V3DB: Audit-on-Demand Zero-Knowledge Proofs for Verifiable Vector Search over Committed Snapshots

Zipeng Qiu*
Hong Kong University of Science and Technology
Hong Kong, China
zqiuao@connect.ust.hk

Wenjie Qu*
National University of Singapore
Singapore, Singapore
wenjiequ@u.nus.edu

Jiaheng Zhang
National University of Singapore
Singapore, Singapore
jhzhang@nus.edu.sg

Binhang Yuan†
Hong Kong University of Science and Technology
Hong Kong, China
biyuan@ust.hk

ABSTRACT

Dense retrieval services underpin semantic search, recommendation, and retrieval-augmented generation, yet clients typically see only a top-k list with no auditable execution evidence. We present V3DB, a verifiable, versioned vector-search service that checks on demand whether an untrusted provider's approximate nearest-neighbor (ANN) result was produced by executing published IVF-PQ semantics on a committed snapshot. V3DB commits to each corpus snapshot and standardizes IVF-PQ into a fixed-shape, five-step query semantics. Given a public commitment and query embedding, the service returns top-k payloads and, when challenged, produces a succinct zero-knowledge proof that the output follows these semantics on the committed snapshot, without revealing corpus embeddings or private index contents to the verifier during audit. To make proving practical, V3DB avoids costly in-circuit sorting and random access with multiset equality/inclusion checks plus lightweight boundary conditions. Our Plonky2 prototype proves up to 22× faster and uses up to 40% less peak memory than the circuit-only baseline, with millisecond verification.

PVLDB Reference Format:
Zipeng Qiu, Wenjie Qu, Jiaheng Zhang, and Binhang Yuan. V3DB: Audit-on-Demand Zero-Knowledge Proofs for Verifiable Vector Search over Committed Snapshots. PVLDB, 19(10): 2604 - 2616, 2026.
doi:10.14778/3828612.3828618

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Relaxed-System-Lab/V3DB.

1 INTRODUCTION

Dense retrieval, i.e., similarity search over embedding vectors, has become a standard primitive for various applications such as semantic search, recommendation, and retrieval-augmented generation (RAG) [27, 38, 44, 45]. This shift has driven the widespread adoption of vector-search services for approximate nearest-neighbour (ANN) queries over large embedding corpora [23, 25, 34]. In many real-world deployments, the retrieval pipeline is executed by a remote service provider: a vector-search service maintains the embedding corpus and ANN index and exposes a retrieval API, while clients observe only the returned top-k list. This separation creates a fundamental accountability gap: a retrieval API typically returns only the final ranked list, with little or no evidence of how the search was executed. When a client disputes a result, the service has no efficient way to demonstrate that it executes the mutually agreed-upon ANN procedure—using the promised parameters on the claimed snapshot of its corpus and index. In particular, the service could stealthily reduce the search budget, skip expensive steps, replay a stale snapshot, or introduce subtle ranking biases under operational or business pressure, and such deviations are difficult to detect from the top-k output alone. This raises the essential question: *when challenged, can a retrieval service provide a short, efficiently checkable certificate that its returned top-k list is exactly the output of a publicly specified ANN query semantics on a specific committed corpus snapshot, without requiring the client to download the corpus or rerun search from scratch?*

The ability to audit disputed retrieval results is important in high-value content settings such as patents, finance, and legal search, where retrieval outputs can directly influence key decisions. Commercial offerings increasingly incorporate semantic search over proprietary corpora (e.g., AI-powered patent semantic search and AI patent search in patent analytics platforms [8, 39]) and expose content-retrieval workflows via APIs [26, 28]. Moreover, a provider may have incentives to manipulate results in ways that are difficult to diagnose externally—for example, suppressing unfavourable items about a sponsor, boosting favourable coverage, or answering from an older snapshot that omits recent updates. At the same time, the underlying corpora, embeddings, and index structures are often proprietary, licensed, or sensitive (e.g., paywalled archives, internal repositories used for RAG, user preference profiles, or biometric templates). Releasing the corpus or index to "prove innocence" is therefore unacceptable. A practical audit mechanism must be compact and cheap to check, enabling *audit-on-demand* dispute resolution by clients and third-party auditors, while disclosing only what the application intends to reveal.

Achieving private, auditable retrieval at a realistic scale is challenging because of the integrity and privacy requirements. There are two broad approaches. One can run retrieval inside trusted execution environments (TEEs) and use remote attestation to provide integrity guarantees about a measured implementation, which can reduce disclosure but shifts trust to hardware vendors, attestation infrastructure, and side-channel assumptions; it also requires

*Equal contribution.
†Corresponding author.

Proceedings of the VLDB Endowment, Vol. 19, No. 10 ISSN 2150-8097.
doi:10.14778/3828612.3828618

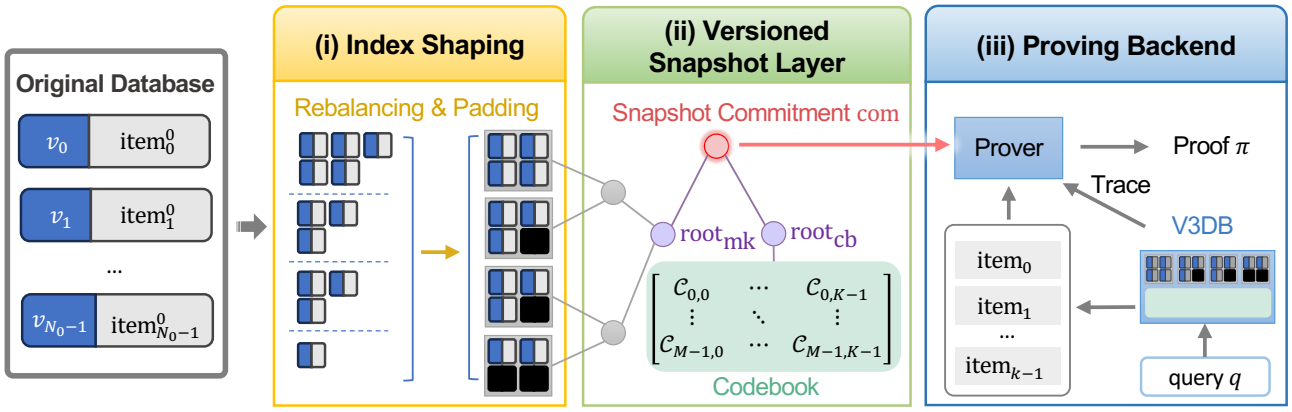

Figure 1: Workflow of V3DB. (i) *Index shaping*: from the original database $\{(v_t, \text{item}_t^0)\}_{t=0}^{N_0-1}$, build a fixed-shape IVF-PQ snapshot via rebalancing and padding. (ii) *Versioned snapshot layer*: publish a snapshot identifier $\text{com} = (\text{root}_{\text{mk}}, \text{root}_{\text{cb}})$, where root_{mk} is a Merkle root committing to the fixed-shape IVF layout and root_{cb} is a hash digest of the PQ codebooks. (iii) *Proving backend*: on query embedding q, return top-k payloads $(\text{item}_0, \ldots, \text{item}_{k-1})$ and, upon challenge, a succinct ZK proof π that the list equals the output of the fixed-shape IVF-PQ semantics on the snapshot committed by com, while hiding snapshot contents and the trace.

always-on protected execution, regardless of whether a query is ever audited later [10, 11, 41]. Alternatively, one can use cryptographic proofs that are generated on demand and can be checked by any party without trusting the operator. We focus on the latter, and in particular on succinct zero-knowledge (ZK) proofs to keep the corpus and index private during verification [4, 5, 21, 37]. At a high level, ZK proofs establish that there exists a private witness—including intermediate values of the retrieval execution—that satisfies a fixed set of arithmetic constraints; prover cost is therefore driven by the circuit size.

ZK proofs impose two practical constraints that shape verifiable retrieval. First, generating a ZK proof is typically substantially more expensive than plaintext execution and scales poorly with the number of constraints, so the proved computation must have a tightly controlled complexity budget. Second, the proved computation must be expressed as a fixed arithmetic circuit: data-dependent control flow (branching) and irregular memory access must be compiled into static relations, often making such steps expensive. Consequently, certifying an *exact* full scan over all N D-dimensional vectors would require encoding $\Theta(ND)$ distance computations and would lead to an impractically large circuit.

This motivates us to focus on an efficient ANN pipeline that avoids full scans and admits a standardized, fixed-shape query procedure. We therefore target inverted-file indices using product quantization (IVF-PQ) [23], whose query execution proceeds in a small sequence of budgeted steps. However, even IVF-PQ contains primitives that are expensive in ZK circuits: probe selection and final top-k extraction are comparison-heavy, candidate scoring relies on lookup tables indexed by PQ codes, and binding accessed snapshot values to a specific version requires authentication for soundness (e.g., Merkle paths inside the proof) [35].

To the best of our knowledge, existing verifiable ANN query schemes in outsourced databases primarily rely on authenticated indices and verification objects (VOs). Such approaches can certify soundness and completeness but typically reveal access-dependent metadata, which is undesirable in many practical settings for large-scale dense retrieval over sensitive corpora [42].

We address this integrity–privacy gap by instantiating retrieval evidence with succinct ZK proofs and present V3DB, a *verifiable, versioned vector database* for dense retrieval at scale. In V3DB, we standardize IVF-PQ into a fixed-shape, five-step query semantics that is amenable to proof generation. Given a public snapshot commitment com and a query embedding, the service returns a top-k list of application-defined payloads and, upon challenge from the client, produces a succinct ZK proof certifying that the list is exactly the output of executing this semantics on the snapshot identified by com, while hiding the underlying corpus embeddings and private index contents from the verifier.

Figure 1 sketches the end-to-end workflow of V3DB: the operator shapes a raw vector database into a fixed-shape IVF-PQ snapshot, commits to each snapshot version, and supports audit-on-demand ZK proofs for disputed query results. We implement both a circuit-only baseline and our optimized multiset-based instantiation in Plonky2 and evaluate their end-to-end costs in detail; the multiset-based design reduces proving time by up to ~ 22× and peak memory by up to ~ 40% while keeping verification in milliseconds.

In summary, we make the following contributions:

- **System and IVF-PQ instantiation.** We present V3DB for verifiable dense retrieval over committed snapshots. We instantiate it for IVF-PQ by standardizing a fixed-shape, five-step query semantics and designing a ZK-friendly index-shaping pipeline that enables fixed-shape execution at scale.
- **Efficient proving backend.** Instead of an impractical circuit-only baseline that directly encodes sorting/selection and random access in-circuit, we avoid these bottlenecks via multiset equality/inclusion checks plus lightweight boundary conditions, and analyze the resulting gate complexity.
- **Implementation and evaluation.** We implement both a baseline and an optimized design, and show through end-to-end

evaluation that the optimized design makes audit-on-demand proof generation practical under realistic budgets.

2 PRELIMINARIES

2.1 ZK Proofs

Zero-knowledge succinct non-interactive arguments of knowledge (ZK-SNARKs) let a prover convince a verifier that a publicly stated computation is correct, without revealing private witness values beyond what is implied by the public inputs/outputs [5, 19, 21, 37]. In V3DB, the public interface consists of a query embedding q, a snapshot version commitment com, and a returned top-k payload list that the verifier checks against the claimed computation.

Circuits as polynomial constraints. In circuit-based SNARKs, the proved computation is expressed as a fixed constraint system: a set of low-degree algebraic relations over a field $\mathbb{F}$. The prover supplies witness values for intermediate wires, and the verifier checks a short proof that these constraints are satisfiable with the claimed public inputs alone. As a minimal example, suppose a circuit computes two candidate values $x_0, x_1 \in \mathbb{F}$ (e.g., from two alternative branches) and uses a private bit $b \in \{0, 1\}$ to select the public output y accordingly. One can enforce: $b(b-1) = 0,\ y = (1-b)x_0 + bx_1$. This encodes conditional behavior without runtime branching.

Fixed-shape execution. A circuit has a predetermined set of constraints and fixed loop bounds across all instances, so data-dependent control flow and irregular memory addressing must be compiled into static relations (e.g., algebraic selection and fixed-size scans). In the example above, if x_0 and x_1 come from two expensive branches, a plaintext program evaluates only one branch, but a circuit must constrain *both* branches and then select using b. This motivates our fixed-shape IVF-PQ semantics and our focus on avoiding costly in-circuit sorting/selection and random access.

Fixed-point representation and range bounds. Circuit constraints are defined over a finite field $\mathbb{F}$, so we represent real-valued embeddings and centroids using fixed-point integers embedded in $\mathbb{F}$ under a public scaling and bit-width chosen per deployment. Our proofs, therefore, attest to correctness with respect to this encoded representation in the circuit. Accordingly, the audited service in V3DB targets at the published fixed-shape fixed-point semantics, where the proof certifies conformance to this audited service, rather than to exact floating-point ANN. To make comparisons and distance computations well defined, we range-bound all relevant witness values so that intermediate results do not wrap around modulo $\mathbb{F}$. Our experiments instantiate this encoding in our prototype and quantify its impact on retrieval utility in practice.

Gate counts and proving time (binning effect). We track circuit size by the number of gate rows G in a PLONK-like system [18]. Modern provers use FFT-based polynomial operations and pad to a power-of-two evaluation domain [3], so we define $G_B := 2^{\lceil \log_2 G \rceil}$ and approximate proving time as $\Theta(G_B \log_2 G_B)$. As a result, reducing G can yield outsized speedups when it pushes the circuit into a smaller G_B bin. For the example constraints above, treating each add/sub/mul as $\Theta(1)$ rows gives a crude estimate $G \approx 6$ (1 mul + 1 sub for the booleanity constraint and 2 mul + 1 sub + 1 add for the selection constraint), hence $G_B = 8$.

2.2 IVF-PQ

IVF-PQ is a widely used ANN index that avoids a full scan by probing a small number of inverted lists and using compact PQ codes for fast approximate *scoring* [23, 25].

Coarse partition (IVF). IVF first trains a set of coarse centroids (e.g., via k-means [2, 32]) and assigns each database vector to its nearest centroid, forming one inverted list per centroid. At query time, instead of scanning the database, the search probes only several the closest inverted lists, yielding a controlled scan budget.

Product quantization (PQ) and table-based scoring. To reduce storage and speed up per-candidate scoring, IVF-PQ applies product quantization to database residuals (database vector minus centroid). Each database vector is stored as a code: the residual is split into blocks, each block is quantized by a small codebook, and the resulting code indices are stored. At query time, the engine builds asymmetric distance computation (ADC) lookup tables from the query residual for each probed list and uses the PQ code indices to fetch and sum table entries, producing approximate distances.

From plain IVF-PQ to verifiable semantics. Together, the IVF and PQ components make IVF-PQ a natural target for audit-on-demand verification: query processing decomposes into a small sequence of steps with explicit budgets (probing and scanning), enabling a well-scoped correctness statement. However, directly proving the plaintext pipeline in a circuit is inefficient. First, inverted lists have variable lengths, inducing data-dependent loop bounds and memory accesses, while circuits require fixed-shape execution. Second, probe selection and final top-k extraction rely on ordering, and PQ scoring uses data-dependent table lookups; both translate to comparison-heavy selection and random access in circuits. To obtain a circuit-friendly statement, V3DB standardizes IVF-PQ into a fixed-shape semantics by enforcing a per-list capacity bound via rebalancing and padding each list to a uniform number of slots with a validity flag. We detail index shaping and formalize the five-step semantics in Section 5.

Notation and terminology. For the remainder of the paper, we use the following notation. Let n_{list} denote the number of inverted lists (coarse centroids) and write the centroid table as $\boldsymbol{\mu} := (\mu_i)_{i=0}^{n_{\text{list}}-1}$, where μ_i is the centroid associated with list i. An *IVF layout* consists of this centroid table together with the corresponding inverted lists. For PQ, we use M sub-quantizers, each with a codebook of size K; we write the *PQ codebooks* as $\{C_{m,k}\}$, where $m \in \{0, \ldots, M-1\}$ and $k \in \{0, \ldots, K-1\}$. Intuitively, each $C_{m,k}$ is a learned codeword used for table-based scoring. In V3DB, we work with a *fixed-shape IVF layout* obtained by rebalancing and padding each list to exactly n slots (a public per-list capacity bound) with an explicit validity flag, and we refer to the fixed-shape IVF layout together with the PQ codebooks as a *snapshot*.

3 RELATED WORK

ANN indices and index shaping. Vector similarity search is widely deployed, and many ANN indices have been studied at scale [25]. We focus on IVF-PQ [23] because its query pipeline admits a standardized, budgeted procedure that can be captured as a fixed-shape semantics. In contrast, graph-based indices such as HNSW [34] rely on adaptive traversals with data-dependent branching and irregular memory access, making ZK authentication

Table 1: Comparison of query-integrity approaches closest to our setting and their main limitations.

Approach	Proof / trust	Key limitation
VO-based ANN	Object-tied authenticated evidence	Leaks access-dependent metadata; verifier cost grows with evidence size.
ZK relational queries	Succinct public proofs for relational operators	Does not directly target ANN-specific ordering and lookup bottlenecks.
TEE retrieval	Enclave attestation and hardware trust	Depends on hardware trust and attestation assumptions.
V3DB	Succinct ZK proof on a committed snapshot	Hides private index state and avoids access-dependent evidence.

substantially more costly. Our fixed-shape layout is also related to capacity-constrained (balanced) clustering [1, 6, 13, 24, 30, 31, 33]; we only require a hard per-list bound that enables circuit compatibility. Unlike prior work that optimizes clustering objectives under such constraints, we use lightweight post-processing to enforce the bound while preserving an IVF-style reference semantics.

Verifiable query processing and retrieval integrity. Prior work on query integrity includes ZK systems for relational query processing (e.g., ZkSQL [29] and PoneglyphDB [22]), authenticated-index and verification-object (VO) methods for outsourced kNN/ANN queries [42], and TEE-based private retrieval [14, 15]. These efforts optimize verification artifacts and trust models. VO-based ANN methods provide query-integrity evidence, but the evidence is object-tied: evidence tied to accessed candidates can reveal which candidate nodes or records were touched during search. ZK systems for relational queries provide succinct public proofs for expressive relational operators, but do not target ANN-specific ordering and lookup bottlenecks over high-dimensional vectors. TEE-based retrieval provides attested execution, but its guarantee depends on hardware trust and attestation assumptions. V3DB targets this gap by standardizing fixed-shape IVF-PQ semantics and providing succinct public auditing on a committed snapshot while hiding private index state. Table 1 summarizes the main distinctions.

4 SYSTEM OVERVIEW

4.1 Workflow and Problem Statement

This subsection introduces V3DB in four parts. The first three are the workflow components depicted in Figure 1: index shaping, the versioned snapshot commitment layer, and the proving backend. We then specify the public interface and the proved statement that the verifier checks. Throughout, we fix a public IVF-PQ configuration, which determines both the plaintext workflow and the circuit shape; Table 2 summarizes the notation used throughout the paper.

Index shaping (offline). Given a raw dataset $\{(v_t, \mathrm{item}_t^0)\}_{t=0}^{N_0-1}$ of N_0 embedding vectors v_t with application-defined payloads item_t^0 and a public configuration (Table 2), the operator builds a fixed-shape IVF-PQ snapshot by applying capacity-constrained rebalancing so that each list has at most n valid records and then padding to exactly n slots with a validity flag per slot.

Versioned snapshot layer (offline). The operator commits to each snapshot version by publishing a public version identifier com before serving queries. We detail the commitment construction and snapshot-binding constraints in Section 5. When the database is updated, the operator repeats index shaping and publishes a new commitment, yielding an epoch-style version history.

Proving backend (online/audit). Given a query embedding q and a claimed snapshot identifier com, the service returns a top-k payload list $(\mathrm{item}_0, \ldots, \mathrm{item}_{k-1})$. Upon challenge, it produces a succinct proof π certifying that there exists a private snapshot consistent with com such that the returned list is exactly the output of our standardized five-step, fixed-shape IVF-PQ semantics. The ZK property hides the snapshot contents and the execution trace from the verifier during verification.

Public interface and proved statement. Given (com, q), the service returns $(\mathrm{item}_0, \ldots, \mathrm{item}_{k-1})$ to the verifier. Upon challenge, it produces a proof π such that

$$\mathsf{Verify}\big(\mathsf{com}, q, (\mathrm{item}_0, \ldots, \mathrm{item}_{k-1}), \pi\big) \in \{1, \mathtt{0}\}.$$

The verifier accepts only when the returned list exactly matches the output of the standardized semantics on the snapshot version committed by com. The only disclosed information is what the application reveals through (com, q) and the returned list.

4.2 Threat Model

We focus on integrity of outsourced dense retrieval. We consider a malicious retrieval operator who controls the corpus, index, and execution and may deviate arbitrarily from the publicly specified workflow while still returning a syntactically valid top-k list. We target integrity violations including (i) *workflow deviation* (e.g., downgrading parameters, skipping steps, cherry-picking candidates, or biasing ranking) and (ii) *version equivocation* (e.g., answering from a stale, mixed, or otherwise different snapshot while claiming a different committed version). The soundness of proof statement rules out *any* deviation that would make the returned list differ from the standardized semantics on a snapshot consistent with the claimed commitment com.

Guarantee boundary. Under standard assumptions for SNARK soundness and hash-based commitment binding, an accepting proof for public inputs $(\mathsf{com}, q, (\mathrm{item}_0, \ldots, \mathrm{item}_{k-1}))$ guarantees that the returned list is the exact output of the published five-step fixed-shape IVF-PQ semantics on a snapshot consistent with com, while keeping the snapshot contents and execution trace hidden beyond the public inputs and outputs. This guarantee is intentionally snapshot-relative. If an operator publishes a low-quality, stale, or otherwise manipulated snapshot and then answers consistently with that published commitment, the proof itself does not reject it. Determining whether the commitment is authorized or not is

Table 2: Key notation for our fixed-shape IVF-PQ configuration, used throughout the paper.

Symbol	Meaning
N_0	number of vectors in the dataset (before shaping/padding)
D	embedding vector dimension
n_{list}	number of inverted lists (coarse centroids)
n_{probe}	number of lists probed per query
n	per-list capacity bound (each list padded to exactly n slots)
M	number of PQ sub-quantizers
K	codebook size per sub-quantizer
k	top-k payload list size
t_{cmp}	comparison bit-length parameter in circuits

therefore handled by the application layer, e.g., through signed snapshot manifests from the data owner or a trusted offline index builder, optionally backed by a transparency log.

Scope. We treat q as a public embedding and focus on proving the retrieval stage. Proving the embedding computation (e.g., the neural inference that produces q) is orthogonal and has been studied by recent ZKML systems [7, 16]. Note we do not aim to provide access-pattern privacy or prevent denial-of-service; rather, we make incorrect execution detectable whenever a proof is produced.

4.3 Proving Cost Overview

In V3DB, we focus on minimizing the gate count needed to express our standardized semantics. A direct circuit-only encoding is expensive because IVF-PQ relies heavily on (i) ordering operations for probe selection and final top-k extraction and (ii) data-dependent table lookups for PQ-based scoring. The circuit-only baseline encodes these primitives directly in-circuit, while our multiset-based design shifts them off-circuit and proves lightweight consistency relations (multiset equality/inclusion plus boundary checks).

Budget abstractions. To discuss configuration trade-offs, we use two standard IVF-PQ tuning notions [23]. The *scan budget* is the number of candidate slots evaluated per query,

$$N_{\text{sel}} := n_{\text{probe}} \cdot n,$$

and the *code budget* is the number of bits stored per vector,

$$B := M \log_2 K,$$

where we assume K is a power of two as in standard IVF-PQ deployments. When the padded capacity is fixed to $N := n_{\text{list}} \cdot n$, fixing N_{sel} is equivalent to fixing the probing ratio $r := n_{\text{probe}}/n_{\text{list}} = N_{\text{sel}}/N$, since $N_{\text{sel}} = r \cdot N$. Accordingly, later references to a "fixed scan budget" may specify either N_{sel} or (N, r), depending on context. We use these notions to express both our cost-driven configuration selection and the evaluation sweeps.

Five-step semantics and per-step costs. Under our standardized semantics, query execution follows five steps: (i) compute centroid distances; (ii) select the n_{probe} closest lists; (iii) construct ADC lookup tables for the probed lists; (iv) score all candidate slots via PQ-table lookups; and (v) extract the final top-k payloads. Table 3 summarizes the dominant gate-count terms of each step under the circuit-only baseline and our multiset-based instantiation. Summing the dominant terms yields the gate complexity:

$$\begin{aligned} G_{\text{baseline}} = \Theta\Big(& n_{\text{list}} D + n_{\text{probe}} n_{\text{list}} t_{\text{cmp}} + n_{\text{probe}} KD \\ & + n_{\text{probe}} nMK + k n_{\text{probe}} n t_{\text{cmp}} \Big), \end{aligned} \tag{1}$$

$$\begin{aligned} G_{\text{multiset}} = \Theta\Big(& n_{\text{list}} D + n_{\text{list}} t_{\text{cmp}} + n_{\text{probe}} KD \\ & + t_{\text{cmp}} n_{\text{probe}} M \max\{K, n\} + n_{\text{probe}} n t_{\text{cmp}} \Big). \end{aligned} \tag{2}$$

Let G_{commit} denote the gate count of the snapshot commitment-check circuit. Under a hash-cost abstraction, our analysis yields

$$G_{\text{commit}} = \Theta\Big(KD + n_{\text{list}} D + n_{\text{probe}} (nM + D + \log n_{\text{list}}) \Big). \tag{3}$$

The end-to-end proof circuit composes the query-semantics constraints (Equations 1–2) with the snapshot-binding constraints (Equation 3). This yields the dominant end-to-end gate complexity:

$$\begin{aligned} G^{\text{total}}_{\text{baseline}} = \Theta\Big(& n_{\text{list}} D + n_{\text{probe}} n_{\text{list}} t_{\text{cmp}} + n_{\text{probe}} KD \\ & + n_{\text{probe}} nMK + k n_{\text{probe}} n t_{\text{cmp}} \\ & + n_{\text{probe}} \log n_{\text{list}} \Big), \end{aligned} \tag{4}$$

$$\begin{aligned} G^{\text{total}}_{\text{multiset}} = \Theta\Big(& n_{\text{list}} D + n_{\text{list}} t_{\text{cmp}} + n_{\text{probe}} KD \\ & + t_{\text{cmp}} n_{\text{probe}} M \max\{K, n\} \\ & + n_{\text{probe}} n t_{\text{cmp}} + n_{\text{probe}} \log n_{\text{list}} \Big). \end{aligned} \tag{5}$$

Compared with the query-semantics costs G_{baseline} and G_{multiset}, snapshot binding introduces an additional $\Theta(n_{\text{probe}} \log n_{\text{list}})$ Merkle-authentication term (Equation 3). Experiment 3 shows this overhead is small in our evaluated regimes; we therefore omit the $\Theta(n_{\text{probe}} \log n_{\text{list}})$ term in later scaling discussions unless stated otherwise. We leverage these expressions to derive a configuration-selection procedure under fixed budgets (Section 5.7).

5 CIRCUIT DESIGN

This section describes how V3DB realizes our standardized fixed-shape IVF-PQ semantics as succinct ZK circuits. We prove the verifiable-retrieval statement from Section 4.1: given public inputs $(\text{com}, q, (\text{item}_0, \ldots, \text{item}_{k-1}))$, the prover shows that the returned payload list is exactly the output of the reference semantics on the snapshot version committed by com. The witness includes snapshot values (centroids, PQ codebooks, opened inverted-list records, and Merkle authentication paths) and intermediate trace values required by the semantics; all of these remain hidden by ZK, while verification remains public and cheap.

We construct two end-to-end circuit instantiations for the same statement. The circuit-only baseline directly encodes ordering and random access in-circuit, while our multiset-based design shifts these expensive primitives off-circuit and verifies consistency using multiset equality/inclusion plus lightweight boundary checks. Both instantiations prove the same public interface and differ only in how the reference relations are enforced. We use the shorthand $[Z] := \{0, 1, \ldots, Z-1\}$ to denote integer index ranges.

Overview. To enumerate the circuit design, we proceed from the published fixed-shape IVF-PQ semantics and the snapshot design that binds proofs to a committed version, to the standard gadgets used in our cost notation, and then to two circuit instantiations of the same semantics: a circuit-only baseline and our multiset-based design. We close with the cost-driven configuration rule, which is used later in the experiments. The core novelty lies in replacing the expensive ordering and lookup bottlenecks in Steps 2, 4, and 5 with multiset-based checks, while Steps 1 and 3 remain standard arithmetic constraints.

5.1 Index Shaping via Rebalancing

After standard k-means, inverted lists can be highly imbalanced. Let $\mathcal{X}_i$ denote the set of database vectors assigned to centroid μ_i and let $n_i := |\mathcal{X}_i|$ be its size. A naive fixed-shape layout sets the padded list capacity to the maximum cluster size,

$$n = \max\{n_0, n_1, \ldots, n_{n_{\text{list}}-1}\},$$

Table 3: Dominant asymptotic arithmetic-gate counts of the query-semantics circuit per standardized IVF-PQ step, comparing the circuit-only baseline and our multiset-based instantiation. Notation follows Table 2. Snapshot binding adds an additional $\Theta(n_{\text{probe}} \log n_{\text{list}})$ Merkle-opening overhead (Equation 3).

Step	Circuit-only baseline	Multiset instantiation	Key idea
(i) Centroid distances	$\Theta(n_{\text{list}} D)$	same	—
(ii) Probe selection	$\Theta(n_{\text{probe}} n_{\text{list}} t_{\text{cmp}})$	$\Theta(n_{\text{list}} t_{\text{cmp}})$	multiset equality + boundary checks
(iii) ADC lookup tables	$\Theta(n_{\text{probe}} KD)$	same	—
(iv) Candidate distances	$\Theta(n_{\text{probe}} nMK)$	$\Theta(t_{\text{cmp}} n_{\text{probe}} M \max\{K, n\})$	multiset inclusion + comparisons
(v) Final Top-k selection	$\Theta(k n_{\text{probe}} n t_{\text{cmp}})$	$\Theta(n_{\text{probe}} n t_{\text{cmp}})$	multiset equality + boundary checks

which is unstable: a single outlier cluster inflates padding for all lists and increases both the scan cost and the circuit size.

Instead, V3DB fixes a hard capacity bound n in advance and post-processes the initial assignments using a lightweight rebalancing procedure inspired by [6] to enforce a uniform per-list bound:

$$|\mathcal{X}_i| \le n \qquad \text{for all } i \in [n_{\text{list}}].$$

With $N := n_{\text{list}} \cdot n \ge N_0$, Algorithm 1 iteratively moves points out of overfull clusters to the nearest available underfull clusters until all clusters satisfy the bound. We then pad each list to exactly n slots and attach a validity flag $f_{i,j} \in \{0,1\}$ for each slot. Each padded slot stores a record $(f_{i,j}, \text{item}_{i,j}, v_{i,j}^{\text{PQ}})$, where $\text{item}_{i,j}$ is an application-defined payload and $v_{i,j}^{\text{PQ}}$ is the PQ code; for padding slots ($f_{i,j} = 0$), the remaining fields may be arbitrary.

Algorithm 1 Capacity-constrained cluster rebalancing to enforce the per-list bound $|\mathcal{X}_i| \le n$ in the fixed-shape IVF layout.

Require: Centroid table $\boldsymbol{\mu} = (\mu_0, \mu_1, \ldots, \mu_{n_{\text{list}}-1})$
Require: Clustered points $\mathcal{X}_i = \{v_{i,j} \mid j \in [n_i]\}$ for each cluster i
Require: Capacity bound n
Ensure: Updated clusters $\{\mathcal{X}_i\}$ such that $|\mathcal{X}_i| \le n$ for all i
  while $\exists\, i$ s.t. $n_i > n$ **do**
    $O \leftarrow \{i \mid n_i > n\}$, $F \leftarrow \{t \mid n_t < n\}$ ▷ *overfull / free clusters*
    $\mathcal{M} \leftarrow \emptyset$ ▷ *candidate moves*
    for each $i \in O$ **do**
      for each $v \in \mathcal{X}_i$ **do**
        $t^\star \leftarrow \arg\min_{t \in F} \|v - \mu_t\|_2^2$
        $\Delta(v,i) \leftarrow \|v - \mu_{t^\star}\|_2^2 - \|v - \mu_i\|_2^2$
        Add $(v, i, t^\star, \Delta(v,i))$ to $\mathcal{M}$
      end for
    end for
    Sort $\mathcal{M}$ by increasing Δ
    for each $(v, i, t, \Delta) \in \mathcal{M}$ in order **do**
      if $n_i > n$ **and** $n_t < n$ **and** $v \in \mathcal{X}_i$ **then**
        $\mathcal{X}_i \leftarrow \mathcal{X}_i \setminus \{v\}$, $\mathcal{X}_t \leftarrow \mathcal{X}_t \cup \{v\}$
        $n_i \leftarrow n_i - 1$, $n_t \leftarrow n_t + 1$
      end if
    end for
  end while
  return $\{\mathcal{X}_i\}_{i=0}^{n_{\text{list}}-1}$

5.2 Fixed-Shape Five-Step IVF-PQ Semantics

We specify the fixed-shape snapshot model used by our circuits and by the reference semantics. The parameters are: database size N_0, padded capacity $N := n_{\text{list}} \cdot n$, vector dimension D, number of inverted lists n_{list}, number of probed lists n_{probe}, top-k size k, per-list capacity n, and PQ parameters (M, K) where $D = Md$ and each block uses a codebook of size K. The committed snapshot contains:

- *Centroids* $\boldsymbol{\mu} = (\mu_0, \ldots, \mu_{n_{\text{list}}-1})$ with $\mu_i \in \mathbb{F}^D$.
- *PQ codebooks* $\{C_{m,k}\}_{m \in [M], k \in [K]}$ with $C_{m,k} \in \mathbb{F}^d$.
- *Fixed-shape inverted lists.* For each list $i \in [n_{\text{list}}]$ and slot $j \in [n]$, the snapshot stores a record $(f_{i,j}, \text{item}_{i,j}, v_{i,j}^{\text{PQ}})$, where $f_{i,j} \in \{0,1\}$ is a validity flag, $\text{item}_{i,j}$ is an application-defined payload, and $v_{i,j}^{\text{PQ}} = (v_{i,j}^{(0)}, \ldots, v_{i,j}^{(M-1)}) \in [K]^M$ is the PQ code. If $f_{i,j} = 0$, the slot is padding and the remaining fields are arbitrary.

We use squared ℓ_2 distance over $\mathbb{F}$ throughout:

$$\text{dist}(x, y) := \sum_{t=0}^{D-1} (x_t - y_t)^2.$$

We now specify the IVF-PQ query computation that our proof system attests to in each challenged audit instance. Given a query $q \in \mathbb{F}^D$ and a snapshot commitment com, the prover claims that there exists a fixed-shape snapshot consistent with com such that the public output is a length-k list of payloads $(\text{item}_0, \ldots, \text{item}_{k-1})$ produced by the following five-step procedure. For later reference, Steps 1 and 3 are arithmetic constraints, while Steps 2 and 5 are ordering bottlenecks and Step 4 is the lookup bottleneck optimized by our multiset-based design.

Step 1. Centroid distances. Define centroid distances

$$d_i := \text{dist}(q, \mu_i), \qquad i \in [n_{\text{list}}],$$

and the index–distance sequence

$$S_{\text{idx,dist}}^{\text{orig}} := \big((0, d_0), (1, d_1), \ldots, (n_{\text{list}} - 1, d_{n_{\text{list}}-1})\big).$$

Step 2. Probe selection. Sort $S_{\text{idx,dist}}^{\text{orig}}$ by the distance to obtain

$$S_{\text{idx,dist}}^{\text{sorted}} := \big((i_0, d_{i_0}), (i_1, d_{i_1}), \ldots, (i_{n_{\text{list}}-1}, d_{i_{n_{\text{list}}-1}})\big),$$

where $(i_0, \ldots, i_{n_{\text{list}}-1})$ is a permutation of $[n_{\text{list}}]$. We only require the first n_{probe} entries to be ordered and to have distances no larger than the remaining ones:

$$d_{i_0} \le d_{i_1} \le \cdots \le d_{i_{n_{\text{probe}}-1}} \quad \text{and}$$
$$d_{i_{n_{\text{probe}}-1}} \le d_{i_t} \quad \text{for all } t \in \{n_{\text{probe}}, \ldots, n_{\text{list}} - 1\}.$$

The probed set is defined as the first n_{probe} indices,

$$P(q) := (i_0, \ldots, i_{n_{\text{probe}}-1}), \qquad |P(q)| = n_{\text{probe}}.$$

Step 3. ADC lookup tables. For each probed list $i \in P(q)$, form the residual query $q - \mu_i$, split it into M blocks $(q - \mu_i)^{(m)} \in \mathbb{F}^d$ (recall $D = Md$), and define the ADC lookup tables

$$\text{LUT}_{i,m,k} := \text{dist}\big(C_{m,k}, (q - \mu_i)^{(m)}\big), \qquad m \in [M],\ k \in [K].$$

Step 4. Candidate distances. For each $i \in P(q)$ and each slot $j \in [n]$, define the ADC distance

$$\widetilde{d}_{i,j} := \sum_{m=0}^{M-1} \mathsf{LUT}_{i,m,v_{i,j}^{(m)}}.$$

To ensure padded entries never affect the result, fix a public constant $d_{\max}$ that exceeds any attainable $\widetilde{d}_{i,j}$ for valid slots (under the enforced range bounds), and define the masked distance

$$D_{i,j} := f_{i,j} \cdot \widetilde{d}_{i,j} + (1 - f_{i,j}) \cdot d_{\max}.$$

Collect all candidate pairs into the flattened sequence

$$S^{\text{orig}}_{\text{item,dist}} := \big((\text{item}_{i,j}, D_{i,j}) \,\big|\, i \in P(q),\ j \in [n]\big),$$

which has length N_{sel} (i.e., the scan budget).

Step 5. Final top-k selection. View $S^{\text{orig}}_{\text{item,dist}}$ as a length-N_{sel} sequence of pairs. Sort it by the distance component to obtain

$$S^{\text{sorted}}_{\text{item,dist}} := \big((\text{item}_0, D_0), (\text{item}_1, D_1), \ldots, (\text{item}_{N_{\text{sel}}-1}, D_{N_{\text{sel}}-1})\big),$$

where we only require the first k entries to be ordered and to have distances no larger than the remaining ones:

$$D_0 \le D_1 \le \cdots \le D_{k-1} \quad \text{and}$$
$$D_{k-1} \le D_t \quad \text{for all } t \in \{k, \ldots, N_{\text{sel}} - 1\}.$$

The query output is the first k items in this order:

$$(\text{item}_0, \text{item}_1, \ldots, \text{item}_{k-1}).$$

5.3 Snapshot Design

To bind each proof to a fixed IVF-PQ snapshot version while keeping the snapshot contents private, V3DB uses a unified snapshot design: a public commitment object com together with circuit-side checks against it. We instantiate com with SNARK-friendly Poseidon-based hashing and Merkle trees [20, 35]. The commitment has two parts: (i) a hierarchical Merkle root root_{mk} over the fixed-shape IVF layout (centroids and padded inverted lists) and (ii) a hash digest root_{cb} over the PQ codebooks. We write $\text{com} := (\text{root}_{\text{mk}}, \text{root}_{\text{cb}})$.

Role of the commitment. The functionality of com is binding rather than endorsement: an accepting proof shows consistency with this particular published snapshot identifier, but the proof does not by itself certify that the publisher selected the authorized snapshot. Figure 2 summarizes the hierarchy reused by the circuit to authenticate all centroids and only the probed inverted lists.

Commitment structure. For each list $i \in [n_{\text{list}}]$ and slot $j \in [n]$, define the leaf digest

$$\text{hash}_{i,j} := \mathsf{Hash}\big(i, j, f_{i,j}, \text{item}_{i,j}, v_{i,j}^{\text{PQ}}\big),$$

where $\mathsf{Hash}(\cdot)$ absorbs the M components of $v_{i,j}^{\text{PQ}}$ (from $v_{i,j}^{(0)}$ to $v_{i,j}^{(M-1)}$) in a fixed canonical order. Let root_i denote the Merkle root of list i. We then bind each centroid together with its list root via

$$\text{hash}_i := \mathsf{Hash}(i, \mu_i, \text{root}_i),$$

where $\mathsf{Hash}(i, \mu_i, \text{root}_i)$ absorbs the D coordinates of μ_i in a fixed canonical order, and define the top-level layout commitment as

$$\text{root}_{\text{mk}} := \mathsf{MerkleTree}(\text{hash}_0, \ldots, \text{hash}_{n_{\text{list}}-1}).$$

Since PQ codebooks are static once trained, we bind them separately by hashing a canonical flattening of all codewords. For simplicity, we assume both n and n_{list} are powers of two so the Merkle trees are full and balanced; otherwise, the snapshot layer pads to the next power of two.

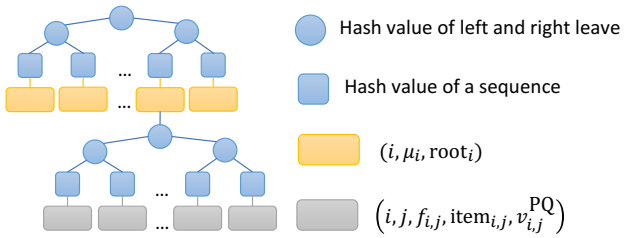

Figure 2: Snapshot design for the fixed-shape IVF layout. The public identifier $\text{com} = (\text{root}_{\text{mk}}, \text{root}_{\text{cb}})$ binds the centroids, padded inverted-list records, and PQ codebooks; during proving, the circuit authenticates all centroids and the opened probed lists against this hierarchy.

Circuit-side binding checks. The end-to-end statement binds the witness values used by the five steps to the public commitment (i.e., com): centroids for Steps 1–2, PQ codebooks for Step 3, and the slots of the probed inverted lists for Step 4 of the current query. The circuit first recomputes the codebook digest from the witness codebooks, which binds the lookup tables used in Step 3. We then recompute the top-level layout root from all witness centroids, along with the witness list-root digests, thereby binding the centroid table used in Steps 1–2 and fixing the committed digests of all inverted lists. Finally, for each probed list $i \in P(q)$, it recomputes the list root from the opened slots and authenticates that root to root_{mk} using a standard MerkleOpen check at position i. Since Step 4 consumes all n slots of each probed list under our fixed-shape semantics, no per-slot authentication paths are needed: authenticating the list root suffices to supply the concrete slot records consumed by the query trace.

Cost. The resulting snapshot-binding cost is summarized by Equation 3; it composes additively with the query-semantics costs discussed in the following query-circuit subsections.

5.4 Gadgets and Cost Model

We express all constraints as arithmetic relations over a field $\mathbb{F}$ and track circuit size via arithmetic gate counts. Witness values are range-bounded to avoid wraparound in comparisons; we write t_{cmp} for the bit-length parameter used by our range-bounded comparison gadget in the circuit.

Many subcircuits used by V3DB are standard (e.g., fixed-point arithmetic for distance/LUT construction, Poseidon hashing, and Merkle-path verification). Since our main optimization is to replace in-circuit control flow with multiset relations, we highlight the *set-oriented* gadgets that are central to the multiset-based design and to our gate-count analysis.

Tuple compression (Compress). We reduce tuple-valued relations to set relations over single field elements via a transcript-derived challenge $\beta \in \mathbb{F}$. For a tuple $\mathbf{x} = (x_0, \ldots, x_{L-1})$, we define

$$\mathsf{Compress}(\mathbf{x}; \beta) = \sum_{i=0}^{L-1} x_i \beta^i,$$

Table 4: Core gadgets and their asymptotic arithmetic-gate complexity. Their informal roles are tuple compression, bounded comparison, one pass of a comparison network, swap propagation to a parallel list, multiset equality, multiset inclusion, and table lookup, respectively. Here L is the tuple/list/table length, t_{cmp} is the comparison bit-length, and $n_{\max} = \max\{|X|, |Y|\}$ for Incl.

Gadget	Gate complexity	Gadget	Gate complexity
Compress	$\Theta(L)$	SetEq	$\Theta(L)$
Cmp	$\Theta(t_{\text{cmp}})$	Incl	$\Theta(t_{\text{cmp}}\, n_{\max})$
BubblePass	$\Theta(L\, t_{\text{cmp}})$	LookUp	$\Theta(L)$
Permute	$\Theta(L)$	—	—

which can be evaluated via Horner's rule using $L-1$ multiplications and $L-1$ additions, for a total of $2L-2 = \Theta(L)$ arithmetic gates.

Range-bounded comparison (Cmp). Let $t_{\text{cmp}} = \lfloor \log_2 |\mathbb{F}| \rfloor$ and assume $x, y \in [0, 2^{t_{\text{cmp}}-1})$. We form

$$\Delta = x - y + 2^{t_{\text{cmp}}-1} \in (0, 2^{t_{\text{cmp}}}).$$

Enforcing a t_{cmp}-bit decomposition of Δ yields a boolean output indicating whether $x < y$. The resulting gate cost is $\Theta(t_{\text{cmp}})$.

Multiset equality (SetEq). After compressing tuples, multiset equality between two size-L sequences $X = \{x_0, \ldots, x_{L-1}\}$ and $Y = \{y_0, \ldots, y_{L-1}\}$ can be enforced via a single polynomial identity with a random challenge $\alpha \in \mathbb{F}$:

$$\prod_{i=0}^{L-1} (\alpha - x_i) = \prod_{i=0}^{L-1} (\alpha - y_i).$$

Evaluating one side uses L subtractions and $L-1$ multiplications, so evaluating both sides costs $4L-2 = \Theta(L)$ arithmetic gates. This captures the essence of permutation-style arguments used in PLONK systems [18].

Multiset inclusion (Incl). To prove multiset inclusion $X \subset_M Y$ with multiplicities, let $n_{\max} = \max\{|X|, |Y|\}$ and pad both multisets to length $n_{\max}$. The prover supplies: (i) a nondecreasing ordering X'' of the padded X and (ii) an aligned sequence Y'' whose multiset equals the padded Y. The circuit enforces: (a) $\mathsf{SetEq}(X, X'')$ and $\mathsf{SetEq}(Y, Y'')$; (b) nondecreasing order of X'' via $(n_{\max}-1)$ comparisons; and (c) multiplicity alignment via $x''_0 = y''_0$ and $(x''_i - x''_{i-1})(x''_i - y''_i) = 0$ for all $i \geq 1$. Overall, this uses two SetEq relations, $(n_{\max}-1)$ comparisons, and $O(n_{\max})$ alignment constraints. The comparisons dominate, giving an asymptotic cost $\Theta(t_{\text{cmp}}\, n_{\max})$. This gadget is a circuit-level abstraction of lookup-style arguments in PLONK systems [17].

Table 4 summarizes the asymptotic gate complexity of the above set-oriented gadgets and also lists a few additional primitives used by the baseline circuit-only instantiation; we treat the latter as standard and do not expand their constraints.

We now use these gadgets to describe two query-circuit instantiations: the circuit-only baseline and our multiset-based design. Both instantiations prove the same public statement and enforce the same fixed-shape five-step IVF-PQ semantics. They share the arithmetic constraints in Steps 1 and 3, and differ in how they enforce the ordering and lookup relations in Steps 2, 4, and 5.

5.5 Baseline Circuit-Only Design

We present the baseline first as a controlled decomposition: every data-dependent operation stays *inside* the circuit, making the direct costs of ordering and lookup explicit before introducing the multiset replacement. Concretely, it realizes the ordering requirements in probe selection (Step 2) and final top-k extraction (Step 5) using explicit comparison networks, realizes PQ-table access in candidate scoring (Step 4) via in-circuit random access, and keeps Steps 1 and 3 as standard fixed-point arithmetic constraints.

In-circuit ordering and selection. To enforce the Step 2 and Step 5 partial-order conditions, the baseline applies comparison-network passes (e.g., BubblePass) to partially sort the corresponding distance sequences, while applying the same swaps to the associated index/payload sequences via Permute. This yields a cost of $\Theta(n_{\text{probe}} n_{\text{list}} t_{\text{cmp}})$ gates for probe selection and $\Theta(k n_{\text{probe}} n t_{\text{cmp}})$ gates for the final top-k extraction.

In-circuit PQ-table access. For each candidate slot and each sub-quantizer, the baseline selects $\mathsf{LUT}_{i,m,v_{i,j}^{(m)}}$ from a length-K table via a LookUp gadget and sums the selected entries to obtain the approximate distance, with invalid (padding) slots masked to a public upper bound. These per-candidate table lookups dominate Step 4, contributing $\Theta(n_{\text{probe}} nMK)$ gates.

Gate complexity. Overall, the baseline query-semantics circuit is dominated by comparison networks (Steps 2 and 5) and in-circuit random access (Step 4), leading to the gate complexity summarized in Equation 1. Composing it with snapshot binding (Equation 3) yields the end-to-end total in Equation 4. These bottlenecks motivate our multiset-based instantiation below, which proves the same semantics while avoiding in-circuit sorting and random access.

5.6 Multiset-Based Design

Compared with the baseline, the multiset-based design keeps the arithmetic constraints unchanged and makes three targeted substitutions: Step 2 replaces in-circuit partial sorting with multiset equality plus boundary checks, Step 4 replaces per-candidate random access with a single inclusion relation over selected LUT tuples, and Step 5 again uses multiset equality plus boundary checks. The prover executes the IVF-PQ pipeline off-circuit to obtain the sorted intermediate sequences and the LUT contributions selected by PQ codes. The circuit recomputes the arithmetic constraints (centroid distances and LUT tables) and verifies that the supplied trace is consistent with these base quantities using multiset relations plus lightweight order/boundary checks. This suffices because, in our reference semantics, sorting/selection and random access only serve to justify partial-order conditions and lookup consistency.

Shared steps (same arithmetic constraints as the baseline). Steps 1 and 3 are identical to the baseline: the circuit recomputes all centroid distances d_i and the full ADC tables $\mathsf{LUT}_{i,m,k}$ using fixed-point distance constraints in the same way. Moreover, the arithmetic part of Step 4 (summing M LUT entries and applying the validity mask with $d_{\max}$) is unchanged; the only difference in Step 4 is how we justify that the prover-selected LUT entries are consistent with the circuit-recomputed tables.

Step 2. Probe selection via multiset equality + boundary checks. The circuit computes the centroid distances $\{d_i\}$ as in

Step 1 and receives a witness sequence:

$$S^{\text{sorted}}_{\text{idx,dist}} = \big((i_0, d_{i_0}), (i_1, d_{i_1}), \ldots, (i_{n_{\text{list}}-1}, d_{i_{n_{\text{list}}-1}})\big).$$

It then enforces the Step 2 relation by composing gadgets as follows in the circuit. First, it applies Compress to each pair (i_t, d_{i_t}) (a length-2 tuple) and uses SetEq to prove multiset equality between $S^{\text{sorted}}_{\text{idx,dist}}$ and $S^{\text{orig}}_{\text{idx,dist}}$; since indices in $S^{\text{orig}}_{\text{idx,dist}}$ are unique, this implies that $(i_0, \ldots, i_{n_{\text{list}}-1})$ is a permutation of $[n_{\text{list}}]$ and binds each claimed distance to the circuit-computed d_i. Second, it enforces the Step 2 partial-order conditions using Cmp: $(n_{\text{probe}} - 1)$ comparisons enforce $d_{i_0} \le \cdots \le d_{i_{n_{\text{probe}}-1}}$, and $(n_{\text{list}} - n_{\text{probe}})$ comparisons enforce $d_{i_{n_{\text{probe}}-1}} \le d_{i_t}$ for all $t \in \{n_{\text{probe}}, \ldots, n_{\text{list}} - 1\}$. This realizes Step 2 with $\Theta(n_{\text{list}} t_{\text{cmp}})$ comparisons, matching the probe-selection term in Equation 2 in our analysis.

Step 4. Candidate distances via multiset inclusion. The circuit constructs the full ADC lookup tables $\mathsf{LUT}_{i,m,k}$ for all probed lists as in Step 3. Instead of performing an in-circuit LookUp for each PQ index, the prover supplies the selected LUT entries $\ell^{(m)}_{i,j} = \mathsf{LUT}_{i,m,v^{(m)}_{i,j}}$ as witness values. The circuit proves the correctness of these witness entries by a multiset inclusion check. After compressing tuples, it enforces that the multiset

$$\big\{(i, m, v^{(m)}_{i,j}, \ell^{(m)}_{i,j}) \;\big|\; i \in P(q),\ j \in [n],\ m \in [M]\big\}$$

is included in the multiset of all LUT tuples

$$\big\{(i, m, k, \mathsf{LUT}_{i,m,k}) \;\big|\; i \in P(q),\ m \in [M],\ k \in [K]\big\}.$$

It then computes masked candidate distances $D_{i,j}$ using $d_{\max}$ and the validity flags $f_{i,j}$. Since the selected-entry multiset has size $|X| = n_{\text{probe}} nM$ while the full LUT multiset has size $|Y| = n_{\text{probe}} MK$, the inclusion gadget operates on $n_{\max} = n_{\text{probe}} M \max\{n, K\}$ elements, yielding the $\Theta(t_{\text{cmp}} n_{\text{probe}} M \max\{n, K\})$ term in Equation 2.

Step 5. Final top-k selection via multiset equality + boundary checks. Given the flattened candidate list $S^{\text{orig}}_{\text{item,dist}}$, the prover supplies a sorted witness sequence $S^{\text{sorted}}_{\text{item,dist}}$. The circuit applies Compress (a length-2 tuple) and uses SetEq to prove multiset equality between the original and sorted candidate sequences. It then enforces the Step 5 top-k partial-order conditions using Cmp: $(k-1)$ comparisons enforce $D_0 \le \cdots \le D_{k-1}$, and $(N_{\text{sel}} - k)$ comparisons enforce $D_{k-1} \le D_t$ for all $t \in \{k, \ldots, N_{\text{sel}} - 1\}$. Finally, it outputs the first k payloads in the provided order, which are then certified by the proof.

Gate complexity. The multiset-based query-circuit complexity is summarized by Equation 2: compared to the baseline, the probe-selection and top-k terms drop from bubble-pass networks to linear-sized boundary checks, and the per-vector LUT lookups are replaced by a single multiset inclusion relation whose size scales with $\max\{K, n\}$ rather than with nK per candidate. The corresponding end-to-end total complexity is given by Equation 5.

5.7 Cost-Driven Configuration Selection

Our gate-count analysis enables a simple, deployment-oriented configuration-selection procedure. Unlike plaintext IVF-PQ tuning, which primarily optimizes retrieval utility under latency constraints, V3DB must additionally account for the cost of generating audit proofs. Proving time is governed by the padded evaluation-domain size $G_B := 2^{\lceil \log_2 G \rceil}$, where G is the circuit gate count; thus, we target minimizing G_B rather than the exact gate count.

Budgets and knobs. We use the scan budget $N_{\text{sel}} := n_{\text{probe}} \cdot n$ and the code budget $B := M \log_2 K$. In our fixed-shape setting, we treat the padded capacity $N := n_{\text{list}} \cdot n$ and the probing ratio $r := n_{\text{probe}}/n_{\text{list}}$ as deployment-level choices. Under a fixed code budget B and power-of-two K, the number of sub-quantizers is determined as $M = B/\log_2 K$. Accordingly, our main knobs are (n_{list}, K), which jointly determine (n_{probe}, n, M).

Fixed scan budget: near-linear scaling in n_{list}. Under fixed N and fixed r, we have $n = N/n_{\text{list}}$ and $n_{\text{probe}} = r\, n_{\text{list}}$. Substituting these relations into the end-to-end cost in Equation 5 shows that, for fixed (B, K) and in the common regime $K \le n$, the query-semantics terms are near-linear in n_{list}, while the remaining terms depend only on the fixed scan budget N_{sel}. Snapshot binding contributes an additive $\Theta(n_{\text{probe}} \log n_{\text{list}}) = \Theta(r\, n_{\text{list}} \log n_{\text{list}})$ term due to per-probed-list Merkle openings (Equation 3). In the parameter ranges we study, this sublinear factor has a negligible impact on the observed end-to-end scaling (validated in Experiment 3).

Fixed code budget: a unimodal dependence on K. Under a fixed code budget B (so $M = B/\log_2 K$) and fixed scan budget, the K-dependent terms in Equation 5 can be viewed as a trade-off between: (i) ADC-table construction, which scales as $\Theta(n_{\text{probe}} KD)$ and increases with K; and (ii) candidate scoring, whose multiset-inclusion component scales as $\Theta(t_{\text{cmp}}\, n_{\text{probe}} M \max\{K, n\})$. In our regime $K \le n$, the latter simplifies to $\Theta(t_{\text{cmp}}\, N_{\text{sel}} \cdot B/\log_2 K)$ and decreases with K, implying a unimodal total cost in K. For completeness, when $K > n$ the inclusion term becomes $\Theta(t_{\text{cmp}}\, n_{\text{probe}} B \cdot K/\log_2 K)$ and is increasing, so the total cost grows beyond $K = n$.

Bin-pruned search. To operationalize these observations, we use a simple search procedure that minimizes the padded bin G_B under fixed deployment budgets (N, B, r) for a target workload. When multiple configurations fall into the same smallest bin, we choose the one with the largest (n_{list}, K) to preserve retrieval utility, since larger n_{list} (under fixed ratio) and larger K (under fixed B) typically improve retrieval quality in plaintext IVF-PQ [23].

6 EXPERIMENTS

We evaluate V3DB's end-to-end workflow for IVF-PQ top-k retrieval over committed snapshots. Our evaluation covers retrieval utility under ZK-friendly preprocessing, proof-generation cost, and ZK-aware configuration trade-offs.

Evaluation environment. All experiments are run on a dedicated dual-socket AMD EPYC server with 256 hardware cores and 1 TiB RAM. We use the default Plonky2 configuration as in the rest of the paper (field, FRI parameters, and recursion layout).

Budgets and bounds. We use the scan budget N_{sel} and code budget B to parameterize IVF-PQ configurations. Prior work suggests that, under these budgets, larger n_{list} (under a fixed probing ratio) and larger K (under a fixed B) often improve retrieval quality in plaintext IVF-PQ [23]; in V3DB, we must additionally account for proving cost. Unless stated otherwise, we fix $B = 64$ and the probing ratio $r := n_{\text{probe}}/n_{\text{list}} = 1/128$, which fixes the scan budget once the padded capacity N is set. We restrict to $n_{\text{list}} \le 8192$ and $K \le 256$.

Evaluation questions. We evaluate three research questions:

Algorithm 2 Pruned search for a configuration that minimizes the padded bin G_B under fixed code budget B and probing ratio r.

Require: Padded capacity N, code budget B, probing ratio $r := n_{\text{probe}}/n_{\text{list}}$
Require: Max $n_{\text{list}}^{\max}$, candidate codebooks $\mathcal{K}$ (powers of two; $M = B/\log_2 K \in \mathbb{N}$)
Require: Gate estimator $\text{GateCount}(D, n_{\text{list}}, K)$
Ensure: Smallest bin $G_B^\star$ and a zk-opt configuration $(n_{\text{list}}^\star, K^\star)$ within that bin

```
N_sel ← r · N                                   ▷ fixed scan budget
n_list ← N / N_sel                              ▷ minimum feasible layout; then n_probe = 1
for each K ∈ 𝒦 do
    G ← GateCount(D, n_list, K)
    G_B(K) ← 2^⌈log2 G⌉
end for
G_B^⋆ ← min_{K∈𝒦} G_B(K)
𝒦 ← {K ∈ 𝒦 | G_B(K) = G_B^⋆}
(n_list^⋆, K^⋆) ← (n_list, max 𝒦)
while 𝒦 ≠ ∅ and n_list < n_list^max do
    n_list ← 2 n_list
    for each K ∈ 𝒦 do
        G ← GateCount(D, n_list, K)
        G_B(K) ← 2^⌈log2 G⌉
        if G_B(K) > G_B^⋆ then
            Remove K from 𝒦                     ▷ K leaves the smallest bin
        end if
    end for
    if 𝒦 ≠ ∅ then
        (n_list^⋆, K^⋆) ← (n_list, max 𝒦)
    end if
end while
return (G_B^⋆, (n_list^⋆, K^⋆))
```

- **RQ-(i).** *How does ZK-friendly preprocessing affect retrieval utility?* Concretely, after applying fixed-point encoding and fixed-shape rebalancing/padding, how much do accuracy metrics change compared to a floating-point IVF-PQ pipeline?
- **RQ-(ii).** *Does our multiset-based design make audit proofs practical?* Concretely, how much does the optimized design reduce gate count and proving time compared to the baseline?
- **RQ-(iii).** *Do measured proving costs match our analytic predictions, and does our tuning procedure yield ZK-efficient deployments?* Concretely, under fixed scan/code budgets, do the observed gate counts and proving times follow the trends predicted by our cost analysis, and does Algorithm 2 identify practical zk-opt configurations?

6.1 Retrieval Utility Evaluation

Experiment 1 answers Question (i) by evaluating whether our ZK-friendly IVF-PQ preprocessing pipeline preserves retrieval utility. We evaluate two benchmark families—classic ANN and information retrieval (IR)—and report standard metrics. Comprehensively, Experiment 1 evaluates whether the ZK-oriented fixed-point encoding and capacity-constrained rebalancing steps materially change retrieval utility relative to the same floating-point IVF-PQ pipeline. Our goal is not to propose a more accurate ANN variant, but to preserve the utility of the matched floating-point pipeline as closely as possible after ZK-oriented preprocessing. We therefore compare the std and zk pipelines under matched configurations and use the standard utility metrics for each benchmark family.

Benchmarks and metrics. We evaluate two benchmark families:

- *Classic ANN benchmarks (SIFT1M and GIST1M) [23].* SIFT1M ($D = 128$) and GIST1M ($D = 960$) provide exact ℓ_2 nearest-neighbor. The Recall@k is reported for $k \in \{1, 10, 50, 100\}$, where Recall@k is the fraction of queries whose exact nearest neighbor appears in the top-k retrieved list [23]. Multiple cutoffs make the dependence on k explicit and give a fuller view of retrieval quality. We also report index construction time and relocated-vector counts.
- *IR benchmark (MS MARCO passage retrieval, dev queries) [36].* We keep dev queries with a single relevant passage, embed queries and passages with `msmarco-MiniLM-L6-v3` [40, 43], and normalize embeddings so cosine ranking matches squared-ℓ_2 ranking. We retrieve top-10 passages and report Hit@10, MRR@10, and NDCG@10 [12]; Hit@k is the relevant-passage hit rate, MRR@k is truncated reciprocal rank, and NDCG@k uses binary relevance. This setup serves as an application-oriented outsourced passage-retrieval workload, complementing the synthetic ANN benchmarks above.

Compared systems. We compare two prototypes:

- *Std-IVF-PQ (std):* a standard floating-point IVF-PQ pipeline.
- *Zk-IVF-PQ (zk):* our ZK-friendly pipeline, which applies fixed-point encoding to all vectors and then runs capacity-constrained rebalancing prior to building the IVF-PQ index.

We use the ℓ_2 distance semantics throughout. Following our fixed-point representation, we instantiate a 16-bit fixed-point scale in our prototype. Concretely, we take the maximum absolute coordinate value $v_{\max}$ over all involved real vectors and centroids, rescale it to $2^{16} - 1$, and encode each coordinate v as $\lfloor (2^{16} - 1) \cdot v / v_{\max} \rceil$ (round-to-nearest integer) before embedding it into the field $\mathbb{F}$.

Configurations. For each dataset, we test two IVF-PQ settings:

- *High-acc:* an accuracy-oriented configuration that maximizes (n_{list}, K) within our global bounds ($n_{\text{list}} \le 8192$ and $K \le 256$).
- *Zk-opt:* a ZK-oriented configuration selected by Algorithm 2 to minimize the padded bin G_B under the same fixed budgets.

Table 5: Experiment 1 datasets and IVF-PQ configurations. N_0 is the raw dataset size, N is the capacity after padding, and each configuration is reported as $(n_{\text{list}}, n_{\text{probe}}, n)$ for high-acc and zk-opt. PQ parameters are $(M, K) = (8, 256)$ throughout.

Dataset	D	N_0	N	Q	high-acc	zk-opt
SIFT1M	128	10^6	2^{21}	10 000	(8192, 64, 256)	(1024, 8, 2048)
GIST1M	960	10^6	2^{21}	1 000	(8192, 64, 256)	(512, 4, 4096)
MS MARCO dev	384	8.8×10^6	2^{24}	52 442	(8192, 64, 2048)	(2048, 16, 8192)

Summary. Experiment 1 spans three datasets (SIFT1M, GIST1M, and MS MARCO dev) and compares two pipelines (std vs. zk) under two configurations (high-acc and zk-opt); Table 5 summarizes the dataset statistics and the corresponding configurations. Since each dataset is evaluated over a large set of queries, we report results computed over the full query set and omit confidence intervals.

Classic ANN benchmarks (SIFT1M and GIST1M). Table 6 reports Recall@k for $k \in \{1, 10, 50, 100\}$ under the four ANN configurations, making the dependence on the cutoff explicit for SIFT1M and GIST1M. Across the reported cutoffs, the zk pipeline remains close to the floating-point baseline, while the main additional cost is offline index construction due to rebalancing.

IR benchmark (MS MARCO passage retrieval). Table 7 reports the IR metrics under the Experiment 1 setup (Table 5). This

Table 6: Classic ANN benchmarks (SIFT1M and GIST1M): std-IVF-PQ (std) vs. zk-IVF-PQ (zk) under the high-acc and zk-opt configurations in Table 5. We report Recall@k for $k \in \{1, 10, 50, 100\}$, together with index construction time and the number of relocated vectors during rebalancing (zk only).

Config	R@1		R@10		R@50		R@100		train		moved
	std	zk	std	zk	std	zk	std	zk	std	zk	zk
SIFT1M											
high-acc	0.3292	0.3315	0.7458	0.7456	0.9313	0.9291	0.9527	0.9574	95.13	207.39	9 165
zk-opt	0.2903	0.2892	0.6723	0.6745	0.8421	0.8372	0.8719	0.8686	31.33	38.76	4 461
GIST1M											
high-acc	0.1000	0.1040	0.2590	0.2630	0.4370	0.4580	0.5310	0.5230	341.56	4542.81	39 345
zk-opt	0.0900	0.0820	0.1990	0.1870	0.3390	0.3260	0.4140	0.4190	77.99	293.26	28 986

Table 7: Utility of std-IVF-PQ (std) vs. zk-IVF-PQ (zk) on MS MARCO passage retrieval (dev). We report Hit@10, MRR@10, NDCG@10, training time, and relocated-vector count.

Config	Hit@10		MRR@10		NDCG@10		train [s]		moved
	std	zk	std	zk	std	zk	std	zk	zk
high-acc	0.3609	0.3601	0.1914	0.1917	0.2315	0.2316	949.91	3559.39	43 733
zk-opt	0.3436	0.3443	0.1807	0.1808	0.2194	0.2197	274.38	662.77	16 767

application-facing setup is evaluated with the standard cutoff-10 IR metrics commonly used in retrieval evaluation. ZK-friendly preprocessing preserves IR utility: Hit@10, MRR@10, and NDCG@10 closely track the floating-point baseline under both configurations. As expected, the zk-opt configuration trades some accuracy for ZK efficiency compared to the high-acc configuration.

Analysis. The std-to-zk gaps remain small overall, which is consistent with the mild preprocessing overheads in these two procedures: *Fixed-point encoding* uses the largest scale to avoid modular wraparound, so its rounding error is small; *Rebalancing* affects only overflow points and moves them to nearby underfull lists, and the moved counts take a small fraction of the indexed vectors across all reported settings. Together, these properties explain why the zk pipeline closely tracks the floating-point baseline across both the classic ANN and IR benchmarks, while the more visible cost of the ZK-oriented preprocessing is the increase in offline index construction time due to rebalancing.

> **Answer to RQ-(i).** *ZK-friendly preprocessing largely preserves retrieval utility across our benchmarks: recall- and ranking-based metrics track a floating-point IVF-PQ baseline under both high-acc and zk-opt configurations. The main cost is offline index shaping (rebalancing and padding), which increases index construction time, especially at high n_{list}.*

6.2 Proof Cost Evaluation

Experiment 2 answers Question (ii) by benchmarking the circuit-only IVF-PQ baseline against our multiset-based design. Since verifiability requires binding the proof to a committed snapshot, all results include the snapshot commitment subcircuit.

Configurations. We consider three IVF-PQ proof-cost operating points and summarize each as a tuple $(N, D, M, K, n_{\text{list}}, n_{\text{probe}}, k)$, where N is the padded index capacity after rebalancing/padding. Here, basic is a typical mid-range configuration; low-acc is a deliberately small configuration that favors the circuit-only baseline in gate count; and large is the largest configuration we can run for the circuit-only baseline in our environment (not a limit of the multiset-based design). These configurations are proof-cost calibration and stress-test rather than claims about representative application-scale deployments. For application context, Experiment 1 instead includes the MS MARCO passage-retrieval setup as an application-facing outsourced retrieval workload.

- *Basic:* (8192, 128, 8, 16, 256, 16, 64);
- *Low-acc:* (8192, 128, 8, 1, 16, 1, 1);
- *Large:* (65536, 256, 16, 256, 512, 64, 128).

Methodology. We instantiate both designs with identical IVF-PQ parameters and include the same snapshot-binding subcircuit. For each configuration, we generate and verify proofs 5 times with randomness and report the mean and 95% confidence interval.

Metrics. We report end-to-end prove time, verify time, proof size, peak memory (RSS), and circuit size (the number of arithmetic gates G and the padded domain size $G_B := 2^{\lceil \log_2 G \rceil}$, which largely governs proving time due to FFT-based polynomial operations) [3, 9].

Results. Table 8 shows that in the **basic** and **large** settings, the multiset-based design outperforms the circuit-only baseline under snapshot binding. In particular, it reduces proving time by about 3.6× (basic) and about 22× (large), while also lowering peak memory and keeping verification in the single-digit millisecond range. In contrast, in the non-practical **low-acc** setting ($K = 1, k = 1, n_{\text{probe}} = 1$), the circuit-only baseline is slightly faster.

Table 8: End-to-end proof cost with snapshot binding. Non-gate metrics are mean ± 95% confidence interval (5 runs), rounded to two decimals; G_B appears in parentheses.

System	Prove [s]	Verify [ms]	Proof Size [kB]	Memory [GiB]	$G\,(G_B)$
basic					
circuit-only	2.85±0.11	7.40±0.00	156.24±1.68	12.53±0.42	101 016(2^{17})
multiset-based	0.79±0.04	7.60±1.70	144.21±0.87	11.09±0.05	21 198(2^{15})
low-acc					
circuit-only	0.27±0.01	5.70±0.10	124.55±1.80	12.28±0.00	4 036(2^{12})
multiset-based	0.29±0.01	5.80±0.10	127.99±1.11	12.27±0.00	5 929(2^{13})
large					
circuit-only	527.01±51.02	12.90±0.00	283.67±2.17	747.33±7.47	14 338 070(2^{24})
multiset-based	24.16±0.93	10.80±0.00	248.99±1.18	447.23±6.92	915 627(2^{20})

Table 9: Fixed-scan-budget sweep over n_{list} for the multiset-based design. We report proving time T_{prove}, end-to-end gate count G, and padded bin G_B.

	SIFT1M		GIST1M	
n_{list}	T_{prove} [s]	$G\,(G_B)$	T_{prove} [s]	$G\,(G_B)$
128	6.7427 ± 0.7124	182 225 (2^{18})	18.1420 ± 7.6997	270 229 (2^{19})
256	5.7520 ± 0.1042	192 846 (2^{18})	12.2058 ± 0.1851	342 232 (2^{19})
512	6.0699 ± 0.0909	213 475 (2^{18})	13.4249 ± 0.4753	485 623 (2^{19})
1024	6.4267 ± 0.1864	254 433 (2^{18})	28.3921 ± 4.5745	772 104 (2^{20})
2048	12.8939 ± 0.2739	336 214 (2^{19})	61.9585 ± 11.4705	1 344 931 (2^{21})
4096	14.1515 ± 0.2884	499 721 (2^{19})	124.4228 ± 25.0629	2 490 531 (2^{22})
8192	29.2398 ± 2.0290	826 743 (2^{20})	263.8778 ± 45.0721	4 781 738 (2^{23})

Analysis. This regime split reflects selective rather than uniform optimization. In the low-acc setting, ordering and lookup are already nearly trivial, so the multiset relations become overhead; in the basic and large settings, these operations dominate, so replacing in-circuit sorting and per-candidate lookup with set relations sharply reduces gate count, with the savings further amplified when the circuit falls into a smaller padded bin G_B. We observe that the proof size and verification time change less because the main savings are on the prover side. In the remaining experiments, we therefore treat the multiset-based design as the default instantiation of V3DB.

Answer to RQ-(ii). *For practical IVF-PQ configurations, the multiset-based design makes audit proofs practical under snapshot binding, reducing proving time by up to ∼ 22× while keeping verification in milliseconds.*

6.3 Configuration Trade-offs

Experiment 3 answers Question (iii) by empirically validating the scaling trends predicted by our cost analysis under fixed scan/code budgets, and by applying our tuning procedure to obtain a ZK-efficient deployment configuration used in Experiment 1.

We study SIFT1M ($D = 128$) and GIST1M ($D = 960$) to cover a wide range of embedding dimensions. For MS MARCO ($D = 384$, $N = 2^{24}$), we run only the configuration-selection procedure to obtain the zk-opt configuration used in Experiment 1 and omit the full sweeps for space.

Fixed scan budget varying n_{list}. We first study the effect of increasing n_{list} at fixed scan budget by holding N and $r := n_{\text{probe}}/n_{\text{list}}$ constant; our analysis predicts near-linear growth in G, plus a mild $\Theta(n_{\text{probe}} \log n_{\text{list}})$ snapshot-binding overhead. We fix $B = 64$ with $(M, K) = (8, 256)$ and $r = 1/128$, set $N = 2^{21}$ for SIFT1M/GIST1M, and sweep $n_{\text{list}} \in \{2^7, \ldots, 2^{13}\}$ with $n_{\text{probe}} = n_{\text{list}}/128$, so N_{sel} remains fixed. For each point, Table 9 reports T_{prove}, G, and G_B, letting us check both the linear gate-count trend and the padded-bin effect on proving time.

Measured G is near-linear in n_{list} (Pearson $r \geq 0.9999995$ on both datasets), whereas proving time follows the padded bin more closely than raw G: a fit to $T_{\text{prove}} \approx \alpha G_B \log_2 G_B + \beta$ gives $r \approx 0.9998$.

Fixed code budget varying K. We next vary the PQ codebook size K under fixed code and scan budgets. Our analysis predicts a unimodal trade-off: larger K raises ADC-table construction cost, while smaller $M = B/\log_2 K$ lowers candidate-scoring cost; in the common regime $K \leq n$, this yields a single minimizing $K^\star$. Because proving time tracks G_B, we run a gate-only grid over feasible $K \in \{2, 4, 16, 256\}$ (integral $M = 64/\log_2 K$), sweeping $n_{\text{list}} \in \{128, 256, 512, 1024\}$ on SIFT1M/GIST1M with $N = 2^{21}$ and $n_{\text{probe}} = n_{\text{list}}/128$. We also apply Algorithm 2 to MS MARCO under $N = 2^{24}$ to obtain the zk-opt configuration used in Experiment 1; Table 10 reports G and G_B.

Table 10 shows the predicted discrete unimodal pattern in K. For fixed (D, n_{list}), the best choice is determined by the smallest padded bin rather than raw G alone; when several K values share that bin, Algorithm 2 selects the larger K, yielding the zk-opt configurations used in Experiment 1. Table 11 reports the corresponding gate counts and estimated proof times of the two configurations in Experiment 1 under the multiset-based design.

Table 10: Gate-count grid over K at fixed code budget $B = 64$; parentheses show G_B, and bold marks the selected zk-opt configuration for each dataset.

	K			
n_{list}	2	4	16	256
SIFT1M *($D = 128$)*				
128	929 950(2^{20})	497 482(2^{19})	281 654(2^{19})	182 225(2^{18})
256	935 409(2^{20})	502 978(2^{19})	287 398(2^{19})	192 846(2^{18})
512	945 715(2^{20})	513 359(2^{19})	298 272(2^{19})	213 475(2^{18})
1024	966 029(2^{20})	533 818(2^{20})	319 722(2^{19})	**254 433**(2^{18})
GIST1M *($D = 960$)*				
128	959 840(2^{20})	527 830(2^{20})	314 746(2^{19})	270 229(2^{19})
256	994 980(2^{20})	563 257(2^{20})	351 919(2^{19})	342 232(2^{19})
512	1 064 650(2^{21})	633 499(2^{20})	425 652(2^{19})	**485 623**(2^{19})
1024	1 203 690(2^{21})	773 683(2^{20})	572 816(2^{20})	772 104(2^{20})

Table 11: Estimated proof-side cost of the Experiment 1 configurations: entries show G ($\hat{T}_{\text{prove}}$), with $\hat{T}_{\text{prove}} = 1.364 \times 10^{-6} G_B \log_2 G_B + 0.261$.

Config	SIFT1M	GIST1M	MS MARCO
high-acc	826 743 (28.87)	4 781 738 (263.43)	3 224 862 (126.12)
zk-opt	254 433 (6.70)	485 623 (13.85)	1 827 516 (60.33)

Analysis. The two sweeps show that the cost model is operational rather than merely descriptive. Under a fixed scan budget, tuning should prioritize minimizing the padded bin G_B; under a fixed code budget, the rule is to stay in the smallest attainable bin and then keep the larger utility-favorable K within that bin. This is the rationale behind our zk-opt selection rule.

Answer to RQ-(iii). *Under a fixed scan budget, measured circuit size scales nearly linearly with n_{list}, and proving time exhibits the expected power-of-two step behavior driven by the padded bin G_B. Under a fixed code budget, varying K exhibits the predicted unimodal trade-off. These trends validate our cost model and motivate a cost-driven tuning procedure for selecting zk-efficient configurations.*

7 CONCLUSION

We presented V3DB, a verifiable, versioned vector database for audit-on-demand IVF-PQ top-k retrieval. Upon challenge, it produces a succinct ZK proof that returned payloads match a standardized fixed-shape semantics on a committed snapshot while hiding embeddings and private index state. V3DB combines ZK-friendly index shaping, snapshot commitments, and multiset-based proving to avoid in-circuit sorting and random access; our Plonky2 prototype achieves up to ∼ 22× faster proving and ∼ 40% lower peak memory than a circuit-only baseline, with millisecond verification, and preserved retrieval utility.

REFERENCES

[1] Tim Althoff, Adrian Ulges, and Andreas Dengel. 2011. Balanced Clustering for Content-based Image Browsing.. In *Informatiktage*. 27–30.

[2] David Arthur and Sergei Vassilvitskii. 2006. *k-means++: The advantages of careful seeding*. Technical Report. Stanford.

[3] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2018. Fast reed-solomon interactive oracle proofs of proximity. In *45th international colloquium on automata, languages, and programming (icalp 2018)*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 14–1.

[4] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. 2014. Succinct {Non-Interactive} zero knowledge for a von neumann architecture. In *23rd USENIX Security Symposium (USENIX Security 14)*. 781–796.

[5] Manuel Blum, Paul Feldman, and Silvio Micali. 2019. Non-interactive zero-knowledge and its applications. In *Providing sound foundations for cryptography: on the work of Shafi Goldwasser and Silvio Micali*. 329–349.

[6] Paul S Bradley, Kristin P Bennett, and Ayhan Demiriz. 2000. Constrained k-means clustering. *Microsoft Research, Redmond* 20, 0 (2000), 0.

[7] 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.

[8] Clarivate. 2024. Clarivate Launches AI-Powered Patent Search Solution in Derwent. https://clarivate.com/news/clarivate-launches-ai-powered-patent-search-solution-in-derwent/ Accessed: 2026-02-06.

[9] James W Cooley and John W Tukey. 1965. An algorithm for the machine calculation of complex Fourier series. *Mathematics of computation* 19, 90 (1965), 297–301.

[10] Victor Costan and Srinivas Devadas. 2016. Intel SGX explained. *Cryptology ePrint Archive* (2016).

[11] Victor Costan, Ilia Lebedev, and Srinivas Devadas. 2016. Sanctum: Minimal hardware extensions for strong software isolation. In *25th USENIX Security Symposium (USENIX Security 16)*. 857–874.

[12] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, Jimmy Lin, Ellen M Voorhees, and Ian Soboroff. 2025. Overview of the TREC 2022 deep learning track. *arXiv preprint arXiv:2507.10865* (2025).

[13] Rieke de Maeyer, Sami Sieranoja, and Pasi Fränti. 2023. Balanced k-means revisited. (2023).

[14] Zeheng Fan, Yuxiang Zeng, Zhuanglin Zheng, and Yongxin Tong. 2025. FedVSE: A Privacy-Preserving and Efficient Vector Search Engine for Federated Databases. *Proceedings of the VLDB Endowment* 18, 12 (2025), 5371–5374.

[15] Zeheng Fan, Yuxiang Zeng, Zhuanglin Zheng, Binhan Yang, and Yongxin Tong. 2025. FedVS: Towards Federated Vector Similarity Search with Filters. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*. 579–590.

[16] 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.

[17] Ariel Gabizon and Zachary J Williamson. 2020. plookup: A simplified polynomial protocol for lookup tables. *Cryptology ePrint Archive* (2020).

[18] Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. 2019. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive* (2019).

[19] Shafi Goldwasser, Silvio Micali, and Chales Rackoff. 2019. The knowledge complexity of interactive proof-systems. In *Providing sound foundations for cryptography: On the work of shafi goldwasser and silvio micali*. 203–225.

[20] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. 2021. Poseidon: A new hash function for {Zero-Knowledge} proof systems. In *30th USENIX Security Symposium (USENIX Security 21)*. 519–535.

[21] Jens Groth. 2016. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 305–326.

[22] Binbin Gu, Juncheng Fang, and Faisal Nawab. 2025. PoneglyphDB: Efficient Non-interactive Zero-Knowledge Proofs for Arbitrary SQL-Query Verification. *Proceedings of the ACM on Management of Data* 3, 1 (2025), 1–27.

[23] 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.

[24] Yushuai Ji, Shengkun Zhu, Shixun Huang, Zepeng Liu, Sheng Wang, and Zhiyong Peng. 2025. Federated and Balanced Clustering for High-dimensional Data. *Proceedings of the VLDB Endowment* 18, 11 (2025), 4032–4044.

[25] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.

[26] Dow Jones. 2026. Factiva News Search APIs. https://developer.dowjones.com/documents/site-docs-factiva_apis-factiva_workflow_apis_rest-factiva_news_search Accessed: 2026-02-06.

[27] 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.

[28] LexisNexis. 2026. Lexis API. https://www.lexisnexis.com/en-us/products/lexis-api.page Accessed: 2026-02-06.

[29] Xiling Li, Chenkai Weng, Yongxin Xu, Xiao Wang, and Jennie Rogers. 2023. Zksql: Verifiable and efficient query evaluation with zero-knowledge proofs. *Proceedings of the VLDB Endowment* 16, 8 (2023).

[30] Hanyang Liu, Junwei Han, Feiping Nie, and Xuelong Li. 2017. Balanced clustering with least square regression. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 31.

[31] Hongfu Liu, Ziming Huang, Qi Chen, Mingqin Li, Yun Fu, and Lintao Zhang. 2018. Fast clustering with flexible balance constraints. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 743–750.

[32] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137.

[33] Mikko I Malinen and Pasi Fränti. 2014. Balanced k-means for clustering. In *Joint IAPR international workshops on statistical techniques in pattern recognition (SPR) and structural and syntactic pattern recognition (SSPR)*. Springer, 32–41.

[34] 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.

[35] Ralph C Merkle. 1987. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*. Springer, 369–378.

[36] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. Ms marco: A human-generated machine reading comprehension dataset. (2016).

[37] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. 2016. Pinocchio: Nearly practical verifiable computation. *Commun. ACM* 59, 2 (2016), 103–112.

[38] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789* (2023).

[39] Questel. 2026. Sophia Search: AI-Powered Semantic Search. https://intelligence.help.questel.com/en/support/solutions/articles/77000587553-sophia-search-ai-powered-semantic-search Accessed: 2026-02-06.

[40] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084* (2019).

[41] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. 2015. Trusted execution environment: What it is, and what it is not. In *2015 IEEE Trustcom/BigDataSE/Ispa*, Vol. 1. IEEE, 57–64.

[42] Chenzhao Wang, Jilian Zhang, Xuyang Liu, Kaimin Wei, and Bingwen Feng. 2024. Verifiable Graph-Based Approximate Nearest Neighbor Search. In *International Conference on Advanced Data Mining and Applications*. Springer, 3–17.

[43] Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in neural information processing systems* 33 (2020), 5776–5788.

[44] Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. 2023. Repocoder: Repository-level code completion through iterative retrieval and generation. *arXiv preprint arXiv:2303.12570* (2023).

[45] Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2024. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 19724–19731.