VERIF: An Efficient Zero-Knowledge Proof System for Verifying IVF-Flat Retrieval in RAG Services

Zhiwen Zhang[1,2,*], Yuao Zhou[1,2,*], Ge Chang[1,2], Cong Li[1,2], Yuejian Fang[1,2], Qingni Shen[1,2,†]
[1]School of Software and Microelectronics, Peking University
[2]Beijing Key Laboratory of Data Intelligence and Security, Peking University
Email: zwzhang25@stu.pku.edu.cn, gozya@stu.pku.edu.cn, gchang25@stu.pku.edu.cn, li.cong@pku.edu.cn, fangyj@ss.pku.edu.cn, qingnishen@pku.edu.cn

Abstract

Retrieval-augmented generation (RAG) services outsource vector search over proprietary corpora, yet clients cannot verify that returned context conforms to the promised index, parameters, and snapshot. We present Verif, the first dedicated zero-knowledge polynomial interactive oracle proof (PIOP) for complete, service-consistent IVF-Flat retrieval. Verif proves top-m centroid selection, authenticated routing, exact full-vector scoring of every routed candidate, final top-k selection, and context binding. Its commitment-eliding reduction keeps query-dependent scores virtual and reduces selection claims directly to inner products over authenticated data. A unified, permutation-free top-t relation with limb-decomposed range arguments handles both selection stages without sorting or score commitments. Against a matched, optimized implementation of the same retrieval relation using a general-purpose circuit-based zkSNARK (Plonky2), our prototype achieves up to an 86.5× prover speedup and reduces peak memory by up to 99.1%. Verif proves retrieval over authenticated SIFT and 768-dimensional Cohere indexes containing 32 million and 8 million vectors in 5.90 and 11.57 seconds, respectively; verification takes 0.62–1.48 seconds. These results demonstrate practical verifiable IVF-Flat retrieval for RAG-as-a-Service.

CCS Concepts

• **Security and privacy → Cryptography**.

Keywords

zero-knowledge proofs, RAG, verifiable vector retrieval, approximate nearest neighbors

1 Introduction

Retrieval-augmented generation (RAG) has become a standard approach for grounding large language models (LLMs) in external knowledge [12, 25]. A RAG system embeds a user query, retrieves the top-k relevant chunks from a vector database, and incorporates them into the LLM prompt. This architecture enables access to up-to-date, domain-specific, and proprietary information without frequent model retraining, and is increasingly deployed in enterprise, healthcare, financial, and legal applications [7, 34, 45].

At the core of RAG is dense retrieval over a large embedding database. A naive exact search compares the query with all N vectors, requiring $\Theta(ND)$ operations for D-dimensional embeddings. Production systems therefore commonly employ approximate nearest-neighbor (ANN) indexes such as IVF-Flat, IVF-PQ, and HNSW [19, 21, 31]. HNSW offers high recall and low latency but requires additional graph storage and adaptive traversal. IVF-PQ reduces storage and scoring costs through product quantization, at the expense of lossy distance estimates. IVF-Flat instead retains complete vectors: it first selects the top-m clusters and then directly scores all vectors within them to retrieve the top-k chunks. This coarse-to-fine search is globally approximate because unprobed clusters are excluded, but introduces no additional PQ error within the candidate domain and avoids the graph-storage and construction overhead of HNSW. It is therefore well suited to accuracy-sensitive workloads that have sufficient memory to retain full vectors but seek to avoid the additional graph memory and index-construction costs of HNSW. IVF-Flat is supported by widely used systems including Faiss, Milvus, and pgvector [21, 22, 42].

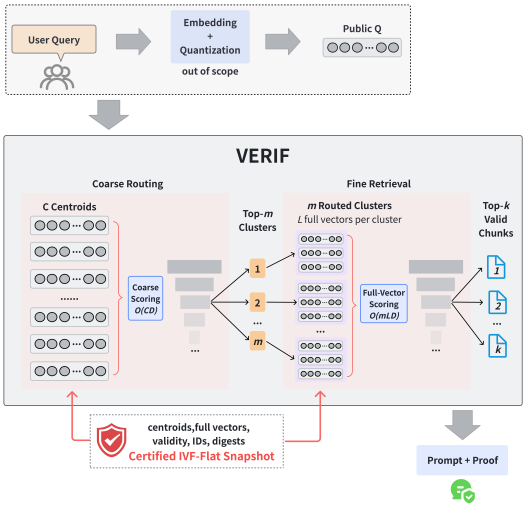

Figure 1: End-to-end scope and proof boundary of Verif.

Operating a production-scale RAG system requires substantial resources for preprocessing, embedding generation, index maintenance, storage, and low-latency retrieval, and may involve proprietary or sensitive corpora that clients cannot directly access. These constraints have driven the adoption of managed platforms that provide hosted knowledge bases and retrieval APIs, commonly

*Equal contribution.
†Corresponding author.

known as *RAG-as-a-Service* [30]. Such services lower the deployment barrier but introduce a fundamental trust gap: the provider controls the corpus, index, and retrieval execution, while clients observe only the returned chunks. A provider may skip retrieval, reduce the advertised search budget, return cached results, query a stale snapshot, or substitute a lower-quality corpus to reduce cost or bias the output. Such deviations are difficult to detect because LLMs can produce plausible responses even from incomplete or irrelevant context. Meanwhile, clients cannot rerun the search without accessing the provider's proprietary or sensitive database. This motivates *service-consistent retrieval*: the returned results should equal the output of an agreed retrieval algorithm, with agreed parameters, over an authenticated database snapshot. Figure 1 summarizes the resulting proof boundary: Verif proves complete coarse-to-fine IVF-Flat retrieval and binds the returned chunks to a certified snapshot, taking the public quantized query as input; query embedding and quantization, prompt construction, and downstream LLM inference remain outside the proved relation.

Zero-knowledge proofs (ZKPs) offer a promising way to establish this guarantee without revealing the private database. Prior work has applied ZKPs to decision trees, logistic regression models, convolutional neural networks (CNNs), federated learning, and general-purpose machine learning frameworks [1, 2, 4, 6, 8–10, 13, 16, 24, 27, 28, 37, 40, 43, 44, 46, 47]. More recently, zkLLM and zkGPT have enabled efficient verification of LLM inference [38, 41]. However, these ZK-for-LLM systems certify inference on a supplied prompt, but not whether the retrieved context used to construct that prompt is correct. A valid LLM proof over manipulated retrieval results can therefore still certify an untrustworthy RAG execution. Although a general-purpose zkSNARK can in principle prove retrieval, directly encoding an exhaustive vector scan or the data-dependent execution of an ANN index incurs prohibitive proving costs. Efficient verifiable RAG thus requires proof systems tailored to the underlying retrieval algorithm.

Recent work has developed index-specific proofs for RAG retrieval. V3DB [36] verifies fixed-shape IVF-PQ using multiset relations and boundary checks, while VeriRAG [26] combines the Goldwasser–Kalai–Rothblum (GKR) protocol with lookup arguments for IVF-PQ retrieval. zkRAG [20] instead designs specialized lookups and checkers for the adaptive graph traversal and priority queues of HNSW. None of these works develops a polynomial interactive oracle proof (PIOP) tailored to IVF-Flat retrieval, and this gap is nontrivial. V3DB observes that proving ND full-vector distance computations produces an impractically large circuit, while VeriRAG reports that PQ is crucial to its efficiency, reducing per-cluster commitment size eightfold in its evaluated configuration. IVF-Flat removes this compression and requires a full D-dimensional score for every candidate in the probed clusters. A straightforward replacement of PQ codes with full vectors would therefore forfeit the compression-dependent efficiency of existing IVF-PQ protocols. The central difficulty is to preserve IVF-Flat's native full-vector search complexity after introducing a proof system: proof-specific representations can otherwise cost more than the retrieval they certify. This manifests in three bottlenecks.

(1) Score materialization breaks the native computation structure. With C centroids and per-cluster capacity L, a query produces C coarse scores and up to mL fine scores. Native retrieval consumes these scores directly in top selection. A modular proof instead separates scoring from selection, forcing it to commit to query-dependent score polynomials and later open them. These polynomial-commitment operations add linear-size multi-scalar multiplication work and memory in both stages beyond the underlying full-dimensional scoring.

(2) Selection completeness induces global rearrangement. Membership and cardinality do not prove top-t: every better-ranked unselected item must also be excluded. Sorting inside the proof introduces comparisons, conditional swaps, and data-dependent routing, while sorting externally materializes permuted arrays together with permutation and boundary checks. IVF-Flat incurs this cost for both top-m cluster selection and top-k candidate selection, with the latter additionally complicated by padded slots and score ties.

(3) Finite fields lack an inexpensive ordering interface. Ranking keys combine shifted scores with identifiers and therefore span many bits, while field arithmetic provides no native notion of sign or nonnegativity. Bit-decomposing every ordering gap incurs work proportional to the key width across all $C+mL$ items, whereas a lookup table covering the complete key range grows exponentially. This cost can dominate an otherwise linear selection proof.

These bottlenecks share a common cause: conventional proof interfaces materialize values and orderings that native IVF-Flat does not need to represent explicitly. Verif therefore follows a *native-complexity* principle. It keeps scores virtual and propagates selection claims directly to exact inner products over authenticated data, eliminating query-dependent score commitments. A common top-t relation replaces sorting and permutation in both stages, while fixed-width limb tables certify the resulting ordering gaps. LogUp and homomorphic commitment folding then authenticate the routed subindex and returned context. For fixed quantization and limb parameters, the dominant online prover work is $O(CD + mLD)$, matching the native IVF-Flat scan footprint.

This work makes the following contributions:

- We present Verif, the first dedicated zero-knowledge PIOP for service-consistent IVF-Flat retrieval in RAG. Verif proves the complete coarse-to-fine retrieval execution over an authenticated private snapshot, covering exact full-vector scoring, top-m cluster selection, authenticated coarse-to-fine routing, top-k candidate selection, and binding of the returned RAG context.
- We introduce a commitment-eliding score–selection reduction that propagates selection consistency directly to exact inner-product claims over offline data commitments. It eliminates query-dependent commitments and openings for original and permuted coarse and fine score arrays while retaining dominant prover work of $O(CD + mLD)$.
- We develop a unified, permutation-free top-t framework for IVF-Flat's two selection stages. Its all-valid instantiation proves top-m centroid selection, while its validity-aware instantiation proves top-k selection over the padded candidate domain. Deterministic keys handle ties, and limb-decomposed range arguments certify ordering without sorting or permutation proofs.

- We implement the complete Verif system, including a CPU prover and an optional CUDA backend that provides reusable Sumcheck and BN254 multi-scalar multiplication components for future proof systems. We evaluate quantization fidelity, end-to-end performance, memory efficiency, scalability, and GPU acceleration across two complementary datasets and multiple IVF-Flat configurations. Against a matched, optimized Plonky2 baseline, it achieves up to an 86.5× prover speedup and reduces peak memory by up to 99.1%. In a quantization-only exact-search experiment, 16-bit affine quantization leaves all 11,000 top-1 results unchanged and retains at least 99.9828% mean top-64 overlap. Verif further proves enlarged SIFT and Cohere indexes containing 32M and 8M vectors in 5.90 and 11.57 seconds, respectively, while verification takes 0.62–1.48 seconds across all evaluated scales.

2 Technical Overview

Verif realizes the native-complexity principle by aligning the proof dataflow with that of IVF-Flat retrieval. Rather than compiling the complete retriever into a monolithic circuit or introducing committed interfaces between its modules, Verif uses specialized algebraic checkers connected through random evaluation claims. Query-time oracles contain selection masks, threshold and gap limbs, and lookup auxiliaries, but no score or sorted-array oracles. We next explain how this design resolves the three bottlenecks from Section 1 and binds the resulting computation to one authenticated snapshot. Figure 2 summarizes this online proof flow.

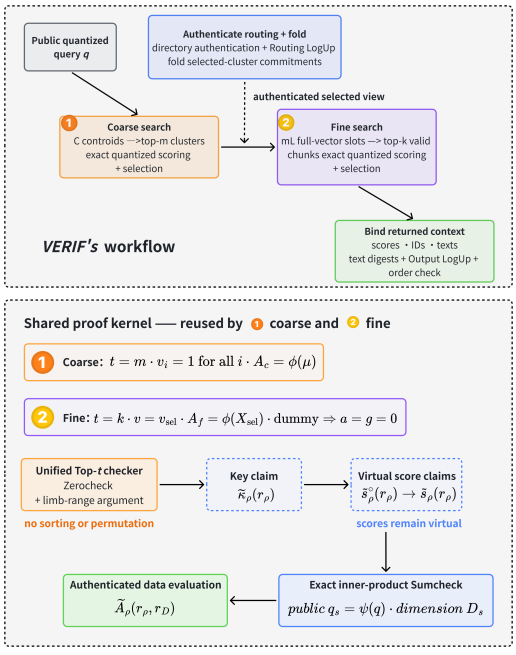

Figure 2: Verif's online execution and shared proof kernel.

Keeping scores virtual through claim propagation. The layer-reduction principle of GKR provides the key design insight: a claim about a later computation layer can be reduced to one about the preceding layer without committing to every intermediate value [14]. Encoding the entire IVF-Flat pipeline as a GKR circuit, however, would combine scoring, selection, range checks, and query-dependent data access within one heterogeneous circuit. Inspired by GKR's principle, Verif develops a stage-specific claim-propagation chain tailored to IVF-Flat retrieval. A selection checker reduces its score-dependent claim to a random evaluation of the corresponding virtual ranking-key array. The key identity exposes a random evaluation of the shifted score array, which an inner Sumcheck over the embedding dimension reduces to an evaluation of the authenticated centroid or selected-vector data; the query contribution is computed from the public embedding. The same selection-to-score-to-data chain handles both coarse and fine search. Consequently, exact inner products are certified where they are consumed, without commitments or openings for the C coarse scores or the mL fine scores.

Avoiding global rearrangement. Verif proves selection completeness using one permutation-free top-t relation. A Boolean mask of weight t identifies the selected items. Each valid item receives a unique deterministic key whose high-order component is its shifted score and whose low-order component encodes the tie-breaking identifier. The prover supplies a private threshold and nonnegative gaps showing that every selected key is at least the threshold, whereas every unselected valid key is strictly below it. Key uniqueness and mask cardinality then imply that the mask contains exactly the top-t items. Coarse search instantiates this relation with $t = m$, all items valid, and centroid identifiers as tie breakers. Fine search uses $t = k$, authenticated validity bits, and global chunk identifiers; dummy slots impose no ordering condition. Thus, both stages avoid sorted score arrays, conditional swaps, and permutation arguments.

Constructing an efficient ordering interface. The top-t relation turns comparisons into range claims for its threshold and gaps, but their ranking keys span too many bits for per-value bit decomposition or a full-range lookup table. Verif decomposes each value into fixed-width limbs, proves its algebraic reconstruction, and batches limb membership checks against compact public range tables using a zero-knowledge-compiled limb-range argument [39]. All non-leading limbs share the table $[0, 2^w)$, while the leading limb uses a table matching its exact residual width. This leading-limb check, together with the field-size condition in Section 3.2, enforces canonical integer ranges and prevents modular wraparound. This range layer is shared by the coarse and fine stages. Our prototype instantiates its compact-table membership checks with batched tlookup [41], while Zerocheck binds the limbs to the original thresholds and gaps.

Authenticating routed data and outputs. Verif binds fine search to the authenticated snapshot using per-cluster commitments to vectors and metadata. The verifier first authenticates the disclosed directory entries against the manifest, while LogUp proves that their identifier–tag pairs match the coarse-selection mask. It then homomorphically folds the corresponding commitments into the virtual fine-search view, avoiding a new concatenated-subindex commitment or m independent openings per terminal claim. A final LogUp argument binds the returned score–identifier–digest records to the fine-selection mask.

For scoring dimension D_s, the dominant online prover work is $O(CD_s + mLD_s)$, matching the native IVF-Flat search footprint. Direct inner product uses $D_s = D$, while squared L2 uses $D_s = D + 1$; both therefore remain $O(CD + mLD)$ asymptotically. The PIOP verifier performs $O(D_s + m + k + \text{polylog}(CLD_s))$ field operations. Our prototype instantiates the PCS layer with PST13, a pairing-based multilinear PCS with a structured setup [32]. Its additive homomorphism allows the verifier to fold the constant-size commitments of the m selected clusters using $O(m)$ group operations. Compatible terminal claims are consolidated into a constant number of opening batches, each contributing $O(\log N_{\text{PCS}})$ group elements to the proof and $O(\log N_{\text{PCS}})$ pairing terms to verification for a committed domain of size N_{PCS}.

3 Background and Preliminaries

This section reviews the RAG retrieval semantics and cryptographic interfaces used by Verif. Throughout the paper, we write $[n] = \{0, \ldots, n-1\}$ and let $\mathbb{F} = \mathbb{F}_p$ denote the prime field underlying the proof system.

3.1 RAG Retrieval

Retrieval-augmented generation consists of indexing, retrieval, and generation. During indexing, documents are divided into text chunks, mapped to embedding vectors, and organized into a vector index. Given a query embedding, retrieval returns the top-k relevant chunks, which are incorporated into the LLM prompt. As in prior verifiable RAG systems, we treat embedding and index construction as offline preprocessing and focus on proving faithful execution of retrieval over an authenticated snapshot. Proving the embedding model and downstream LLM inference is outside our scope.

3.2 IVF-Flat and Fixed-Point Semantics

Let an embedding database contain N vectors of dimension D. IVF-Flat partitions them into C clusters with centroids $\{\boldsymbol{\mu}_c\}_{c\in[C]}$. Each cluster contains at most L vectors and is padded to exactly L slots; consequently, $N \le CL$. Unused slots are padded and marked by validity bits. Given a query q, the authorized retrieval policy specifies a scoring rule of the form

$$s(q, x) = \alpha(q) + \langle \psi(q), \phi(x) \rangle_{D_s},$$

where $\alpha(q)$ and $\psi(q)$ are publicly computable from the query, while $\phi(x)$ is authenticated as part of the snapshot. Coarse search computes

$$s_c(c) = s(q, \mu_c)$$

for all C centroids and selects the top-m clusters. Fine search then computes

$$s_f(c, \ell) = s(q, X_c[\ell])$$

for every slot in the selected clusters and returns the top-k valid chunks.

For maximum inner-product search, $\alpha(q) = 0$, $\psi(q) = q$, $\phi(x) = x$, and $D_s = D$. Squared L2 distance admits the same interface. If quantized coordinates lie in a public interval $[Z_{\min}, Z_{\max}]$, let

$$M = D(Z_{\max} - Z_{\min})^2,$$

and define

$$\psi(q) = (q, 1), \qquad \phi(x) = (2x, -\|x\|_2^2),$$
$$\alpha(q) = M - \|q\|_2^2.$$

Then

$$s(q, x) = M - \|q - x\|_2^2 \in [0, M],$$

so maximizing $s(q, x)$ is exactly equivalent to minimizing squared L2 distance. This lifting uses $D_s = D + 1$. For L2-normalized embeddings, cosine similarity is also covered because

$$\|q - x\|_2^2 = 2 - 2\langle q, x \rangle.$$

Thus, Verif supports direct inner product, squared L2, and normalized cosine retrieval through the same scoring interface. Its scoring work is

$$O(CD_s + mLD_s),$$

which remains $O(CD + mLD)$ for these instantiations. The returned top-k is exact within the candidate domain selected by coarse search. Unlike IVF-PQ, IVF-Flat retains complete vectors and introduces no additional product-quantization error within this domain.

Fixed-point quantization. Proof systems operate over finite fields, whereas embeddings are typically real-valued. Following the affine quantization framework adopted by zkCNN [18, 27], we use one global public affine map for the database and queries. Let $U = 2^b - 1$, and let $v_{\min}$ and $v_{\max}$ be the public bounds fixed during snapshot preparation. A real-valued coordinate z is represented as

$$\widehat{z} = \text{clip}_{[0,U]}\left(\text{round}\left(\frac{(z - v_{\min})U}{v_{\max} - v_{\min}}\right)\right).$$

The same map is applied to database and query coordinates, while centroids are constructed in the resulting integer space. Before rounding, the common translation cancels from pairwise differences and the positive scale preserves squared-L2 ordering. Verif certifies the resulting integer retrieval semantics; the effect of rounding relative to floating-point retrieval is evaluated experimentally. Our implementation uses $b = 16$, giving coordinates in $[0, 2^{16} - 1]$.

An integer z is canonically embedded into $\mathbb{F}_p$ as

$$\iota(z) = \begin{cases} z, & z \ge 0, \\ p + z, & z < 0. \end{cases}$$

This encoding is algebraically exact and introduces no additional accuracy loss. Let $B_{\max}$ bound the absolute value of every integer intermediate, including scores, ranking keys, thresholds, and gaps. We require $2B_{\max} < p$, ensuring that accepted field relations have an unambiguous signed-integer interpretation and cannot wrap modulo p. Deterministic identifiers resolve equal scores.

3.3 Proof-System Interfaces

PIOPs and compilation. A zero-knowledge argument provides completeness, knowledge soundness, and zero knowledge relative to its public statement and permitted leakage. We describe Verif as a public-coin polynomial interactive oracle proof (PIOP) [3]: snapshot polynomials are preprocessing oracles, whereas masks, limbs, and lookup witnesses are query-time oracles. A polynomial commitment scheme (PCS) [23, 32] compiles these oracles into commitments and their terminal queries into evaluation proofs. We reserve *oracle* and *evaluation* for the PIOP level, and *commitment* and *opening* for its PCS realization.

The zero-knowledge compiler masks private oracle evaluations, including private scalar and terminal data evaluations, before a hiding PCS commits to the resulting oracles. Fiat–Shamir [11] then makes the public-coin protocol non-interactive in the random-oracle model. We denote this compiler by ZKComp; its masking and compilation are specified in Appendix A.3.

Multilinear extensions and Sumcheck. For $a \in \mathbb{F}^{2^\ell}$, its multilinear extension is

$$\widetilde{a}(\mathbf{r}) = \sum_{\mathbf{b}\in\{0,1\}^\ell} a(\mathbf{b})\widetilde{\mathrm{eq}}(\mathbf{r},\mathbf{b}),$$

where $\widetilde{\mathrm{eq}}$ is the multilinear equality polynomial. Sumcheck [29] reduces a bounded-degree hypercube sum to one random evaluation, with prover work linear in the domain and logarithmic verifier work. Zerocheck [5] uses such a reduction to prove

$$F(\mathbf{b}) = 0 \qquad \forall \mathbf{b} \in \{0,1\}^\ell,$$

rather than asserting that F is the zero polynomial.

Polynomial commitments. The PCS used by Verif is hiding, extractably binding, and additively homomorphic:

$$\mathsf{Com}\left(\sum_i \alpha_i f_i\right) = \sum_i \alpha_i \mathsf{Com}(f_i).$$

This lets the verifier derive commitments to public linear combinations of per-cluster preprocessing oracles, avoiding a new commitment to the concatenated selected subindex. The concrete PCS and batching strategy are described with the implementation.

LogUp and range lookups. For lookup values $\{q_i\}$, table values $\{t_j\}$, and multiplicities $\{\mu_j\}$, LogUp [15, 33] proves

$$\sum_i \frac{1}{\gamma + q_i} = \sum_j \frac{\mu_j}{\gamma + t_j},$$

using inverse-product and sum relations verified by Zerocheck and Sumcheck. We use it for coarse-to-fine routing and output binding. For range checks, we follow Lasso's structured-table decomposition but instantiate each limb-membership proof with batched tlookup. This certifies fixed-width limbs against compact tables without bit-decomposing complete ranking keys or constructing tables over their full range.

Section 6 presents the unmasked PIOP-level reductions and states the PCS realization only for nontrivial selected-cluster folding. Masking, transcript ordering, batching, and complete security definitions appear in the appendix.

4 System and Security Model

Participants and workflow. We consider a data owner, a retrieval server, and a client. The data owner constructs and certifies a fixed-shape IVF-Flat snapshot together with an authorized retrieval policy. The server stores the private snapshot and the information required to open its commitments. For each retrieval, the client identifies an accepted snapshot, submits a public quantized query, and requests an authorized value of k. The verifier authenticates the manifest, checks the query and parameter bounds, and derives the prescribed probe budget m. The server returns a canonical routing list I_{sel}, an ordered result $R_{\mathrm{out}} = ((u_r, s_r, T_r))_{r\in[k]}$, and a Verif proof.

Authenticated snapshot. Let $N_{\mathrm{cap}} = CL$. A snapshot contains C centroids and C clusters, each represented by L aligned slots containing a vector representation, a validity bit, a globally unique chunk identifier, and a text digest. The authorized policy Π fixes the scoring interface $(D_s, \alpha, \psi, \phi)$ from Section 3.2, and the preprocessing commitments authenticate the corresponding scoring rows $\phi(\mu_c)$ and $\phi(X_c[\ell])$. This definition is metric independent: each snapshot uses the scoring map authorized by Π, and the squared-L2 lift is only one instantiation of the common interface. The signed manifest binds the index dimensions, scoring and quantization parameters, centroid commitment, authenticated commitment-directory root, tag-polynomial commitment, snapshot identifier and version, and authorized retrieval policy. During preprocessing, the trusted data owner certifies coordinate bounds, Boolean validity, identifier ranges and uniqueness, canonical dummy slots, the prescribed scoring representation, and the correspondence between valid digests and canonical text. The online protocol treats these certified objects as fixed preprocessing oracles and proves only the query-dependent retrieval computation; Appendix A gives the complete well-formedness conditions and authenticated statement.

Adversary and guarantee. The data owner is trusted only to certify the intended snapshot and policy, whereas the retrieval server is fully malicious: it may fabricate scores, omit candidates, reduce the probe budget, alter routing, or substitute records. Acceptance establishes exact execution of the prescribed quantized IVF-Flat semantics. Coarse search ranks all C authenticated centroids and selects exactly the top m; I_{sel} contains precisely those clusters; fine search ranks every valid slot in the routed clusters and returns exactly the top k under the authorized deterministic tie-breaking rule. The returned identifiers, scores, and text digests are bound to the same snapshot. This guarantee does not cover corpus quality, offline embedding or clustering, full-database exact nearest-neighbor search, or downstream LLM inference. Denial of service is outside our scope.

Privacy and leakage. Besides the public query, policy, and snapshot commitments, Verif reveals I_{sel}, the authenticated directory entries for those clusters, and therefore the coarse access pattern, together with the returned scores, identifiers, and text. It hides all other database contents and scores, the fine-selection mask, and the private thresholds and gaps. Appendix A formalizes this leakage, and Appendix B gives the corresponding security definitions. Section 5 develops a common commitment-eliding kernel for the two ranking stages.

5 A Commitment-Eliding Proof Kernel

Coarse and fine search appear to be different computations: one ranks centroids, while the other ranks vectors in the routed clusters. Algebraically, however, both perform the same task. They derive exact scores from authenticated rows, encode deterministic ranking keys, and select the highest-ranked t eligible items. A conventional modular proof would expose the boundary between scoring and selection by committing to score and key polynomials, often together with a sorted or permuted copy. Native IVF-Flat creates none of these representations.

Verif instead defines one proof kernel that keeps scores and keys virtual until they are consumed at a random evaluation point. The

kernel combines exact scoring, a validity-aware top-t relation, limb-based range arguments, and a terminal reduction to authenticated data. Section 6 invokes it once for coarse search and once for fine search.

For a logical domain $[n]$, let $\ell_n = \lceil \log_2 n \rceil$, and let χ_n select its canonical embedding in $\{0,1\}^{\ell_n}$. Arrays are zero-extended outside their logical domains. The two item domains are

$$\begin{aligned} \Omega_c &= [C], & \chi_c &= \chi_C, \\ \Omega_f &= [m] \times [L], & \chi_f(j,\ell) &= \chi_m(j)\chi_L(\ell). \end{aligned}$$

We write $\widetilde{a}$ for the multilinear extension of a zero-extended array a.

5.1 Virtual Exact Scoring

Native IVF-Flat computes each score and immediately consumes it during selection. Committing to the complete score arrays would add query-dependent PCS work over C coarse items and mL fine candidates, even though every score is already determined by the public query and authenticated data. Our first optimization is to retain this native data flow: a score is a derived function, not an independent witness oracle.

Let $q_s = \psi(q) \in \mathbb{Z}^{D_s}$ be the public scoring representation from Section 3.2. For $\rho \in \{c, f\}$, define the authenticated stage tensors

$$A_c(i,d) = \phi(\mu_i)[d], \qquad A_f((j,\ell),d) = \phi\big(X_{I_{\mathrm{sel}}[j]}[\ell]\big)\,[d].$$

Thus A_c is the policy-defined scoring tensor for the centroids, whereas $A_f = \phi(X_{\mathrm{sel}})$ is that of the logical routed-vector view. The latter is not committed as a concatenated query-time oracle; Section 6.2 derives its required evaluations from the authenticated per-cluster commitments.

Let $S_{\min}$ and $S_{\max}$ be the public score bounds. For an item $z \in \Omega_\rho$, define

$$\begin{aligned} s_\rho(z) &= \alpha(q) + \sum_{d \in [D_s]} q_s[d] A_\rho(z,d), \\ s^\circ_\rho(z) &= s_\rho(z) - S_{\min}, \\ \kappa_\rho(z) &= U_\rho s^\circ_\rho(z) + (U_\rho - 1)\chi_\rho(z) - \mathrm{id}_\rho(z). \end{aligned} \tag{1}$$

Here

$$\begin{aligned} U_c &= 2^{\ell_C}, \qquad \mathrm{id}_c(i) = i, \\ U_f &= 2^{\ell_N}, \quad \mathrm{id}_f(j,\ell) = u_{\mathrm{sel}}(j,\ell), \quad \ell_N = \lceil \log_2 N_{\mathrm{cap}} \rceil. \end{aligned}$$

The high-order component of κ_ρ ranks scores, while its suffix gives priority to the smaller centroid or global chunk identifier. Coarse indices are distinct, and every valid chunk has a globally unique identifier; hence all eligible keys are distinct. This turns the authorized tie-breaking rule into ordinary integer ordering.

Let $B_s = \lceil \log_2(S_{\max} - S_{\min} + 1) \rceil$. Coarse keys fit in $B_c = B_s + \ell_C$ bits, and valid fine keys fit in $B_f = B_s + \ell_N$ bits. At a padded item point r_ρ, Equation 1 becomes

$$\begin{aligned} \widetilde{s}_\rho(r_\rho) &= \alpha(q)\widetilde{\chi}_\rho(r_\rho) + \sum_{d \in \{0,1\}^{\ell_{D_s}}} \widetilde{A}_\rho(r_\rho,d) q_s[d], \\ \widetilde{s}^\circ_\rho(r_\rho) &= \widetilde{s}_\rho(r_\rho) - S_{\min}\widetilde{\chi}_\rho(r_\rho), \\ \widetilde{\kappa}_\rho(r_\rho) &= U_\rho \widetilde{s}^\circ_\rho(r_\rho) + (U_\rho - 1)\widetilde{\chi}_\rho(r_\rho) - \widetilde{\mathrm{id}}_\rho(r_\rho), \end{aligned} \tag{2}$$

where q_s is canonically zero-padded beyond D_s. These identities allow any later claim about a score or key to be reduced directly to the authenticated stage tensor.

5.2 A Unified, Permutation-Free Top-t Relation

Proving that a mask has weight t establishes only that it selects t items; it does not establish that they are the highest-ranked ones. A generic proof may sort the keys inside the circuit or accept an externally sorted array and verify a permutation plus a boundary condition. Both approaches construct global representations absent from native IVF-Flat, and both costs are incurred twice.

Verif instead separates selected and unselected keys with one private threshold. Let a_ρ be the selection mask and let v_ρ indicate eligibility. The coarse instance uses $t_c = m$ and $v_c = 1$; the fine instance uses $t_f = k$ and $v_f = v_{\mathrm{sel}}$. Both satisfy

$$\begin{aligned} &a_\rho(z)(a_\rho(z) - 1) = 0, \\ &a_\rho(z)(1 - v_\rho(z)) = 0, \qquad \sum_{z \in \Omega_\rho} a_\rho(z) = t_\rho, \\ &\qquad g_\rho(z) = a_\rho(z)(\kappa_\rho(z) - \tau_\rho) \\ &\qquad\qquad + (v_\rho(z) - a_\rho(z))(\tau_\rho - 1 - \kappa_\rho(z)), \end{aligned} \tag{3}$$

where τ_ρ is a private threshold.

For a valid item, $a_\rho(z) = 1$ gives $g_\rho(z) = \kappa_\rho(z) - \tau_\rho$, while $a_\rho(z) = 0$ gives $g_\rho(z) = \tau_\rho - 1 - \kappa_\rho(z)$. Requiring $g_\rho(z) \geq 0$ therefore places every selected key on or above the threshold and every unselected valid key below it. For an invalid fine-stage slot, the validity constraint forces $a_f = 0$, after which the same equation gives $g_f = 0$ and imposes no ordering condition. Booleanity, mask weight, and key uniqueness then imply that a_ρ is exactly the top-t_ρ eligible set. For the fine stage, these relations are satisfiable only when the routed view contains at least k valid slots; otherwise no accepting selection mask exists.

Compact range arguments. Finite-field arithmetic does not itself prove that τ_ρ and g_ρ are nonnegative integers. A table covering all B_ρ-bit values would have size 2^{B_ρ}, while bit-decomposing every gap would introduce B_ρ Boolean witnesses per item. We instead decompose only the threshold and gaps into fixed-width limbs.

For public limb width w, let $\lambda_B = \lceil B/w \rceil$ and $b_B^{\mathrm{top}} = B - w(\lambda_B - 1)$. We write $\mathrm{Range}_B(z)$ when

$$z = \sum_{r=0}^{\lambda_B - 1} 2^{wr} z^{(r)}, \qquad z^{(r)} \in \begin{cases} [0, 2^w), & r < \lambda_B - 1, \\ [0, 2^{b_B^{\mathrm{top}}}), & r = \lambda_B - 1. \end{cases}$$

The relation requires $\mathrm{Range}_{B_\rho}(\tau_\rho)$ and $\mathrm{Range}_{B_\rho}(g_\rho(z))$ for all items. The limb-range argument batches the ordinary limbs against the shared table $[0, 2^w)$ and uses the exact residual-width table for the leading limb. Thus the range cost is linear in the number of limbs rather than the full key range, while the field-size condition of Section 3.2 prevents modular aliases.

This relation is permutation-free and identical in both stages. Only the domain, eligibility vector, target cardinality, identifier, and key width change.

5.3 Closing Selection Claims to Exact Inner Products

The prover fixes a_ρ, the threshold limbs, and the gap limbs before the corresponding challenges are sampled. Zerocheck and Sumcheck establish Booleanity, eligibility, and mask weight in Equation 3; the limb-range argument establishes the limb ranges. No score or ranking-key oracle is submitted.

After reconstructing the threshold and gaps from their limbs, define the padded residual

$$F_\rho(x) = \sum_r 2^{wr}\widetilde{g}_\rho^{(r)}(x) - \widetilde{a}_\rho(x)\big(\widetilde{\kappa}_\rho(x) - \tau_\rho\widetilde{\chi}_\rho(x)\big) - \big(\widetilde{v}_\rho(x) - \widetilde{a}_\rho(x)\big)\big((\tau_\rho - 1)\widetilde{\chi}_\rho(x) - \widetilde{\kappa}_\rho(x)\big). \tag{4}$$

Here $\widetilde{v}_c = \widetilde{\chi}_c$. On logical Boolean points, Equation 4 is precisely the threshold–gap relation; on padding points, all terms vanish. A randomly weighted Zerocheck reduces these pointwise identities to a random item point r_ρ. The terminal relation now requires $\widetilde{\kappa}_\rho(r_\rho)$, but there is deliberately no key oracle from which to open it.

Let $\xi_\rho = \widetilde{\chi}_\rho(r_\rho)$ and $\widehat{\imath}_\rho = \widetilde{\mathrm{id}}_\rho(r_\rho)$. Using Equation 2, the verifier peels the virtual key and invokes Sumcheck only over the scoring dimension:

$$\begin{aligned}\widehat{\kappa}_\rho &= U_\rho\widehat{s}_\rho^{\circ} + (U_\rho - 1)\xi_\rho - \widehat{\imath}_\rho,\\ \widehat{\sigma}_\rho^{\mathrm{ip}} &= \widehat{s}_\rho^{\circ} + (S_{\min} - \alpha(q))\xi_\rho\\ &= \sum_{d\in\{0,1\}^{\ell_{D_s}}} \widetilde{A}_\rho(r_\rho, d)q_s[d]\\ &\xrightarrow{\text{Sumcheck}} \widetilde{A}_\rho(r_\rho, r_D)\widetilde{q}_s(r_D).\end{aligned} \tag{5}$$

The verifier computes $\widetilde{q}_s(r_D)$. In the coarse stage, the remaining private evaluation is authenticated against the centroid commitment. In the fine stage, the vector and identifier evaluations are authenticated against the folded commitments derived in Section 6.2.

The scoring branch of the claim flow is therefore

$$\boxed{\widetilde{\kappa}_\rho(r_\rho) \Longrightarrow \widetilde{s}_\rho^{\circ}(r_\rho) \Longrightarrow \widetilde{s}_\rho(r_\rho) \Longrightarrow \widetilde{A}_\rho(r_\rho, r_D)}.$$

The kernel introduces mask, threshold-limb, gap-limb, and lookup witnesses, but no explicit score, ranking-key, sorted-score, or permuted-score polynomial. Consequently, its PCS realization adds no commitment or opening for any of these omitted arrays.

For stage ρ, the prover work is

$$O\Big(|\Omega_\rho|D_s + |\Omega_\rho|\lambda_{B_\rho}\Big).$$

With fixed score and limb widths, the two invocations have dominant cost $O(CD_s + mLD_s) = O(CD + mLD)$, matching the native IVF-Flat scoring footprint.

5.4 Why LogUp Preserves Virtuality

After a top-t mask is proved, the protocol must expose a prescribed projection of its support: coarse selection exposes routed clusters, and fine selection exposes returned records. A fixed-table lookup is well suited to the static limb tables above, but these projections are query dependent. Encoding them through a conventional lookup or permutation would first materialize the selected keys or tuple arrays, reintroducing the boundary that Section 5.1 removes.

LogUp instead supports a mask-weighted projection directly. For a source tuple $T(z)$, public projected tuples Q, and random tuple encoding enc_β, the common relation is

$$\sum_{y\in Q}\frac{1}{\gamma - \mathrm{enc}_\beta(y)} = \sum_{z\in\Omega}\frac{a(z)}{\gamma - \mathrm{enc}_\beta(T(z))}. \tag{6}$$

Section 6 uses two instances:

$$\begin{aligned}\text{routing:}\quad & a = a_c, \qquad T(c) = (c, \eta_c),\\ & Q = \{(I_j, \eta_{I_j})\}_{j\in[m]},\\ \text{output:}\quad & a = a_f, \qquad T(z) = (\kappa_f(z), h_{\mathrm{sel}}(z)),\\ & Q = \{(\kappa_r, h_r)\}_{r\in[k]}.\end{aligned}$$

The selected side of Equation 6 requires only random evaluations of the existing mask and tuple fields. In the output instance, its virtual κ_f evaluation is discharged by Equation 5 at the LogUp terminal point. This point need not equal the fine-selection terminal point: both claims use the same virtual-key-to-authenticated-data interface.

This division of labor is deliberate. The limb-range argument proves membership in fixed structured range tables; LogUp proves query-dependent projections of already certified selection masks. Together they preserve the virtual score representation across selection, routing, and output binding.

6 Authenticating the Coarse-to-Fine Execution

Section 5 proves one ranking stage relative to an authenticated data tensor and exposes its result only as a selection mask. We now compose two kernel invocations while preserving authentication across their query-dependent boundary:

$$\begin{aligned}\text{coarse kernel} &\longrightarrow \text{routing projection}\\ &\longrightarrow \text{folded selected view}\\ &\longrightarrow \text{fine kernel} \longrightarrow \text{output projection}.\end{aligned}$$

The resulting protocol authenticates every transition without materializing coarse scores, fine scores, ranking keys, or a concatenated selected subindex.

We use a typed, injective bit-string encoding Canon and domain-separated collision-resistant hash-to-field functions $H, H_{\mathrm{tag}} : \{0,1\}^* \to \mathbb{F}_p$ for returned-text digests and directory tags, respectively; $\|$ denotes concatenation of canonical encodings.

6.1 Binding Coarse Selection to Routing

The coarse invocation sets $A_c(i,\cdot) = \phi(\mu_i)$, $v_c = 1$, and $t_c = m$. By Section 5.2, its mask a_c selects exactly the top-m centroids; by Section 5.3, every score claim reduces to the authenticated centroid commitment.

The server must next disclose which clusters constitute the fine domain. Merely publishing m cluster identifiers is insufficient: they must be the support of the already proven a_c, rather than a new server-chosen set. Let

$$I_{\mathrm{sel}} = \big(I_{\mathrm{sel}}[0], \ldots, I_{\mathrm{sel}}[m-1]\big)$$

be the disclosed list. The verifier checks

$$0 \le I_{\mathrm{sel}}[0] < \cdots < I_{\mathrm{sel}}[m-1] < C,$$

which gives a canonical representation of the routed set.

For each cluster, the authenticated directory contains

$$\begin{aligned}\mathsf{Dir}[c] &= \big(\mathsf{Com}_{X,c}, \mathsf{Com}_{v,c},\\ &\qquad \mathsf{Com}_{u,c}, \mathsf{Com}_{h,c}\big),\\ \eta_c &= H_{\mathrm{tag}}\big(c \parallel \mathsf{Dir}[c]\big).\end{aligned}$$

The signed manifest binds both the directory root and the commitment to the global tag array. For each disclosed cluster, the verifier authenticates its directory entry and recomputes η_c.

Routing instantiates the masked projection of Equation 6. After a_c, I_{sel}, and the selected tags are fixed, the verifier samples $\alpha, \beta \in \mathbb{F}$ and proves

$$\sum_{j\in[m]} \frac{1}{\alpha - I_{\mathrm{sel}}[j] - \beta\eta_{I_{\mathrm{sel}}[j]}} = \sum_{c\in[C]} \frac{a_c(c)}{\alpha - c - \beta\eta_c}. \tag{7}$$

LogUp verifies the inverse-product identities and equality of the two sums using Zerocheck and Sumcheck. Crucially, its right side uses the same committed a_c as the coarse kernel; no second routing mask is accepted. Because a_c is Boolean with weight m and the public list is strictly increasing, Equation 7 binds the list exactly to the coarse-selected clusters. The tag binds each identifier to the complete tuple of commitments used below.

This reuse of a_c gives routing a narrow role: the coarse kernel proves that the correct clusters were selected, while LogUp proves that precisely those clusters were disclosed.

6.2 Folding the Authenticated Selected View

The selected clusters are query dependent, so their concatenation cannot be committed during preprocessing. Creating a new online commitment to the complete selected subindex would introduce another large commitment operation; opening all m cluster commitments separately at every terminal claim would instead increase proof and verification cost.

We avoid both choices through partial-evaluation folding. For $Y \in \{X, v, u, h\}$, define

$$Y_{\mathrm{sel}}(j, z) = Y_{I_{\mathrm{sel}}[j]}(z),$$

where $z = (\ell, d)$ with $d \in [D_s]$ for scoring rows and $z = \ell$ for metadata. At a fine-stage cluster point r_m, multilinearity gives

$$\begin{aligned} \widetilde{Y}_{\mathrm{sel}}(r_m, z) &= \sum_{j\in[m]} \widetilde{\mathrm{eq}}(r_m, j)\widetilde{Y}_{I_{\mathrm{sel}}[j]}(z), \\ \mathrm{Com}^{\mathrm{sel}}_Y(r_m) &= \sum_{j\in[m]} \widetilde{\mathrm{eq}}(r_m, j)\mathrm{Com}_{Y,I_{\mathrm{sel}}[j]}. \end{aligned} \tag{8}$$

The second equality follows from PCS additive homomorphism. The verifier derives the folded commitment directly from the authenticated directory entries; the prover does not submit a new selected-view commitment.

For a fine terminal point $r_f = (r_m, r_L)$, the vector opening uses the remaining point (r_L, r_D), while validity, identifier, and digest openings use r_L. Thus one folded opening per required data type replaces m independent cluster openings. Compatible openings are further batched by the PCS compiler.

Equation 8 is the bridge between the two kernel invocations: routing determines the folding coefficients, and the resulting commitments authenticate every private value consumed by fine selection and output binding.

6.3 Binding Fine Selection to Returned Context

The fine invocation sets $A_f = \phi(X_{\mathrm{sel}})$, $v_f = v_{\mathrm{sel}}$, and $t_f = k$. The folded commitments authenticate its vectors, validity bits, and global chunk identifiers. The unified kernel therefore proves that a_f selects exactly the top-k valid slots among all candidates in the routed clusters. Its virtual key claim again terminates at an authenticated vector evaluation through Equation 5.

It remains to prove that the server returns the records selected by a_f. Let

$$R_{\mathrm{out}} = \big((u_r, s_r, T_r)\big)_{r\in[k]}$$

be the public ordered output. The verifier checks $u_r \in [0, N_{\mathrm{cap}})$ and $s_r \in [S_{\min}, S_{\max}]$, and computes

$$\kappa_r = U_f(s_r - S_{\min}) + (U_f - 1 - u_r), \qquad h_r = H(\mathrm{Canon}(T_r)).$$

It separately checks $\kappa_0 > \kappa_1 > \cdots > \kappa_{k-1}$ in ordinary integer arithmetic.

After R_{out} and a_f are fixed, the verifier samples $\gamma, \delta \in \mathbb{F}$. The second masked projection proves

$$\sum_{r\in[k]} \frac{1}{\gamma - \kappa_r - \delta h_r} = \sum_{z\in\Omega_f} \frac{a_f(z)}{\gamma - \kappa_f(z) - \delta h_{\mathrm{sel}}(z)}. \tag{9}$$

The right side reuses the exact fine-selection mask, rather than a fresh output witness. Its digest evaluation is authenticated through the folded commitment of Section 6.2. Its ranking key remains virtual: at the LogUp terminal point, the verifier invokes Equation 5 again and reduces $\widetilde{\kappa}_f$ directly to the folded vector and identifier commitments.

Equation 9 establishes multiset equality, while the public strict-order check supplies the final order. The injective key encoding binds each public score and identifier, and collision resistance of H binds the returned canonical text to its authenticated digest. Consequently, output binding requires neither a committed fine-key array nor a permuted copy of the selected records.

6.4 Composition, Cost, and Security

The complete argument establishes four linked events:

$$\begin{aligned} E_c &: \ a_c = \mathrm{Top}_m(\kappa_c), \\ E_r &: \ I_{\mathrm{sel}} = \mathrm{Inc}(\mathrm{supp}(a_c)), \\ E_f &: \ a_f = \mathrm{Top}_k(\kappa_f; v_{\mathrm{sel}}), \\ E_o &: \ R_{\mathrm{out}} = \mathrm{Desc}(\mathrm{Record}(\mathrm{supp}(a_f))). \end{aligned}$$

Here $\mathrm{Top}_t(\kappa; v)$ denotes the indicator of the t highest-key eligible items; Inc, Record, and Desc denote increasing support enumeration, slot-to-record mapping, and descending key order, respectively. The two kernel invocations establish E_c and E_f; Equation 7 establishes E_r; and Equation 9, together with the public order check, establishes E_o. Directory authentication and Equation 8 ensure that all four events refer to one certified snapshot. Hence

$$E_c \wedge E_r \wedge E_f \wedge E_o \Longrightarrow \mathcal{R}_{\mathrm{IVF}},$$

relative to the fixed certified preprocessing oracles.

For fixed score and limb widths, the dominant prover work is $O(CD_s + mLD_s) = O(CD + mLD)$. The PIOP verifier performs $O(D_s + m + k + \mathrm{polylog}(CLD_s))$ field operations, while selected-cluster folding requires $O(m)$ group operations. Compatible terminal claims are consolidated into a constant number of PCS opening batches, so proof growth is logarithmic in the committed domain rather than linear in the database size.

Native-work optimality. In the explicit dense-vector model, evaluating all C centroid scores and all mL routed-vector scores

requires $\Omega(CD_s + mLD_s)$ arithmetic work. VERIF matches this lower bound while adding only $O(C\lambda_{B_c} + mL\lambda_{B_f})$ field work for the ordering ranges and no score- or key-oracle commitments. It therefore achieves native-work-optimal dominant complexity while eliminating proof-level score, key, and sorted-array materialization. This structural specialization directly reduces selection, commitment, and memory overhead, as quantified in Section 7.3.

THEOREM 1 (SECURITY OF THE VERIF PIOP). *Assume the employed Zerocheck/Sumcheck, limb-range, and LogUp protocols satisfy their standard completeness and knowledge-soundness guarantees, and that* ZKComp *is relation preserving, knowledge preserving, and simulation secure. For every fixed well-formed preprocessing instance, VERIF is a complete, knowledge-sound, and zero-knowledge PIOP for the online relation $\mathcal{R}_{\mathrm{IVF}}$ defined in Sections 4–6, relative to the leakage specified in Section 4.*

COROLLARY 2 (PCS AND FIAT–SHAMIR COMPILATION). *Assume the data-owner signature is EUF-CMA secure, the authenticated directory and employed hashes are binding and collision resistant, the additively homomorphic PCS is hiding and extractably binding, and the employed multi-round Fiat–Shamir knowledge-soundness theorem applies. PCS compilation followed by Fiat–Shamir yields a non-interactive zero-knowledge argument of knowledge for the authenticated retrieval relation $\mathcal{R}_{\mathrm{ret}}$.*

Appendix A specifies the complete message order, including commitment of challenge-dependent LogUp inverse oracles and transcript-bound rejection of zero denominators. Appendix B gives the formal security definitions and proofs of Theorem 1 and Corollary 2.

7 Implementation and Evaluation

We conduct a detailed evaluation of VERIF to determine whether fixed-point quantization preserves retrieval quality, whether its specialized proof design provides practical end-to-end performance and scalability, and how effectively its prover can be accelerated using a commodity GPU.

7.1 Implementation and Experimental Setup

We implement VERIF in C++14, with an OpenMP-based CPU backend and an optional CUDA backend. The GPU backend accelerates the dominant Sumcheck and BN254 multi-scalar multiplication operations while preserving the same protocol and serialized proof format as the CPU backend. We instantiate the multilinear polynomial commitments over BN254 using MCL and derive all verifier challenges through domain-separated Fiat–Shamir transcripts. Unless otherwise stated, vector coordinates are quantized to 16-bit integers ($b = 16$), while the limb-range arguments use 12-bit limbs ($w = 12$).

We conduct all proof-performance, scalability, and GPU experiments on an Ubuntu 22.04 server equipped with 64 AMD EPYC Rome vCPUs, 251 GiB of host memory, and NVIDIA RTX 3090 GPUs with 24 GiB of device memory each. At each matched comparison point, VERIF and Plonky2 use the same CPU-thread budget. Original SIFT and Cohere profiles use eight and 64 threads, respectively; all enlarged profiles use 64 threads. Each GPU experiment uses a single RTX 3090. The ranking-only study in Section 7.2 ran separately on Windows 11 with 16 GiB of memory; its full environment is archived.

We report steady-state online proving and verification time, excluding one-time index construction and offline preprocessing. Proof size denotes the complete serialized proof, host memory denotes peak resident set size (RSS), and GPU memory denotes peak device-memory consumption.

7.2 Impact of Quantization

VERIF certifies retrieval over integer coordinates, so we measure whether its 16-bit affine map changes retrieval rankings. We compare exact full-database squared-L_2 top-64 search before and after quantization on the one-million-vector SIFT and Cohere datasets, using all 10,000 and 1,000 queries, respectively. Both sides use float64 accumulation and deterministic (distance, DocID) tie-breaking; IVF routing and proof generation are excluded.

Table 1: Effect of 16-bit quantization on exact squared-L_2 retrieval.

Dataset	Q	Top-1	Ov.@64	ΔR@1	ΔSR@64
SIFT1M	10,000	1.000000	0.999917	0.0000	−0.0052
Cohere-1M	1,000	1.000000	0.999828	0.0000	+0.0031

Table 1 shows that quantization leaves all 11,000 top-1 results unchanged and preserves at least 99.9828% mean top-64 overlap. The Δ columns report quantized minus float retrieval against the published ground truth, in percentage points; the largest absolute set-recall change is 0.0052 points. No query coordinate is clipped.

Cohere provides cosine ground truth, whereas our evaluated policy is squared L_2. Its released vectors are not unit-normalized; float and quantized L_2 both obtain Recall@1 = 0.888 against the cosine list. Thus, the cosine–L_2 gap is a policy difference, not quantization loss. Under this fixed squared-L_2 policy, any larger IVF-to-exact gap therefore primarily reflects candidate filtering rather than quantization.

7.3 CPU Performance and Scalability

Datasets and baseline. We evaluate VERIF on two complementary datasets. SIFT1M contains 128-dimensional image descriptors and is a standard benchmark for evaluating approximate nearest-neighbor search. We additionally use Cohere-1M-Wikipedia-768d, whose 768-dimensional text embeddings more closely represent the vector dimensions and semantic content encountered in practical RAG services. All proof-performance experiments use squared L_2, realized through the exact $D + 1$ lifting of Section 3.2; the SIFT and Cohere scoring dimensions are therefore 129 and 769. SIFT natively uses squared L_2. Cohere provides cosine ground truth but is evaluated under the authorized squared-L_2 policy; this distinction and its quantization effect are discussed in Section 7.2. We fix $k = 64$ and scale the IVF index shape and probe budget with the database size. At every comparison point, VERIF and Plonky2 use the same index configuration and retrieval parameters. This subsection reports CPU results; GPU acceleration is evaluated separately.

No existing system directly implements the same authenticated, complete-vector IVF-Flat relation as VERIF. We therefore build a

matched baseline using the general-purpose Plonky2 [35], whose 64-bit field arithmetic and custom gates suit fixed-shape scoring and selection. To establish a stronger baseline for comparison, we incorporate the applicable index-independent optimization shared by V3DB and VeriRAG [26, 36]: moving sorting outside the circuit and checking the externally sorted fine-stage witness with a transcript-bound multiset relation. We further prune unselected slots, use bounded partial coarse selection, and reuse fixed-shape circuits. Both systems use the same indexes, retrieval parameters, and CPU-thread budgets.

End-to-end performance. Table 2 reports prover time, verifier time, and serialized proof size on both datasets. Times are in seconds and proof sizes are in MiB. An em dash indicates that larger Plonky2 configurations were not attempted after its first OOM.

Table 2: CPU performance on SIFT1M and Cohere-1M-Wikipedia-768d.

	Plonky2			VERIF		
N	Prover time (s)	Verifier time (s)	Proof size (MiB)	Prover time (s)	Verifier time (s)	Proof size (MiB)
SIFT1M, $D = 128$ (8 threads)						
8K	7.077	0.010	0.156	2.291	0.622	0.887
16K	28.043	0.011	0.176	2.487	0.645	0.927
32K	59.969	0.012	0.183	2.849	0.643	0.967
65K	244.162	0.012	0.198	3.434	0.661	1.007
131K		OOM		4.504	0.751	1.047
262K		–		6.365	0.892	1.088
524K		–		7.492	0.915	1.107
1M		–		9.654	0.898	1.127
Cohere-1M-Wikipedia-768d, $D = 768$ (64 threads)						
8K	13.340	0.011	0.181	1.886	0.646	0.889
32K	70.149	0.012	0.195	2.505	0.677	0.969
65K	280.316	0.014	0.217	3.242	0.746	1.009
131K		OOM		4.740	0.960	1.049
1M		–		10.795	1.329	1.129

VERIF consistently reduces the dominant proving cost. At the largest configuration completed by both systems, $N = 65{,}536$, VERIF is 71.1× faster on SIFT and 86.5× faster on Cohere. Plonky2 runs out of memory at $N = 131{,}072$ on both datasets, whereas VERIF completes the one-million-vector instances in 9.654 and 10.795 seconds, respectively.

At the largest jointly completed configurations, VERIF targets the dominant server-side proving bottleneck: it cuts proving time by 71.1×–86.5× while retaining approximately 0.6–1.3-second verification and approximately 1 MiB proofs, both practical for networked RAG services. Plonky2 offers millisecond verification and smaller proofs at those points, but reaches the 251 GiB host-memory limit at 131K vectors, whereas VERIF completes both one-million-vector workloads.

Memory efficiency. Figure 3 compares peak resident memory before Plonky2 reaches the 251 GiB host-memory limit. At $N = 65{,}536$, VERIF uses only 0.626 GiB on SIFT, compared with 68.783 GiB for Plonky2, reducing peak memory by 99.1%. On Cohere, the corresponding values are 3.016 and 125.257 GiB, a 97.6% reduction. At $N = 131{,}072$, VERIF still requires only 1.305 GiB on SIFT and 6.070 GiB on Cohere, while Plonky2 is terminated by the host OOM killer.

Larger databases. After Plonky2 reaches its memory limit, we further evaluate VERIF on enlarged indexes. We construct these

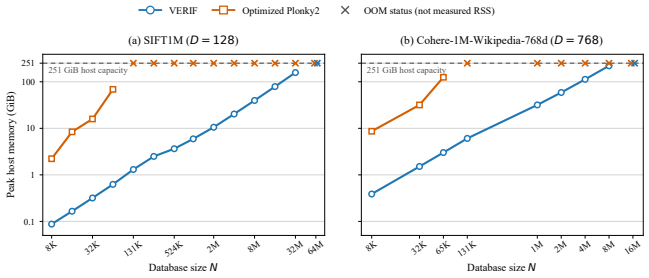

Figure 3: Peak host memory of VERIF and Plonky2 before Plonky2 reaches the 251 GiB system limit.

snapshots by deterministically repeating the quantized one-million-vector databases and their cluster assignments while assigning fresh DocIDs; they are system-scale stress tests rather than larger natural corpora. To isolate database size from the online scan budget, we keep $C = 8{,}192$, $mL = 65{,}536$, and $k = 64$ fixed while increasing the authenticated snapshot size. Table 3 reports the resulting CPU performance.

Table 3: VERIF CPU performance on enlarged indexes.

N	Prover time (s)	Verifier time (s)	Proof size (MiB)	Peak RSS (GiB)
SIFT1M, $D = 128$ (64 threads)				
2M	5.652	0.770	1.126	10.56
4M	5.677	0.780	1.125	20.31
8M	5.498	0.804	1.124	39.72
16M	5.579	0.815	1.123	78.83
32M	5.898	0.875	1.123	157.90
64M		OOM		
Cohere-1M-Wikipedia-768d, $D = 768$ (64 threads)				
2M	10.711	1.328	1.127	58.88
4M	10.748	1.331	1.126	112.84
8M	11.573	1.483	1.126	220.82
16M		OOM		

On SIFT, VERIF's prover time remains between 5.50 and 5.90 seconds from 2M to 32M vectors, while verification time and proof size remain nearly constant. Cohere exhibits the same behavior, taking 10.71–11.57 seconds from 2M to 8M vectors; the 0.8% decrease from the separate 1M profile to 2M, measured once with fixed C and mL but different (m, L), should not be interpreted as an N-induced speedup. Peak RSS grows with the authenticated snapshot size, leading to an observed OOM at 64M for SIFT. For Cohere, extrapolating the measured 4M–8M memory trend gives approximately 437 GiB at 16M, exceeding the available 251 GiB.

These results reflect VERIF's separation between authenticated snapshot size and query-dependent proof work: with $C = 8{,}192$ and $mL = 65{,}536$ fixed, increasing N raises storage and RSS but leaves the dominant $O(CD + mLD)$ online work unchanged. Small non-monotonic differences, such as Cohere 1M–2M, should be treated as single-run profile variation rather than N-induced speedups. This decoupling is a key advantage of VERIF: its online proving cost is governed primarily by the routed workload rather than the total authenticated snapshot size.

7.4 GPU Acceleration

All preceding results use the CPU prover. We additionally implement a CUDA backend that accelerates the dominant Sumcheck and BN254 multi-scalar multiplication (MSM) paths using a single RTX 3090. Witness preparation, protocol orchestration, and several auxiliary operations remain on the CPU, leaving additional headroom for a more fully GPU-resident implementation. Because Sumcheck and MSM are protocol-level primitives, these acceleration components can serve as reusable building blocks for future proof systems using the same primitives. All reported runs execute the GPU Sumcheck and MSM paths without CPU fallback. The observed end-to-end speedups are bounded by the remaining CPU fraction; workloads with larger Sumcheck domains or MSM batches and less CPU-side preprocessing can therefore obtain substantially greater gains from the same GPU components.

Table 4 compares matched CPU and GPU proving times. Speedup is computed as CPU time divided by GPU time.

Table 4: GPU acceleration of the VERIF prover.

N	CPU prover time (s)	GPU prover time (s)	Speedup
SIFT1M, $D = 128$ (8 threads)			
8K	2.291	1.925	1.19×
16K	2.487	2.071	1.20×
32K	2.849	2.325	1.23×
65K	3.434	2.700	1.27×
131K	4.504	3.229	1.40×
262K	6.365	3.976	1.60×
524K	7.492	4.503	1.66×
1M	9.654	5.494	1.76×
Enlarged-index regime (64 threads)			
2M	5.652	4.849	1.17×
4M	5.677	4.773	1.19×
8M	5.498	4.732	1.16×
16M	5.579	4.830	1.16×
32M	5.898	5.066	1.16×
Cohere-1M-Wikipedia-768d, $D = 768$ (64 threads)			
8K	1.886	1.625	1.16×
32K	2.505	2.277	1.10×
65K	3.242	2.889	1.12×
131K	4.740	4.103	1.16×
1M	10.795	9.078	1.19×
Enlarged-index regime (64 threads)			
2M	10.711	9.161	1.17×
4M	10.748	9.159	1.17×
8M	11.573	9.434	1.23×

Under the original index-scaling profiles, GPU acceleration becomes increasingly effective as the SIFT workload grows, reducing the one-million-vector proving time from 9.654 to 5.494 seconds and providing a 1.76× speedup. Cohere achieves 1.10×–1.19× speedups over the same range and completes the one-million-vector workload in 9.078 seconds.

The enlarged profiles fix $C = 8{,}192$, $mL = 65{,}536$, and $k = 64$, thereby isolating snapshot-scale growth under a constant online scan workload. GPU proving remains within 4.73–5.07 seconds for SIFT and 9.16–9.43 seconds for Cohere, corresponding to 1.16×–1.19× and 1.17×–1.23× speedups over the matched CPU runs. Consequently, a single commodity GPU proves the 32M-vector SIFT and 8M-vector Cohere instances in 5.066 and 9.434 seconds, respectively, without changing the protocol or serialized proof format.

Summary. Taken together, these experiments show that Verif improves the practicality of verifiable IVF-Flat retrieval along the two dimensions that most directly determine deployment cost: prover latency and memory consumption. The isolated study in Section 7.2 confirms that 16-bit quantization negligibly affects exact rankings. At the largest configuration completed by both systems, Verif reduces CPU proving time by 71.1×–86.5× and peak host memory by 97.6%–99.1% relative to the optimized Plonky2 baseline. Verif also completes both one-million-vector workloads with approximately 1 MiB proofs and at most 1.33 seconds of verification, whereas Plonky2 runs out of memory at 131K vectors.

The online proof cost remains stable when the authenticated indexes are enlarged: the largest successful SIFT and Cohere stress tests contain 32M and 8M vectors, respectively. With one RTX 3090, their proofs are generated in 5.07 and 9.43 seconds, and both natural one-million-vector workloads also complete in under ten seconds. Although host memory continues to scale with the in-memory snapshot representation, the proof computation itself remains practical and benefits from further GPU acceleration. These results indicate that Verif's specialized proof design substantially lowers the main computational barriers to deploying verifiable retrieval in RAG-as-a-Service systems.

8 Related Work

Zero-knowledge machine learning. Zero-knowledge proofs have been developed for decision trees, logistic regression, CNN inference and training, and federated learning [1, 2, 4, 8, 9, 13, 16, 24, 27, 28, 37, 40, 43, 44, 46, 47]. General-purpose systems such as Mystique, Zeno, and ZKML [6, 10, 43] compile broad classes of machine-learning computations into generic proof backends. Although these systems can in principle arithmetize an IVF-Flat retriever, their abstractions target fixed tensor programs rather than authenticated, query-dependent access to an external vector index. A direct encoding would materialize the coarse and fine scores as circuit witnesses and express selection and routing using generic comparison, sorting, or memory gadgets, thereby losing the native coarse-to-fine structure of IVF-Flat. More recent systems, including zkLLM and zkGPT [38, 41], substantially improve proofs of transformer inference. They certify inference on an already constructed prompt, however, and therefore do not establish that its retrieved context was correctly obtained from an authenticated snapshot. These systems are complementary to verifiable retrieval rather than substitutes for it.

Zero-knowledge proofs for RAG retrieval. V3DB [36] provides audit-on-demand proofs for fixed-shape IVF-PQ retrieval over committed snapshots, using multiset equality/inclusion and boundary checks to avoid direct in-circuit sorting and random access. Concurrent VeriRAG [26] also targets IVF-PQ, combining per-cluster commitments, GKR, and lookup arguments; it verifies top-k selection through permuted distance arrays, difference checks, and range proofs. The transferable off-circuit sorting idea is already included in our optimized baseline. Their PQ-specific codebook and ADC-lookup optimizations do not transfer directly to IVF-Flat, which has no PQ codes and must certify a full D-dimensional score per candidate. VeriRAG's chunk merging would also change the matched index and retrieval workload. Concurrent zkRAG [20] instead targets HNSW, whose graph-traversal, membership, and

priority-queue relations do not arise in fixed-shape IVF-Flat. VeriANN [17] instead combines LSH, PIR, and authenticated garbled circuits to provide query privacy and retrieval integrity under a two-server non-colluding model, orthogonal to the public-query, single-server ZK setting considered here. In contrast, Verif targets the complete IVF-Flat coarse-to-fine semantics. Its commitment-eliding score–selection reduction certifies exact full-vector inner products where they are consumed, while a common permutation-free threshold–gap relation proves both selection stages. This avoids score and permuted-score oracles while retaining dominant prover work of $O(CD + mLD)$.

9 Conclusion

A central obstacle to verifiable retrieval is that proof-specific bookkeeping can cost more than the search it certifies. Verif shows that this overhead is not inherent to IVF-Flat. Its central design principle is to keep scores at their point of use: ranking claims flow through virtual keys to authenticated inner products, while the resulting masks directly drive routing and output binding. The proof therefore follows the native candidate scan rather than a materialized score or permutation pipeline. Across SIFT and Cohere, 16-bit quantization has negligible impact on exact squared-L_2 rankings. On one-million-vector indexes, CPU proving takes 9.65 and 10.80 seconds, verification takes at most 1.33 seconds, and proofs remain approximately 1 MiB. With a fixed scan budget, Verif further proves 32M-vector SIFT and 8M-vector Cohere indexes in 5.90 and 11.57 seconds. Relative to the optimized general-purpose baseline, it reduces prover time by up to 86.5× and peak memory by up to 99.1%. Together, these results move complete, service-consistent IVF-Flat verification beyond a conceptual guarantee and into the operating range of practical RAG-as-a-Service deployment.

References

[1] Kasra Abbaszadeh, Christodoulos Pappas, Jonathan Katz, and Dimitrios Papadopoulos. 2024. Zero-knowledge proofs of training for deep neural networks. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. 4316–4330.

[2] David Balbás, Dario Fiore, María Isabel González Vasco, Damien Robissout, and Claudio Soriente. 2023. Modular sumcheck proofs with applications to machine learning and image processing. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 1437–1451.

[3] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. 2016. Interactive oracle proofs. In *Theory of Cryptography Conference*. Springer, 31–60.

[4] Matteo Campanelli, Antonio Faonio, Dario Fiore, Tianyu Li, and Helger Lipmaa. 2024. Lookup arguments: improvements, extensions and applications to zero-knowledge decision trees. In *IACR International Conference on Public-Key Cryptography*. Springer, 337–369.

[5] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. 2023. Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 499–530.

[6] Bing-Jyue Chen, Suppakit Waiwitlikhit, Ion Stoica, and Daniel Kang. 2024. Zkml: An optimizing system for ml inference in zero-knowledge proofs. In *Proceedings of the Nineteenth European Conference on Computer Systems*. 560–574.

[7] Chanyeol Choi, Jihoon Kwon, Jaeseon Ha, Hojun Choi, Chaewoon Kim, Yongjae Lee, Jy-yong Sohn, and Alejandro Lopez-Lira. 2025. Finder: Financial dataset for question answering and evaluating retrieval-augmented generation. In *Proceedings of the 6th ACM International Conference on AI in Finance*. 638–646.

[8] Haohua Duan, Zedong Peng, Liyao Xiang, Yuncong Hu, and Bo Li. 2024. A verifiable and privacy-preserving federated learning training framework. *IEEE Transactions on Dependable and Secure Computing* 21, 5 (2024), 5046–5058.

[9] Haohua Duan, Liyao Xiang, Xinbing Wang, Pengzhi Chu, and Chenghu Zhou. 2023. A new zero knowledge argument for general circuits and its application. *IEEE Transactions on Information Forensics and Security* 18 (2023), 3906–3920.

[10] Boyuan Feng, Zheng Wang, Yuke Wang, Shu Yang, and Yufei Ding. 2024. Zeno: A type-based optimization framework for zero knowledge neural network inference. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*. 450–464.

[11] Amos Fiat and Adi Shamir. 1986. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*. Springer, 186–194.

[12] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* (2023).

[13] Sanjam Garg, Aarushi Goel, Somesh Jha, Saeed Mahloujifar, Mohammad Mahmoody, Guru-Vamsi Policharla, and Mingyuan Wang. 2023. Experimenting with zero-knowledge proofs of training. In *Proceedings of the 2023 ACM SIGSAC conference on computer and communications security*. 1880–1894.

[14] Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. 2015. Delegating computation: interactive proofs for muggles. *Journal of the ACM (JACM)* 62, 4 (2015), 1–64.

[15] Ulrich Haböck. 2022. Multivariate lookups based on logarithmic derivatives. Cryptology ePrint Archive, Paper 2022/1530. https://eprint.iacr.org/2022/1530

[16] Meng Hao, Hanxiao Chen, Hongwei Li, Chenkai Weng, Yuan Zhang, Haomiao Yang, and Tianwei Zhang. 2024. Scalable zero-knowledge proofs for non-linear functions in machine learning. In *33rd USENIX Security Symposium (USENIX Security 24)*. 3819–3836.

[17] Xiangyu Hui, Xingliang Yuan, Olga Ohrimenko, and Sid Chi-Kin Chau. 2026. Practical and Verifiable Encrypted Vector Search for Retrieval-Augmented Generation. Cryptology ePrint Archive, Paper 2026/923. https://eprint.iacr.org/2026/923

[18] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *2018 IEEE/CVF conference on computer vision and pattern recognition*. IEEE, 2704–2713.

[19] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 33, 1 (2010), 117–128.

[20] Yanze Jiang, Xinyang Yang, Xuanming Liu, Yanpei Guo, and Jiaheng Zhang. 2026. zkRAG: Efficiently Proving RAG Retrieval in Zero Knowledge. Cryptology ePrint Archive, Paper 2026/709. https://eprint.iacr.org/2026/709

[21] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE transactions on big data* 7, 3 (2019), 535–547.

[22] Andrew Kane and pgvector Contributors. 2021. pgvector: Open-Source Vector Similarity Search for Postgres. https://github.com/pgvector/pgvector. GitHub repository.

[23] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. 2010. Constant-size commitments to polynomials and their applications. In *International conference on the theory and application of cryptology and information security*. Springer, 177–194.

[24] Seunghwa Lee, Hankyung Ko, Jihye Kim, and Hyunok Oh. 2024. vcnn: Verifiable convolutional neural network based on zk-snarks. *IEEE Transactions on Dependable and Secure Computing* 21, 4 (2024), 4254–4270.

[25] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems* 33 (2020), 9459–9474.

[26] Chenqi Lin, Yubo Cui, Zhelei Zhou, Cheng Hong, Yufei Wang, Zhaohui Chen, and Meng Li. 2026. VeriRAG: Efficient Zero-Knowledge Proofs for Verifiable Retrieval-Augmented Generation. Cryptology ePrint Archive, Paper 2026/637. https://eprint.iacr.org/2026/637

[27] Tianyi Liu, Xiang Xie, and Yupeng Zhang. 2021. Zkcnn: Zero knowledge proofs for convolutional neural network predictions and accuracy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2968–2985.

[28] Tao Lu, Yuxun Chen, Zonghui Wang, Xiaohang Wang, Wenzhi Chen, and Jiaheng Zhang. 2025. Batchzk: A fully pipelined gpu-accelerated system for batch generation of zero-knowledge proofs. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*. 100–115.

[29] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. 1992. Algebraic methods for interactive proof systems. *Journal of the ACM (JACM)* 39, 4 (1992), 859–868.

[30] Rohan Mahapatra, Harsha Santhanam, Christopher Priebe, Hanyang Xu, and Hadi Esmaeilzadeh. 2025. In-storage acceleration of retrieval augmented generation as a service. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*. 450–466.

[31] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.

[32] Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. 2013. Signatures of correct computation. In *Theory of Cryptography Conference*. Springer, 222–242.

[33] Shahar Papini and Ulrich Haböck. 2023. Improving logarithmic derivative lookups using GKR. *IACR Cryptol. ePrint Arch.* 2023 (2023), 1284.

[34] Nicholas Pipitone and Ghita Houir Alami. 2024. Legalbench-rag: A benchmark for retrieval-augmented generation in the legal domain. *arXiv preprint arXiv:2408.10343* (2024).

[35] Polygon Zero. 2025. Plonky2. GitHub repository, release v1.1.0. https://github.com/0xPolygonZero/plonky2/releases/tag/v1.1.0 Source code.

[36] Zipeng Qiu, Wenjie Qu, Jiaheng Zhang, and Binhang Yuan. 2026. V3DB: Audit-on-Demand Zero-Knowledge Proofs for Verifiable Vector Search over Committed Snapshots. *arXiv preprint arXiv:2603.03065* (2026).

[37] Wenjie Qu, Yanpei Guo, Yue Ying, and Jiaheng Zhang. 2026. Verfcnn, optimal complexity zksnark for convolutional neural networks. In *2026 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1–18.

[38] Wenjie Qu, Yijun Sun, Xuanming Liu, Tao Lu, Yanpei Guo, Kai Chen, and Jiaheng Zhang. 2025. {zkGPT}: An Efficient Non-interactive Zero-knowledge Proof Framework for {LLM} Inference. In *34th USENIX Security Symposium (USENIX Security 25)*. 2045–2063.

[39] Srinath Setty, Justin Thaler, and Riad Wahby. 2024. Unlocking the lookup singularity with lasso. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 180–209.

[40] Haochen Sun, Tonghe Bai, Jason Li, and Hongyang Zhang. 2024. Zkdl: Efficient zero-knowledge proofs of deep learning training. *IEEE Transactions on Information Forensics and Security* 20 (2024), 914–927.

[41] Haochen Sun, Jason Li, and Hongyang Zhang. 2024. zkllm: Zero knowledge proofs for large language models. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. 4405–4419.

[42] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, et al. 2021. Milvus: A purpose-built vector data management system. In *Proceedings of the 2021 international conference on management of data*. 2614–2627.

[43] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. 2021. Mystique: Efficient conversions for {Zero-Knowledge} proofs with applications to machine learning. In *30th USENIX Security Symposium (USENIX Security 21)*. 501–518.

[44] Jiasi Weng, Jian Weng, Gui Tang, Anjia Yang, Ming Li, and Jia-Nan Liu. 2023. pvcnn: Privacy-preserving and verifiable convolutional neural network testing. *IEEE Transactions on Information Forensics and Security* 18 (2023), 2218–2233.

[45] Guangzhi Xiong, Qiao Jin, Zhiyong Lu, and Aidong Zhang. 2024. Benchmarking retrieval-augmented generation for medicine. In *Findings of the Association for Computational Linguistics: ACL 2024*. 6233–6251.

[46] Jiaheng Zhang, Zhiyong Fang, Yupeng Zhang, and Dawn Song. 2020. Zero knowledge proofs for decision tree predictions and accuracy. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2039–2053.

[47] Yixiao Zheng, Changzheng Wei, Xiaodong Qi, Hanghang Wu, Yuhan Wu, Li Lin, Tianmin Song, Ying Yan, Yanqing Yang, Zhao Zhang, et al. 2026. ZKSL: Verifiable and Efficient Split Federated Learning via Asynchronous Zero-Knowledge Proofs.. In *NDSS*.

A Complete VERIF Protocol

A.1 Statement, Witness, and Leakage

Let pp contain the field and proof-system parameters, the data owner's signature-verification key, the domain-separated functions H and H_{tag}, the canonical encoding, and the public limb tables. The authorized policy Π fixes the IVF-Flat dimensions, the scoring interface $(D_s, \alpha, \psi, \phi)$, quantization and limb parameters, dummy encoding, and admissible retrieval parameters.

We write $\mathsf{WF}_\Pi(\mathsf{Snap}) = 1$ when all centroid and slot arrays have the prescribed dimensions; every authenticated scoring row equals the policy-defined ϕ-representation of its certified vector; validity bits are Boolean; every valid identifier lies in $[0, N_{\mathrm{cap}})$ and is globally unique; invalid slots use the canonical dummy encoding; valid digests correspond to canonical text; and the certified coordinate and intermediate bounds satisfy the field-size condition of Section 3.2.

Throughout Section 6 and Appendix A, the committed-oracle notation $\widetilde{\mu}$, $\widetilde{X}_c$, Com_μ, and $\mathsf{Com}_{X,c}$ refers to the multilinear extensions and commitments of the D_s-dimensional scoring tensors $(\phi(\mu_i))_{i\in[C]}$ and $(\phi(X_c[\ell]))_{\ell\in[L]}$, respectively, rather than to the underlying raw D-dimensional vectors.

Let $S = (S_{\min}, S_{\max})$.

For any snapshot satisfying $\mathsf{WF}_\Pi(\mathsf{Snap}) = 1$, the data owner publishes the signed manifest

$$\mathcal{M} = \big(\mathsf{sid}, \mathsf{ver}, C, L, D, b, S, w, \Pi, \mathsf{Com}_\mu, \mathsf{root}_{\mathsf{Dir}}, \mathsf{Com}_\eta\big)$$

and $\sigma_{\mathcal{M}} \leftarrow \mathsf{Sign}_{\mathsf{sk}_{\mathsf{do}}}(\mathcal{M})$. For every cluster c, the authenticated directory contains

$$\mathsf{Dir}[c] = \big(\mathsf{Com}_{X,c}, \mathsf{Com}_{v,c}, \mathsf{Com}_{u,c}, \mathsf{Com}_{h,c}\big),$$
$$\eta_c = H_{\mathrm{tag}}\big(c \parallel \mathsf{Dir}[c]\big).$$

The manifest authenticates the directory root and the commitment to the tag array $(\eta_c)_{c\in[C]}$.

For a public quantized query q and authorized output size k, the policy determines the probe budget m. The public instance is

$$x = \big(\mathcal{M}, \sigma_{\mathcal{M}}, q, k, m, I_{\mathsf{sel}}, \mathsf{Auth}_{\mathsf{sel}}, R_{\mathsf{out}}\big),$$

where I_{sel} is the increasing list of routed cluster identifiers, $\mathsf{Auth}_{\mathsf{sel}}$ contains their authenticated directory entries, and

$$R_{\mathsf{out}} = \big((u_r, s_r, T_r)\big)_{r\in[k]}$$

is the ordered retrieval result.

The signed manifest is treated as authenticated preprocessing for the online relation. Let $\mathsf{Snap}_{\mathcal{M}}$ denote the fixed well-formed snapshot certified by $\mathcal{M}$. The server stores this snapshot and the opening state for its preprocessing commitments. These preprocessing oracles are fixed parameters of the online relation rather than values that must be extracted from every accepted query proof.

For one query, the prover uses authenticated evaluations of the centroid and tag arrays μ, η and of the selected cluster arrays $\{Y_{I_{\mathsf{sel}}[j]}\}_{j\in[m]}$ for $Y \in \{X, v, u, h\}$. Evaluation binding requires these values to agree with the commitments certified by $\mathcal{M}$; the online extractor need not recover any unqueried cluster. The extractable query-time witness ω_{q} contains the coarse and fine masks, threshold limbs, gap limbs, and the auxiliary witnesses required by Zerocheck, LogUp, and the limb-range argument. Scores, ranking keys, and the concatenated selected subindex remain derived virtual functions rather than witness oracles.

The authenticated online relation $\mathcal{R}_{\mathsf{ret}}(x; \omega_{\mathrm{q}}) = 1$ is parameterized by the preprocessing commitments certified in $\mathcal{M}$. It holds when the manifest signature and policy checks succeed, every disclosed directory entry authenticates against $\mathsf{root}_{\mathsf{Dir}}$, and all preprocessing evaluations used by the protocol open consistently against Com_μ, Com_η, and the commitments in the disclosed directory entries. The query-time witness must additionally satisfy the relations in Sections 5–6 relative to these fixed preprocessing oracles. Equivalently,

$$\big(I_{\mathsf{sel}}, R_{\mathsf{out}}\big) = \mathsf{IVFFlat}_\Pi(\mathsf{Snap}_{\mathcal{M}}, q, k)$$

under the prescribed quantized scoring, validity, and deterministic tie-breaking semantics. Collision resistance of H additionally binds every returned canonical text to its authenticated digest.

The permitted per-query leakage is

$$\mathcal{L}_\Pi = \big(\mathcal{M}, \sigma_{\mathcal{M}}, q, k, m, I_{\mathsf{sel}}, \mathsf{Auth}_{\mathsf{sel}}, R_{\mathsf{out}}\big).$$

Thus, VERIF reveals the coarse access pattern and returned records, but hides all remaining snapshot contents and scores, the fine-selection mask, private thresholds and gaps, and lookup witnesses.

A.2 The End-to-End Protocol

Protocol 1 presents the complete public-coin execution. Zerocheck, Sumcheck, LogUp, and the limb-range argument are invoked through their standard interfaces. All public outputs and challenge-independent query-time oracles are fixed before the corresponding verifier challenges. LogUp inverse oracles are constructed only after their denominator challenges are sampled. Below, $\widehat{\mathsf{Com}}$ denotes the commitment handle produced by the zero-knowledge PCS compiler described in Appendix A.3.

A.3 Zero-Knowledge and PCS Compilation

Protocol 1 describes the logical public-coin PIOP. Directly revealing its private oracle evaluations would disclose information about the snapshot, masks, or selection thresholds. VERIF therefore applies a standard simulation-secure zero-knowledge compiler for Sumcheck-based polynomial protocols before committing to the oracles. We denote this transformation by ZKComp and write $\widehat{\mathsf{Com}}_f$ for the resulting compiled commitment handle of an oracle f.

The compiler uses fresh masking randomness for every query and consistently masks repeated occurrences of the same terminal claim. Its terminal interface preserves the underlying polynomial relations while revealing only simulatable masked evaluations. Private thresholds are represented as constant witness oracles, and the same compiled limbs are used by both their reconstruction relations and limb-range checks. As an implementation detail, for each stage $\rho \in \{\mathrm{c}, \mathrm{f}\}$, the compiler samples a fresh common offset and sets

$$\Delta_\rho \xleftarrow{\$} [0, 2^{128}) \cap \mathbb{Z},$$
$$\kappa_\rho^{\mathrm{zk}}(z) = \kappa_\rho(z) + \Delta_\rho,$$
$$\tau_\rho^{\mathrm{zk}} = \tau_\rho + \Delta_\rho,$$
$$B_\rho^{\mathrm{zk}} = \max\{B_\rho, 128\} + 1.$$

The common shift preserves ordering and cancels from every gap. It therefore changes only the compiled range representation, not

Protocol 1 (VERIF). Let Snap be a certified IVF-Flat snapshot with signed manifest $(\mathcal{M}, \sigma_{\mathcal{M}})$, let q be the public quantized query, let k be authorized by policy Π, and let m be the probe budget deterministically prescribed by Π for (q, k).

Setup. $\mathsf{VERIF.KeyGen}(1^{\lambda_{\mathrm{sec}}})$ runs $\mathsf{pp}_{\mathsf{PCS}} \leftarrow \mathsf{PCS.Setup}(1^{\lambda_{\mathrm{sec}}})$ and $(\mathsf{sk}_{\mathsf{do}}, \mathsf{vk}_{\mathsf{do}}) \leftarrow \mathsf{Sig.KeyGen}(1^{\lambda_{\mathrm{sec}}})$, fixes the domain-separated hashes, canonical encoding, and public limb tables, places these descriptions together with $\mathsf{pp}_{\mathsf{PCS}}$ and $\mathsf{vk}_{\mathsf{do}}$ in pp, and outputs $(\mathsf{pp}, \mathsf{sk}_{\mathsf{do}})$.

Snapshot commitment. $\mathsf{VERIF.Commit}(\mathsf{Snap}, \Pi)$ commits to $\widetilde{\mu}$ and every $\widetilde{Y}_c$ for $Y \in \{X, v, u, h\}$, constructs $\mathsf{Dir}[c]$ and $\eta_c = H_{\mathrm{tag}}(c \parallel \mathsf{Dir}[c])$, authenticates the directory, commits to $\widetilde{\eta}$, and publishes $(\mathcal{M}, \sigma_{\mathcal{M}})$, where $\sigma_{\mathcal{M}} \leftarrow \mathsf{Sig.Sign}_{\mathsf{sk}_{\mathsf{do}}}(\mathcal{M})$. The server receives the snapshot and PCS opening state.

Online protocol. The parties execute $\langle \mathsf{VERIF.Prove}, \mathsf{VERIF.Verify} \rangle(\mathcal{M}, \sigma_{\mathcal{M}}, q, k)$ over the fine domain $\Omega_{\mathrm{f}} = [m] \times [L]$.

1. Statement and witness commitments. The prover computes $(I_{\mathrm{sel}}, R_{\mathrm{out}}) \leftarrow \mathsf{IVFFlat}_{\Pi}(\mathsf{Snap}, q, k)$, forms the logical witnesses $\omega_{\mathrm{c}} = (a_{\mathrm{c}}, \{\tau_{\mathrm{c}}^{(r)}, g_{\mathrm{c}}^{(r)}\}_r)$ and $\omega_{\mathrm{f}} = (a_{\mathrm{f}}, \{\tau_{\mathrm{f}}^{(r)}, g_{\mathrm{f}}^{(r)}\}_r)$, and sends I_{sel}, $\mathsf{Auth}_{\mathrm{sel}}$, R_{out}, $\widehat{\mathsf{Com}}(\omega_{\mathrm{c}})$, and $\widehat{\mathsf{Com}}(\omega_{\mathrm{f}})$. The hatted handles include the compiler masking material specified in Appendix A.3. Scores, keys, and the concatenated selected subindex remain virtual. At this point, the public instance x is fixed as defined in Appendix A.1; every subsequent challenge is bound to x and the complete preceding transcript.

2. Public checks. The verifier checks the signature and every disclosed directory entry. It checks that q and k satisfy all public bounds fixed by Π, recomputes the prescribed m, and requires $1 \le m \le C$, $1 \le k \le mL$, and $0 \le I_{\mathrm{sel}}[0] < \cdots < I_{\mathrm{sel}}[m-1] < C$. It computes $q_s = \psi(q)$ and $\alpha(q)$. Writing $R_{\mathrm{out}} = ((u_r, s_r, T_r))_{r<k}$, it checks $0 \le u_r < N_{\mathrm{cap}}$ and $S_{\min} \le s_r \le S_{\max}$, computes

$$h_r = H(\mathrm{Canon}(T_r)), \qquad \kappa_r = U_{\mathrm{f}}(s_r - S_{\min}) + (U_{\mathrm{f}} - 1 - u_r),$$

and requires $\kappa_0 > \cdots > \kappa_{k-1}$ in ordinary integer arithmetic.

3. Coarse top-m. Zerocheck and Sumcheck prove $a_{\mathrm{c}}(a_{\mathrm{c}} - 1) = 0$, $\sum_{i<C} a_{\mathrm{c}}(i) = m$, and $F_{\mathrm{c}}(b) = 0$ for every $b \in \{0,1\}^{\ell_C}$. The compiler realizes the logical limb-range claims using $\mathrm{Range}_{B_{\mathrm{c}}^{\mathrm{zk}}}(\tau_{\mathrm{c}}^{\mathrm{zk}})$ and $\mathrm{Range}_{B_{\mathrm{c}}^{\mathrm{zk}}}(g_{\mathrm{c}}(i))$ for every $i < C$, as specified in Appendix A.3. After the compiler-only translation is discharged, the logical terminal claim is reduced as

$$\begin{aligned} \widetilde{\kappa}_{\mathrm{c}}(r_{\mathrm{c}}) &\Rightarrow \widetilde{s}_{\mathrm{c}}^{\circ}(r_{\mathrm{c}}) \Rightarrow \widetilde{s}_{\mathrm{c}}(r_{\mathrm{c}}) \\ &\Rightarrow \widetilde{A}_{\mathrm{c}}(r_{\mathrm{c}}, r_D^{\mathrm{c}}). \end{aligned}$$

4. Routing authentication. After I_{sel}, its authenticated tags, and a_{c} are fixed, the verifier samples $(\alpha, \beta) \xleftarrow{\$} \mathbb{F}_p^2$. The prover commits to the routing inverse oracles, and LogUp proves

$$\sum_{j<m} (\alpha - I_{\mathrm{sel}}[j] - \beta\eta_{I_{\mathrm{sel}}[j]})^{-1} = \sum_{c<C} a_{\mathrm{c}}(c)(\alpha - c - \beta\eta_c)^{-1}.$$

5. Fine top-k. Zerocheck and Sumcheck prove $a_{\mathrm{f}}(a_{\mathrm{f}} - 1) = 0$, $a_{\mathrm{f}}(1 - v_{\mathrm{sel}}) = 0$, $\sum_{z\in\Omega_{\mathrm{f}}} a_{\mathrm{f}}(z) = k$, and $F_{\mathrm{f}}(b) = 0$ for every $b \in \{0,1\}^{\ell_m + \ell_L}$. The compiler realizes the logical limb-range claims using $\mathrm{Range}_{B_{\mathrm{f}}^{\mathrm{zk}}}(\tau_{\mathrm{f}}^{\mathrm{zk}})$ and $\mathrm{Range}_{B_{\mathrm{f}}^{\mathrm{zk}}}(g_{\mathrm{f}}(z))$ for every $z \in \Omega_{\mathrm{f}}$, as specified in Appendix A.3. At $r_{\mathrm{f}} = (r_m, r_L)$, after discharging the compiler-only translation, the logical terminal claim is reduced as

$$\begin{aligned} \widetilde{\kappa}_{\mathrm{f}}(r_{\mathrm{f}}) &\Rightarrow \big(\widetilde{s}_{\mathrm{f}}^{\circ}(r_{\mathrm{f}}), \widetilde{u}_{\mathrm{sel}}(r_{\mathrm{f}})\big) \\ &\Rightarrow \big(\widetilde{s}_{\mathrm{f}}(r_{\mathrm{f}}), \widetilde{u}_{\mathrm{sel}}(r_{\mathrm{f}})\big) \\ &\Rightarrow \big(\widetilde{A}_{\mathrm{f}}(r_{\mathrm{f}}, r_D^{\mathrm{f}}), \widetilde{u}_{\mathrm{sel}}(r_{\mathrm{f}})\big). \end{aligned}$$

6. Output binding. After R_{out} and a_{f} are fixed, the verifier samples $(\gamma, \delta) \xleftarrow{\$} \mathbb{F}_p^2$. The prover commits to the output inverse oracles, and LogUp proves

$$\sum_{r<k} (\gamma - \kappa_r - \delta h_r)^{-1} = \sum_{z\in\Omega_{\mathrm{f}}} a_{\mathrm{f}}(z)(\gamma - \kappa_{\mathrm{f}}(z) - \delta h_{\mathrm{sel}}(z))^{-1}.$$

This instance uses the unshifted fine keys. Its virtual key claims are reduced through the logical interface of Step 5, while its digest claims are authenticated against the folded digest commitments. If any routing or output denominator is zero, the current trial ends before the corresponding inverse-oracle commitment and the entire execution restarts with fresh compiler randomness. Such a trial abort occurs with negligible probability and therefore incurs only negligible completeness loss.

7. Folded terminal openings. For every required point r_m and $Y \in \{X, v, u, h\}$, the verifier derives

$$\widehat{\mathsf{Com}}_Y^{\mathrm{sel}}(r_m) = \sum_{j<m} \widetilde{\mathrm{eq}}(r_m, j)\,\widehat{\mathsf{Com}}_{Y, I_{\mathrm{sel}}[j]}.$$

The prover supplies batched PCS openings for all committed preprocessing, query-time, inverse-oracle, and folded terminal claims. No score, ranking-key, or concatenated selected-view oracle is separately committed or opened. The verifier accepts exactly when all public checks, subprotocols, directory authentications, and openings verify.

the unmasked retrieval relation of Section 5. Challenge-dependent routing and output inverse oracles are constructed and committed only after their respective LogUp challenges have been sampled and before any subsequent Zerocheck or Sumcheck challenge. Thus, no challenge-dependent witness can be chosen after the claim that it is used to prove.

VERIF requires a hiding, extractably binding, and additively homomorphic multilinear PCS. Linearity is essential for the selected-view construction: for every required point r_m and $Y \in \{X, v, u, h\}$, the verifier derives

$$\widehat{\mathsf{Com}}_Y^{\mathrm{sel}}(r_m) = \sum_{j\in[m]} \widetilde{\mathrm{eq}}(r_m, j)\,\widehat{\mathsf{Com}}_{Y, I_{\mathrm{sel}}[j]}.$$

Linearity of ZKComp makes this a valid compiled commitment to the correspondingly masked selected-view claim. The prover combines the associated opening states using the same public coefficients. Consequently, VERIF introduces neither a commitment to a concatenated selected subindex nor m independent openings for each terminal claim. Once all terminal claims are fixed, compatible claims sharing a domain and evaluation point are combined with a fresh random linear combination and opened as one PCS batch.

Our current prototype instantiates this interface with PST13 [32], a pairing-based multilinear PCS with a structured setup. PST13 provides constant-size commitments, linear prover work, and logarithmic-size evaluation proofs and verifier work. Its additive homomorphism is well matched to VERIF's selected-view construction, allowing the verifier to fold the m selected per-cluster commitments using $O(m)$ group operations. These properties align with VERIF's linear-prover design while keeping commitment folding and terminal openings compact.

Finally, Fiat–Shamir derives every verifier challenge from the public instance and the complete preceding transcript:

$$c_i = H_{\mathsf{FS}}\big(\mathsf{dst}_i \parallel x \parallel \mathsf{tr}_{i-1} \parallel M_i\big),$$

where distinct phase labels dst_i separate coarse selection, routing, fine selection, output binding, and PCS batching. The message order is exactly that of Protocol 1; a restarted execution uses fresh compiler randomness and therefore a fresh transcript; no accepting proof contains a zero denominator. The resulting proof consists of the compiled oracle commitments, component-protocol messages, masked terminal claims, and batched PCS openings. Under the security of ZKComp and the hiding and extractable-binding properties of the PCS, this compilation preserves the PIOP's knowledge soundness and zero knowledge. Appendix B gives the resulting security argument.

B Security of VERIF

Let $\Pi_{\mathsf{V}} = (P, V)$ denote the public-coin VERIF PIOP and let $\lambda = \lambda_{\mathsf{sec}}$. We write $\mathsf{Prep}_{\mathcal{M}}$ for the preprocessing oracles fixed by the certified manifest $\mathcal{M}$.

The relation $\mathcal{R}_{\mathsf{IVF}}$ denotes the online relation defined in Sections 4–6, parameterized by $\mathsf{Prep}_{\mathcal{M}}$. Its extractable witness is the query-time witness ω_{q}; the preprocessing oracles themselves are fixed relation parameters. After PCS compilation, signature verification, and directory authentication, this relation becomes $\mathcal{R}_{\mathsf{ret}}$ of Appendix A.1.

Security definitions. Completeness requires that, for every valid online instance and witness $(x, \omega_{\mathsf{q}}) \in \mathcal{R}_{\mathsf{IVF}}$,

$$\Pr\big[\langle P(\omega_{\mathsf{q}}), V\rangle(x) = 1\big] \geq 1 - \mathrm{negl}(\lambda).$$

Knowledge soundness requires that, for every polynomial-time prover P^*, there exists an expected polynomial-time extractor Ext^{P^*} such that

$$\Pr\Big[V^{P^*}(x) = 1 \ \wedge\ \mathcal{R}_{\mathsf{IVF}}(x; \omega_{\mathsf{q}}^*) = 0\Big] \leq \epsilon_{\mathsf{KS}},$$

$$\omega_{\mathsf{q}}^* \leftarrow \mathsf{Ext}^{P^*}(x).$$

The probability is over the verifier and extractor randomness, and ϵ_{KS} is negligible.

Zero knowledge relative to $\mathcal{L}_\Pi$ requires that, for every polynomial-time verifier V^*, there exists a polynomial-time simulator Sim satisfying

$$\mathsf{View}_{V^*}\big[\langle P(\omega_{\mathsf{q}}), V^*\rangle(x)\big] \approx_c \mathsf{Sim}\big(\mathsf{pp}, \mathcal{L}_\Pi(\mathsf{Snap}_{\mathcal{M}}; q, k)\big).$$

For multiple queries against one snapshot, the definition uses the joint sequence of their permitted leakage.

Proof of Theorem 1. *Completeness.* The honest prover runs the prescribed quantized IVF-Flat algorithm and sets a_{c} and a_{f} to its coarse and fine selection masks. It chooses

$$\tau_{\mathsf{c}} = \min_{a_{\mathsf{c}}(i)=1} \kappa_{\mathsf{c}}(i), \qquad \tau_{\mathsf{f}} = \min_{a_{\mathsf{f}}(z)=1} \kappa_{\mathsf{f}}(z),$$

and constructs the gaps and their canonical limb decompositions according to Section 5. By construction, all Booleanity, cardinality, validity, threshold–gap, and range relations hold.

The exact-score relations hold by the definitions of the coarse and fine inner products. The routing list is the increasing support of a_{c}, so the two routing multisets coincide. Likewise, the ordered output records are exactly those selected by a_{f}, so the output multisets coincide. Consequently, every polynomial identity, limb-range lookup, and LogUp equality submitted by the honest prover is valid. After retrying the negligible set of LogUp challenges producing a zero denominator, component completeness gives

$$\Pr[\langle P(\omega_q), V\rangle(x) = 1] \geq 1 - \varepsilon_{\mathrm{comp}} = 1 - \mathrm{negl}(\lambda).$$

Knowledge soundness. All challenge-independent masks, threshold limbs, and gap limbs are fixed before their corresponding challenges. Knowledge soundness of Zerocheck and Sumcheck extracts oracles satisfying Booleanity, cardinality, validity, and the two gap identities. Limb-range extraction binds the same limbs to their claimed canonical integer ranges. The field-size condition then prevents a false field relation from being interpreted as a wrapped integer relation.

Every extracted key claim is connected to authenticated data through

$$\begin{aligned}\widetilde{\kappa}_{\mathsf{c}} &\Rightarrow \widetilde{s}_{\mathsf{c}}^{\circ} \Rightarrow \widetilde{s}_{\mathsf{c}} \\ &\Rightarrow \widetilde{A}_{\mathsf{c}},\end{aligned}$$

$$\begin{aligned}\widetilde{\kappa}_{\mathsf{f}} &\Rightarrow \big(\widetilde{s}_{\mathsf{f}}^{\circ}, \widetilde{u}_{\mathsf{sel}}\big) \\ &\Rightarrow \big(\widetilde{s}_{\mathsf{f}}, \widetilde{u}_{\mathsf{sel}}\big) \\ &\Rightarrow \big(\widetilde{A}_{\mathsf{f}}, \widetilde{u}_{\mathsf{sel}}\big).\end{aligned}$$

Thus, except with the score-reduction soundness error, the extracted scores are the prescribed exact inner products.

Let the following events denote the consequences of the four protocol phases:

$$\begin{aligned}\mathcal{E}_{\mathsf{c}} &: a_{\mathsf{c}} = \mathsf{Top}_m(\kappa_{\mathsf{c}}), \\ \mathcal{E}_{\mathsf{r}} &: I_{\mathsf{sel}} = \mathsf{Inc}(\mathrm{supp}(a_{\mathsf{c}})), \\ \mathcal{E}_{\mathsf{f}} &: a_{\mathsf{f}} = \mathsf{Top}_k(\kappa_{\mathsf{f}}; v_{\mathsf{sel}}), \\ \mathcal{E}_{\mathsf{o}} &: R_{\mathsf{out}} = \mathsf{Desc}\big(\mathsf{Record}(\mathrm{supp}(a_{\mathsf{f}}))\big).\end{aligned}$$

Here $\mathcal{E}_{\mathsf{r}}$ also binds every selected view to the corresponding authenticated cluster, and $\mathcal{E}_{\mathsf{o}}$ binds scores, identifiers, and digests to the selected slots. The extracted threshold–gap relations, routing LogUp soundness, and output LogUp soundness imply

$$\mathcal{E}_{\mathsf{c}} \wedge \mathcal{E}_{\mathsf{r}} \wedge \mathcal{E}_{\mathsf{f}} \wedge \mathcal{E}_{\mathsf{o}} \implies \mathcal{R}_{\mathsf{IVF}}(x; \omega^*) = 1.$$

If n_{sc} is the public number of Zerocheck and Sumcheck invocations, a union bound gives

$$\begin{aligned}\epsilon_{\mathsf{PIOP}} \leq\ & n_{\mathsf{sc}}\epsilon_{\mathsf{sc}} + 2\epsilon_{\mathsf{range}} + \epsilon_{\mathsf{LU}}^{\mathsf{route}} \\ & + \epsilon_{\mathsf{LU}}^{\mathsf{out}} + \epsilon_{\mathsf{tuple}}.\end{aligned}$$

The final term covers random compression of the routing and output tuples. Therefore, except with negligible probability, the extractor obtains ω_{q}^* containing masks, thresholds, limbs, and auxiliary query-time oracles that satisfy $\mathcal{R}_{\mathsf{IVF}}$ relative to $\mathsf{Prep}_{\mathcal{M}}$. The preprocessing oracle model already fixes the centroid, tag, and per-cluster arrays, so knowledge soundness does not require extracting unqueried cluster contents from a single online proof.

Zero knowledge. The leakage $\mathcal{L}_\Pi$ already contains the manifest, query, routing list, disclosed directory entries, and returned records. In particular, it determines the support of a_{c}. It contains neither the remaining snapshot contents nor the fine mask, thresholds, gaps, lookup witnesses, or private terminal evaluations.

Let $\mathcal{H}_0$ be the real transcript. Define successive hybrids by replacing the coarse-selection, routing, fine-selection, and output-binding transcripts with their component simulators:

$$\mathcal{H}_0 \approx_c \mathcal{H}_1 \approx_c \mathcal{H}_2 \approx_c \mathcal{H}_3 \approx_c \mathcal{H}_4.$$

In every hybrid, ZKComp replaces private oracle evaluations with consistent masked handles. Routing and output inverse-oracle transcripts are simulated only after their public LogUp challenges, matching the real message order. Linearity of the compiler similarly simulates selected-view claims without exposing their constituent oracle evaluations.

The final hybrid depends only on $\mathcal{L}_\Pi$ and equals the output distribution of Sim. By sequential composition,

$$\Delta_c(\mathcal{H}_0, \mathcal{H}_4) \le \epsilon_{\mathsf{ZKComp}} + \sum_{\rho \in \{\mathsf{c},\mathsf{r},\mathsf{f},\mathsf{o}\}} \epsilon_\rho^{\mathsf{zk}} = \mathrm{negl}(\lambda).$$

Fresh compiler randomness for every query gives the same conclusion for the multi-query leakage definition. □

Proof of Corollary 2. Consider an accepting compiled proof. Signature unforgeability binds its manifest to a data-owner-authorized snapshot, while directory binding ensures that every disclosed entry belongs to that manifest. Collision resistance binds directory tags to their entries and returned canonical texts to their authenticated digests.

PCS extraction recovers the query-time polynomials associated with prover-generated commitments. For the fixed preprocessing commitments, evaluation binding ensures that every accepted centroid, tag, and selected-cluster terminal value agrees with its certified oracle. Additive binding further ensures that each folded commitment evaluates to the prescribed public linear combination of the selected per-cluster oracles. Unqueried cluster polynomials remain part of the fixed authenticated preprocessing and need not be extracted from one online proof.

By the assumed multi-round Fiat–Shamir knowledge-soundness theorem, a prover producing an accepting non-interactive proof induces an expected polynomial-time extractor for an accepting execution of the underlying public-coin protocol, except with reduction error ϵ_{FS}. Protocol 1 fixes every challenge-independent oracle before its corresponding challenge, and commits each challenge-dependent LogUp inverse oracle before any subsequent challenge, as required by the compiled message order.

Theorem 1 then extracts a query-time witness ω_{q}^* satisfying $\mathcal{R}_{\mathsf{IVF}}$ relative to $\mathsf{Prep}_{\mathcal{M}}$. Together with the signature, directory, and PCS checks, this establishes

$$\mathcal{R}_{\mathsf{ret}}(x; \omega_{\mathsf{q}}^*) = 1.$$

For zero knowledge, the simulator from Theorem 1 is composed with the masking interface of ZKComp and the hiding PCS. The simulator generates the compiled commitments, masked terminal claims, and corresponding opening transcripts, and programs the random oracle consistently with the simulated public-coin execution. Domain-separated phase labels preserve the message order of Protocol 1, and a restarted execution uses fresh compiler randomness and a fresh transcript. Consequently,

$$\mathsf{View}_{V^*}^{\mathsf{NIZK}}(x) \approx_c \mathsf{Sim}_{\mathsf{PCS},\mathsf{FS}}\big(\mathsf{pp}, \mathcal{L}_\Pi\big).$$

Writing ϵ_{sig}, ϵ_{auth}, ϵ_{hash}, ϵ_{PCS}, and ϵ_{FS} for the corresponding computational and reduction errors, the total soundness error is bounded by

$$\epsilon_{\mathsf{total}} \le \epsilon_{\mathsf{PIOP}} + \epsilon_{\mathsf{sig}} + \epsilon_{\mathsf{auth}} + \epsilon_{\mathsf{hash}} + \epsilon_{\mathsf{PCS}} + \epsilon_{\mathsf{FS}} = \mathrm{negl}(\lambda).$$

This proves the claimed non-interactive zero-knowledge argument of knowledge. □