0.8349	0.8349	1.00000	0
ogbl-collab	GCN	204,886	0.9393	0.9393	0.9435	0.9435	0.99995	11
ogbl-collab	GraphSAGE	204,886	0.9112	0.9112	0.9225	0.9225	0.99991	19
ogbl-collab	GIN	204,886	0.9656	0.9656	0.9668	0.9668	0.99962	78
ogbl-collab	GAT	204,886	0.8892	0.8891	0.9083	0.9082	0.99985	31

B EXPERIMENT 2: CONTROLLED SYNTHETIC EVALUATION (RQ3–RQ4)

B.1 EXPERIMENTAL DESIGN

We use the proving backend and measurement definitions of Appendix A.1, with the controls below.

Synthetic graphs and model dimensions. We instantiate GCN, GraphSAGE, GIN, and GAT for both NC and LP, giving eight model–task groups. All synthetic points use one Barabási–Albert base graph ($N = 256$, $m = 2$, initialized from a star on three nodes, master seed 42). Features are binary with probability $1/2$; weights are drawn from $\mathrm{Uniform}[-0.25, 0.25]/\sqrt{\text{fan-in}}$ per family, layer role, and coordinate; GAT attention vectors use the per-head input width as fan-in; biases are fixed at ± 164 Q14 units. Base widths are input/hidden/output $8/8/4$ with $L = 2$ and inline $B = 16$, except where the scanned axis changes one of these. GAT uses two heads in non-final layers and one in the final layer; hidden width is the total across heads. Each point is executed three times; synthetic graphs carry no labels and we do not evaluate predictive quality on them.

The seven axes and their interventions are as follows. *Total graph size* appends disjoint Barabási–Albert components so that only the graph size, the added disconnected components, and the authentication height change; the queried neighbor lists, features, and degrees are untouched. *Density* adds legal missing edges in steps while preserving the per-layer dependency sets of the query (for LP, the union of both endpoints' closures). *Dependency* keeps N, the edge count, the per-node degree sequence, and connectivity fixed and rewires edges, changing neighborhood overlap and therefore the closure work at fixed global statistics. *Depth* varies $L \in \{1, 2, 3, 4, 6, 8\}$ with the graph, query, widths, and parameter-generation rule fixed; parameter shapes change with L by construction. *Input width* varies the feature dimension with regenerated features and matching parameters. *Hidden width* varies the total hidden width while keeping two attention heads in non-final GAT layers and one in the final layer. *Block size* varies $B \in \{2, 4, 8, 16\}$ with graph, model, query, and active T fixed, so only the folding granularity changes. Table 14 summarizes what each intervention holds fixed and what it varies.

Table 14: Experiment 2 interventions: what each axis holds fixed and what it varies. These are clarifications of the described design, not claims that one axis isolates one causal mechanism.

Axis	Held fixed	Varied or growing
Total graph size N	Query, queried neighbor lists/features/degrees, model, depth, B	Appended disconnected components and authentication height; local K_ℓ, U, A_q, T stay fixed.
Density	Query and per-layer dependency sets, model, depth, B	Neighbor-list lengths and degrees, hence U and T; dependency-set counts stay fixed.
Dependency overlap	N, edge count, per-node degrees, connectivity, query; depth three	Rewired neighbors: dependency-set overlap and the resulting K_ℓ, U, A_q, T.
Depth L	Graph, query, widths, parameter-generation rule, B	Layer count, closure, updates, parameter shapes, candidate branches, memo height.
Input width	Graph and query structure, depth, other widths, B	Regenerated features and matching parameters; feature-authentication dimensions; T stays fixed.
Hidden width	Graph, query, depth, input/output widths, B; GAT head counts	Parameter/state widths and circuit work; GAT activation instructions can change T.
Block size B	Graph, model, query, architecture, active T	Block count and padding, per-block circuit and key size; folding and compression costs.

Synthetic query selection. For each model–task group, all axes except density select the lower-median-T query among all nodes (NC) or unordered node pairs (LP) of the 256-node base graph, at depth three for the dependency axis and two otherwise. The query is then fixed across axis levels. Density uses the highest-degree node for NC or the two highest-degree nodes for LP, with ties broken by node identifier. No query is selected at the arithmetic-mean T.

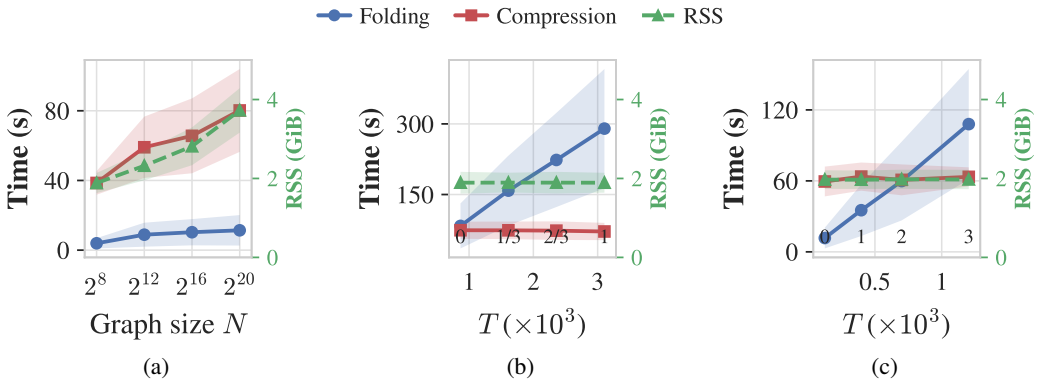


Figure 4: Structural controls (mean over eight family/task groups, three repeats each; shaded bands show ± one sample SD across the groups). Appended components grow authentication height, folding, compression, and memory at fixed traversal work; density and rewiring grow folding with T (x-axes in units of 10^3), while compression changes little over the tested density and rewiring levels.

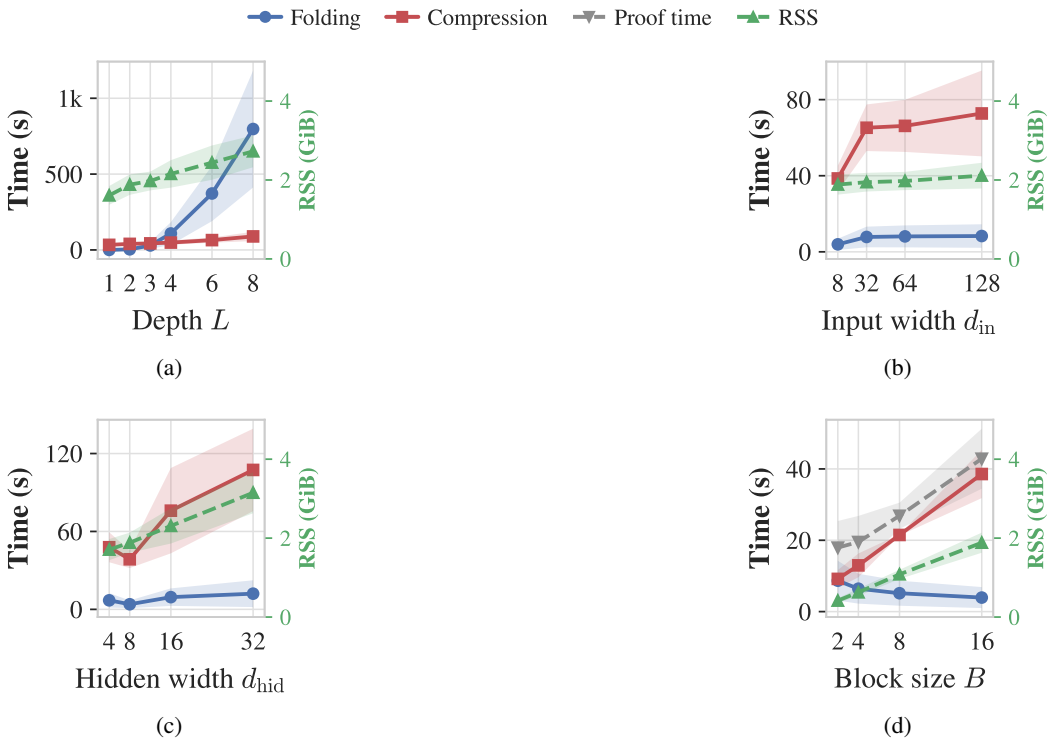


Figure 5: Architectural controls (mean over eight family/task groups, three repeats each; shaded bands show ± one sample SD across the groups). Depth raises folding, compression, and memory; width raises proof-time components and the smallest hidden width is not the fastest point; block size trades folding for compression at fixed T.

B.2 RQ3: Graph size and query dependencies

Figure 4 reports the graph-size, density, and dependency-overlap scans.

B.3 RQ4: Architecture and proof configuration

Figure 5 reports the depth, width, and block-size scans.

B.4 RQ3–RQ4: COMPLETE SCAN RECORDS

Repeats and shared-baseline accounting. Each configuration uses a fixed graph, model, and query; the three runs per point are process repeats, not additional graph instances. Five levels reproduce the base configuration under different axes (graph-size-256, depth-2, input-8, hidden-8, block-16) and share the same graph, model, query, and workload; these results are reused across axes rather than repeated, except that the depth-2 level uses its own GCN/NC repeats and therefore differs slightly from the other levels. The 30 axis levels across eight model–task groups and three repeats comprise 720 measurement entries, represented by 627 independently verified proof runs and 93 entries reusing shared-baseline results. Three of those proofs re-measure one base configuration, so the study comprises 208 distinct configurations in total. Table 15 summarizes every axis level; all 627 measured proofs verified independently. Scheduling differed across axis groups, so absolute times are descriptive comparisons within an axis rather than a controlled speed ranking between axes.

Table 15: Synthetic scans: mean over the eight family/task groups and three repeats per level. T and block counts are deterministic structural quantities; wall is whole-child, fold/compress are proof-time components, RSS is whole-child peak, and proof is the compressed proof size. Shared baseline levels share one workload and are not counted as separate configurations.

Axis	Level	T	Blocks	Wall (s)	Fold (s)	Compress (s)	RSS (MiB)	Proof (B)
Total graph size N	256	96	6	68.8	3.93	38.6	1929	12,961
Total graph size N	4096	96	6	105.9	8.87	59.1	2377	13,039
Total graph size N	65536	96	6	119.3	10.29	65.6	2882	13,078
Total graph size N	1048576	96	6	146.2	11.38	80.2	3814	13,195
Density	0	867	54	203.9	83.36	74.1	1938	12,961
Density	1/3	1,612	101	278.5	158.27	74.0	1940	12,961
Density	2/3	2,358	148	344.2	223.39	73.3	1939	12,961
Density	1	3,105	194	407.6	289.99	71.5	1937	12,961
Dependency	0	125	8	111.1	11.67	59.5	2019	13,000
Dependency	1	397	25	136.8	35.20	63.6	2015	13,000
Dependency	2	700	44	158.3	59.83	60.9	2026	13,000
Dependency	3	1,202	76	209.3	107.98	63.4	2021	13,000
Depth L	1	12	1	55.4	0.09	33.4	1648	12,692
Depth L	2	96	6	70.1	3.95	39.9	1927	12,961
Depth L	3	618	39	100.3	29.36	43.2	2022	13,000
Depth L	4	2,140	134	187.4	108.48	48.4	2206	13,039
Depth L	6	6,732	421	474.5	372.07	64.7	2498	13,156
Depth L	8	11,448	716	931.0	797.19	89.1	2791	13,234
Input width	8	96	6	68.8	3.93	38.6	1929	12,961
Input width	32	96	6	114.0	7.81	65.2	1991	13,000
Input width	64	96	6	115.1	8.12	66.2	2022	13,000
Input width	128	96	6	127.7	8.30	72.7	2161	13,039
Hidden width	4	95	6	86.4	6.96	47.8	1749	12,723
Hidden width	8	96	6	68.8	3.93	38.6	1929	12,961
Hidden width	16	100	6	134.0	9.42	76.0	2365	13,398
Hidden width	32	106	7	184.2	12.09	107.4	3225	14,233
Block size B	2	96	48	24.3	8.67	9.2	426	12,220
Block size B	4	96	25	28.5	6.42	13.0	641	12,415
Block size B	8	96	12	41.5	5.15	21.5	1108	12,688
Block size B	16	96	6	68.8	3.93	38.6	1929	12,961

C EXACT INTEGER OPERATOR RECURRENCES

C.1 SHARED ENCODINGS AND PROJECTIONS

This section specifies the deterministic function proved by our framework, independently of floating-point library conventions. All vectors and coefficients below are integers, all sums follow the registered neighbor order, and every intermediate must satisfy its declared range. Set $\sigma = 2^{14}$, $\sigma_n = 2^{16}$, and $\sigma_e = 2^{24}$. Define $\rho_\ell(z) = \max(z, 0)$ for non-final layers and $\rho_L(z) = z$, coordinatewise. The first-layer input scale is the registered feature scale σ_x; later-layer inputs have scale σ. Let p_ℓ be the public projection width of a branch, distinct from the state width d_ℓ. GIN uses $W_\ell \in \mathbb{Z}^{d_{\ell-1} \times p_\ell}$, $b_\ell \in \mathbb{Z}^{p_\ell}$, $V_\ell \in \mathbb{Z}^{p_\ell \times d_\ell}$, and $c_\ell \in \mathbb{Z}^{d_\ell}$. GCN and GAT have $p_\ell = d_\ell$. GraphSAGE also has $p_\ell = d_\ell$ per branch, with separate $d_{\ell-1} \times d_\ell$ neighbor/root matrices and one d_ℓ-dimensional bias. For example, the final layer of the selected Cora two-layer NC GIN has $p_2 = 32$ and $d_2 = 7$. A dense projection in any branch is

$$z_{u,j}^{(\ell)} = \left\lfloor \frac{\sum_i h_{u,i}^{(\ell-1)} W_{\ell,ij}}{\sigma_{\ell-1}} \right\rfloor, \qquad \sigma_0 = \sigma_x, \quad \sigma_{\ell-1} = \sigma \text{ for } \ell > 1. \tag{9}$$

For binary features this reduces to an exact sum of selected Q14 coefficients. Biases are not included until the update below. Padding contributes zeros and is not an additional feature or learned coordinate.

C.2 GCN, GRAPHSAGE, AND GIN

For GCN, α_{uv} is the positive integer nearest to $\sigma_n / \sqrt{\delta_u \delta_v}$, with an exact halfway case rounded upward. It is characterized without floating-point square roots by

$$(2\alpha_{uv} - 1)^2 \delta_u \delta_v \;\le\; 4\sigma_n^2 \;<\; (2\alpha_{uv} + 1)^2 \delta_u \delta_v. \tag{10}$$

The update is $h_v^{(\ell)} = \rho_\ell(\lfloor \sum_{u \in R(v)} \alpha_{uv} z_u^{(\ell)} / \sigma_n \rfloor + b_\ell)$, coordinatewise. The nearest-integer rule for α is distinct from the floor division of the accumulated messages.

GraphSAGE uses two projections from Equation (9), z_u^{nbr} and z_u^{root}, with separately authenticated matrices. Its update is

$$h_v^{(\ell)} = \rho_\ell\left(\left\lfloor \frac{\sum_{u \in R(v) \setminus \{v\}} z_u^{\mathrm{nbr}}}{\max(1, \delta_v - 1)} \right\rfloor + z_v^{\mathrm{root}} + b_\ell \right). \tag{11}$$

The empty neighbor sum is zero. GIN first forms $g_v = \max(\sum_{u \in R(v)} z_u^{(\ell)} + b_\ell, 0)$ and then

$$h_{v,j}^{(\ell)} = \rho_\ell\left(\left\lfloor \frac{\sum_i g_{v,i} V_{\ell,ij}}{\sigma} \right\rfloor + c_{\ell,j} \right), \tag{12}$$

where g_v has p_ℓ coordinates and $h_v^{(\ell)}$ has d_ℓ coordinates. We fix $\epsilon = 0$, so the registered self entry contributes exactly once. This fixes projection-before-aggregation rounding rather than claiming equivalence to every integer implementation of a GIN layer.

C.3 GAT ATTENTION AND EXPONENTIAL APPROXIMATION

Let I_k be the coordinate range of head k, and $a^{\mathrm{src}}, a^{\mathrm{dst}}$ its authenticated attention vectors. With the projection z_u from Equation (9), define

$$a_{u,k}^s = \left\lfloor \frac{\sum_{j \in I_k} z_{u,j} a_j^{\mathrm{src}}}{\sigma} \right\rfloor, \qquad a_{v,k}^t = \left\lfloor \frac{\sum_{j \in I_k} z_{v,j} a_j^{\mathrm{dst}}}{\sigma} \right\rfloor, \tag{13}$$

$$e_{uv,k} = \lambda(a_{u,k}^s + a_{v,k}^t), \qquad \lambda(x) = \begin{cases} x, & x \ge 0, \\ \lfloor x/5 \rfloor, & x < 0. \end{cases} \tag{14}$$

Both attention dot products are floored separately before addition. For an integer $\Delta \ge 0$ at scale σ, set $i = \lfloor 1024 \min(\Delta, 16\sigma)/\sigma \rfloor$, with bits $i_0, \ldots, i_{14}$ from least to most significant. Define $w_0 = \sigma_e$ and, in increasing bit order,

$$w_{j+1} = \begin{cases} \lfloor w_j f_j / \sigma_e \rfloor, & i_j = 1, \\ w_j, & i_j = 0, \end{cases} \qquad \mathcal{E}(\Delta) = w_{15}. \tag{15}$$

The integer factors $(f_0, \ldots, f_{14})$, in that order, are

$$\begin{aligned}&(16760840, 16744480, 16711808, 16646655, 16517109,\\ &\ 16261035, 15760736, 14805841, 13066109, 10175896,\\ &\ 6171993, 2270549, 307285, 5628, 2).\end{aligned} \tag{16}$$

Let $m_{v,k} = \max_{u \in R(v)} e_{uv,k}$ and $\omega_{uv,k} = \mathcal{E}(m_{v,k} - e_{uv,k})$. After the complete second neighbor-list pass, for $j \in I_k$,

$$b'_{v,j} = \left\lfloor \frac{\sum_{u \in R(v)} z_{u,j}\omega_{uv,k}}{\sum_{u \in R(v)} \omega_{uv,k}} \right\rfloor + b_{\ell,j}. \tag{17}$$

The denominator is positive since a maximum-score entry has weight σ_e. A non-final layer applies $\mathrm{ELU}_{\mathcal{Q}}(x) = x$ for $x \geq 0$, and $\mathrm{ELU}_{\mathcal{Q}}(x) = \lfloor \sigma\mathcal{E}(-x)/\sigma_e \rfloor - \sigma$ otherwise. The final layer returns b'_v directly. Equations (14)–(17) specify the negative slope, clipping, grid, factor order, and every truncation; they do not assert exact real softmax. NC returns the final integer vector and LP returns the exact integer dot product, interpreted at scale σ^2.

D Conditional security arguments

D.1 Statement and invariants

Scope of the relation. Let $x = (\mathcal{A}, \mathcal{Q}, N, D_g, B, \mathsf{policy}, \mathsf{task}, C_\theta, C_G, q, y, t)$ be the formal public statement. Additional published counts, artifact sizes, and timings are operational disclosures. The statement binds t through compressed verification; it does not independently attest every reported statistic or force the honest planner's minimal T. Let w contain parameter and graph openings, authenticated features and chain links, memo paths and values, and the constrained execution trace. The owner registration is fixed externally. Graph canonicalization, graph origin, and approval of model parameters are not silently added to what the local circuit proves.

Proof of Proposition 1. For $\ell = 0$, the required states are exactly the original features indexed by S_0; cached first-layer projections are computed from those same features and parameters. Assume that local evaluation agrees with full-graph evaluation for each required predecessor at layer $\ell - 1$. For any $v \in S_\ell$, the recursion defining $S_{\ell-1}$ includes all $u \in R(v)$. The target itself is included because the self entry occurs in its neighbor list. Every message therefore reads identical preceding-layer values, original endpoint degrees, and parameters. Ordered aggregation and update apply the same numerical contract, including all rounding and activation operations. The resulting $h_v^{(\ell)}$ is identical in both executions. By induction this holds at layer L, and the same task readout gives the same output. The argument does not hold for an operator using an unregistered global statistic or for an extracted graph that recomputes degrees after dropping edges.

Memo invariant. Before each constrained transition, every nonempty memo location contains either a correctly authenticated first-layer raw projection at stage zero or the correctly evaluated full vector $h_v^{(\ell)}$ at its exact node-layer address. Initially the memo is canonical EMPTY, so the invariant is vacuous. A raw-projection write authenticates the feature vector to the registered graph and uses the model parameters authenticated in the current block. A hidden-state write is possible only after a valid start, complete neighbor-list consumption, and the prescribed update. The predecessor reads are valid by the invariant. The empty-entry check prevents overwriting an established value. Reads bind the stage, identifier, and vector, preventing a value written for one location from being reused at another. Thus writes preserve the invariant. The planner may produce extra valid states, but none permits an incorrect required state.

Neighbor-list completeness and attention. The node leaf binds the list's length and initial chain head. Induct on the cursor j. At a valid edge instruction, matching the current chain head to Equation (4) fixes the target, position, source identifier, and next head, except if hash binding is broken. An honest chain terminates at cursor δ_v with head zero. Requiring both conditions prevents an accepted proper prefix or an appended suffix. For GAT, the first pass completes before rewind and preserves the authenticated initial chain head. The second pass starts from that same head and must itself finish. Thus the denominator, weighted numerator, and maximum refer to the same complete

neighbor list. Activation completion precedes the write, so an incomplete activation cannot become a reusable state.

D.2 CONDITIONAL GUARANTEES

Correctness (Theorem 1, part i). Assume registration is valid and every honest intermediate is in the declared range. By Proposition 1, the complete local execution produces the same query result as full-graph evaluation under the declared integer contract. The planner's layer order supplies all predecessor values before use. Correct graph and memo paths satisfy authentication, and every neighbor list is consumed to its endpoint. Model and graph openings are identical across blocks, so commitment checks succeed. NC or LP readout authenticates the final-layer values, publishes the expected result, and clears the terminal scratch/memo fields. The final block is padded by $Bt - T$ legal terminal instructions. This yields a satisfying witness for all blocks, and backend completeness yields acceptance. This is conditional on the complete integer execution being admissible, not only its final output. Correctness here includes proof completeness for honest execution, not a guarantee of predictive accuracy.

Soundness (Theorem 1, part ii). Assume a polynomial-time adversary produces an accepted compressed proof for a different output. By the backend's knowledge-soundness assumption, an extractor obtains a satisfying block computation, except with negligible probability. Canonical encoding and commitment binding ensure that every block uses openings consistent with the previously fixed C_θ, C_G; changing parameters or the internal graph root would otherwise give a binding violation. Applying the memo invariant to the extracted computation establishes that all authenticated final-layer reads are correct complete states for the registered model and graph. Neighbor-list completeness prevents omission of required messages. The constrained readout and exact DONE-state check force the claimed y to equal the inference result, contradicting the assumed different output. Consequently, an incorrect accepted result entails a backend soundness failure, a binding failure, or a failure of the explicitly assumed registration/compilation contracts. The statement concerns computational correctness, not knowledge that an external data source is truthful.

Zero knowledge (Theorem 1, part iii). For a valid statement x, assume that the published compressed backend has a simulator producing a proof computationally indistinguishable from an honest one without access to w. Applying this simulator establishes witness privacy relative to the already public x. The graph root and parameter fingerprint require a separate argument: with independent uniform secret randomizers, the domain-separated secret-prefix hashes in Equation (5) are assumed computationally hiding in the stated random-oracle-style model. Collision resistance alone provides no such conclusion. Reusing commitments reveals linkage, and the public architecture, result, and length can already distinguish deployments.

Only the compressed zero-knowledge object is published. The selected backend introduces fresh random instance/witness material before final compression; inspecting this mechanism is not a substitute for its security proof. The final public state must contain no witness-dependent scratch fields: otherwise those fields would be part of x and simulator-based proof privacy would not hide them. Real deployments must also avoid releasing private witnesses, roots, predictable openings, or their identifying hashes through diagnostics. Public benchmark models and graphs are deliberately disclosed for reproducibility, so there is no claim of hiding previously disclosed content. No privacy guarantee is made against output inference, query linkage, timings, raw logs, or the prover holding the data.

E COST DECOMPOSITION AND EXACT TRACE COUNTS

E.1 BLOCK TRANSITIONS AND COMPLETION

The blocks in Section 4.4 share the public commitments C_G, C_θ; each privately authenticates the internal graph root used for graph paths and the full coefficient vector used for arithmetic. Neither R_G nor a private parameter digest is exposed as a public boundary value. Readout retains only the public result and configuration in canonical DONE, clearing the memo root and scratch fields. Padding instructions preserve this terminal state and are illegal before completion. The prototype folds these

fixed-shape blocks with Nova and compresses the argument using an inner-product-argument (IPA) commitment backend and Spartan (Kothapalli et al., 2021; Setty, 2019). Verification binds the public initial state, declared block count, and entire canonical final state.

E.2 Exact instruction counts

E.2.1 GCN, GraphSAGE, and GIN

For the quantities in Equation (2), K_0 counts distinct required raw nodes, not parameters or necessarily projection instructions. Let $c = 1$ for cached first-layer projections and zero for inline projections. Non-attention models require cK_0 initial projection instructions, A_q starts, U edge instructions, A_q finishes, and one readout, yielding Equation (7).

E.2.2 GAT-specific schedule and workload

For GAT, the first pass authenticates the complete neighbor list and computes each head's maximum logit. A rewind transition preserves these maxima and resets the cursor and chain head to the same authenticated list start. The second pass authenticates that entire list again to accumulate the weighted numerator and denominator; it must reach the registered length and terminal head before finishing. Non-final layers complete the declared exponential linear unit (ELU) activation on every coordinate before committing the state. Let $w = 4\lceil \max_{1\le \ell \le L} d_\ell/4 \rceil$ be the globally padded output width. The two complete edge passes contribute $2U$, each required neighbor list has a start, rewind, and finish, and each non-final node-layer update has $w/4$ four-coordinate activation instructions plus one commit, giving

$$T = cK_0 + 2U + 3A_q + \left(\frac{w}{4} + 1\right) \sum_{\ell=1}^{L-1} K_\ell + 1. \tag{18}$$

The activation count uses the global padded width, not an independently minimized width for each layer. Actual edge-instruction counts are U for non-attention models and $2U$ for GAT. GAT edge counts already measured as two-pass traversals must not be doubled again.

E.2.3 Readout and padding

NC and LP have one and two final endpoints, respectively, but each has a single readout instruction; their readout arithmetic and authentication costs differ. Every padded trace has exactly $t = \lceil T/B \rceil$ blocks and $Bt - T$ terminal instructions, with supported $1 \le B \le 16$.

E.3 Private authentication and arithmetic shape

Let $\mathsf{HashCost}(k)$ be the constraints for hashing k field words. Canonical signed-32 parameter encoding adds 2^{31} to each coefficient and packs at most seven 32-bit words into one field element. At fixed security and numerical widths,

$$\kappa_\theta(P) = \Theta(32P) + \mathsf{HashCost}(3 + \lceil P/7 \rceil) + O(\lceil P/7 \rceil), \qquad \kappa_G = \mathsf{HashCost}(7) + O(1). \tag{19}$$

Parameter order is increasing layer, then first matrix, second matrix if present, first bias, second bias if present, source attention vector, and target attention vector, omitting absent roles. GraphSAGE's first matrix is the column concatenation $[W_\ell^{\mathrm{nbr}} \mid W_\ell^{\mathrm{root}}]$; GIN's second matrix is V_ℓ. Matrices are row-major and vectors use increasing index. For each consecutive group of $m \le 7$ coefficients, the packed word is $\sum_{j=0}^{m-1} (\theta_j + 2^{31}) 2^{32j}$. The public count fixes the last group's length. This layout derives from the public architecture, not a witness-selected layout. A single authenticated coefficient allocation is shared within a block but repeated in the next block. Even all-zero secret weights receive the same constraints as nonzero weights. The application-block relation is therefore Equation (8), with no secret-sparsity shortcut.

One fixed-shape instruction includes a graph path and two memo paths; LP requires a third memo path for its second endpoint. This contributes path work proportional to $D_g + (2 + \mathbf{1}_{\mathrm{LP}}) D_m$, in addition to feature, chain-link, and value hashing. It also contains the configured candidate projections, updates, attention branches, range checks, and selectors. A dense transformation contributes work proportional to the relevant input/output dimension product. Compiling all candidate layers means that increasing

L changes both the trace and the per-instruction circuit. Raw projection remains a synthesized branch even on inactive or cached instructions. These facts preclude identifying constrained work solely with native floating-point multiply counts.

E.4 Preprocessing and storage

Graph preprocessing and storage. Let F_X count stored feature entries and W_X count encoded feature words hashed during graph construction. With fixed security and numeric word sizes, constructing authentication on canonical neighbor lists takes $O(F_X + W_X + N + |E|)$ time. For sparse binary input, F_X counts nonzero indices, but each nonempty feature-matrix row hashes $\lceil d_0/248 \rceil$ packed words; the all-zero digest is computed once. Thus $W_X = (N_{\mathrm{nz}} + 1)\lceil d_0/248 \rceil$, where N_{nz} is the number of nonempty feature-matrix rows. Signed-16 input is stored densely, with $F_X = Nd_0$ and $W_X = \Theta(N(1 + \lceil d_0/15 \rceil))$, including per-node metadata and the lower-order initial zero digest. Only at fixed d_0 can the preprocessing bound be reduced to $O(F_X + N + |E|)$ uniformly across these encodings. The graph store itself remains $O(F_X + N + |E|)$: packed feature words are temporary, while features, neighbor-list chains, feature digests, and tree nodes persist.

Query and backend storage. Let w_m bound memo-vector width, including any padded projection representation. There are $A_q + cK_0$ verified memo values; an upper bound on sparse memo storage is $O((A_q + cK_0)(w_m + D_m))$, allowing authentication paths to share nodes. The materialized padded trace has an upper bound of $O(Bt(d_0 + w_m + D_g + D_m))$ stored witness elements and an additional one-time $O(P)$ shared parameter/opening representation. Temporary neighbor projections, public parameters, backend assignments, recursive accumulators, and compression workspace are extra. This is not a streaming constant-memory implementation. The cost of canonicalizing a noncanonical input must be charged separately from constructing authentication on already sorted neighbor lists.

With backend-augmented circuit size M_B and state arity s_z, setup, folding, compression, proof size, and verification retain their respective backend dependences. The circuit itself contains no graph-wide inference loop, but it includes logarithmic authentication depth and architecture-dependent dimensions. Consequently, keeping the query closure fixed while extending a disconnected graph tests a different scalability claim from expanding the closure at fixed N. Neither implies that all prover resources are independent of graph size.

F Related-work context and the motivating scenario

F.1 Expansion-centric verifiable graph queries as inspiration

Wu et al. (2026) target zero-knowledge verifiability for graph databases: a data owner proves, without revealing the graph, that an issued declarative graph query (e.g., reachability or shortest path) was evaluated correctly and completely. Their central design choice is to decompose these declarative queries into fine-grained primitive operators—most importantly the *expansion* from a node to its neighbors—each verified by a small, composable circuit rather than by one monolithic in-circuit graph operator, with cryptographic checks such as permutation arguments establishing multiset equality of expansions. Our construction is inspired by this expansion-centric decomposition and by its explicit completeness argument for neighbourhood multisets; we extend the idea from graph-query operators to finite-depth message passing, where the expansion is layered (the dependency sets $S_0, \ldots, S_L$), the authenticated object is the registered complete neighbor list with its original degree, and every expanded state must itself have been produced and verified before it can be reused. The proved statement therefore differs (a node- or node-pair MPNN inference under committed parameters and graph) as does the proving strategy (query-local folding of fixed-shape blocks), but the underlying motivation—replacing a prohibitive in-circuit graph operator with authenticated, composable expansions—is shared.

F.2 zkGCN as a direct predecessor

Ding et al. (2026) target end-to-end GCN inference integrity while protecting model parameters. Their construction extracts layer computations into rank-one constraints and uses Groth16 proofs for convolution, activation, global pooling, and classification layers. The evaluation uses two-layer GCN

graph classification on MUTAG, NCI1, and REDDIT-BINARY. The conference is ICA3PP 2025; the cited proceedings chapter was published in 2026. We distinguish the task and proof interface, rather than claiming that this work does not support message passing: GCN is itself an MPNN.

In the interface described in their Section 4.1, layer inputs and outputs are included among public primary inputs, while weights and biases are private witnesses. Their convolutional relation in Equation (2) takes normalized, aggregated node features as its input; Algorithm 1 describes the transformation producing this matrix before the affine relation is encoded. This is a different boundary from the relation studied here. zkMPNN authenticates raw features, complete incoming neighbor lists, and original degrees to a registered graph, and permits reuse of preceding-layer states only after their computation is verified. The private relation binds those dependencies to a public node or node-pair query and its readout, without publishing intermediate vectors. Its realized operator families include GCN, GraphSAGE, GIN, and GAT at configurable finite depth. These differences motivate a comparison of verification statements, not a speed ranking between different tasks, disclosure settings, and backends. We do not report an experimental comparison against that system: running it under matched commitments, numerical semantics, and disclosure is outside the scope of this study.

F.3 Model-integrity detection and inference proofs

Wu et al. (2024) study model-centric attacks on GNN services, including parameter perturbations and model replacement, and construct query-based node fingerprints for integrity verification. That detection problem is related to, but distinct from, a proof for a particular query's complete execution over committed inputs. zkCNN and ZKML address neural-inference proofs and their efficient construction (Liu et al., 2021; Chen et al., 2024); zkMPNN builds on the general possibility of proving arithmetic computations and focuses on specifying and enforcing complete graph-query dependencies. More broadly, verifiable computation and succinct arguments (Kothapalli et al., 2021; Setty, 2019) provide the machinery; the contribution here is the relation and the query-local authenticated message-passing construction that makes it provable at graph scale.

F.4 What the financial example establishes

The financial-fraud literature supports the use of graph relationships in prediction (Cheng et al., 2025), and the AWS example describes a cloud deployment workflow (Shen et al., 2022). Neither source reports a deployment of zkMPNN. The AWS workflow uses a heterogeneous relational GCN; it serves as motivating domain context rather than evidence that our current implementation processes heterogeneous pipelines. Our illustrative audit uses an already registered static, single-relation graph and an authorized prover holding both graph and model. It does not claim private computation across institutions that withhold their inputs from that prover. Likewise, the topology-based re-identification result of Narayanan & Shmatikov (2009) motivates avoiding unnecessary graph disclosure; it does not establish protection from inference through public outputs or metadata. Graph/model registration must be accepted before a claim is proved, and the proof does not establish the truth or completeness of the underlying real-world transaction records. The subgraph-omission example in the Introduction is explanatory, not an additional benchmark or financial prediction result.