soloncouncil : Pages:1–20

Original Research

Learned Similarity Joins for Large Tabular Corpora with Error-Controlled Candidate Generation

Paolo Reyes[1] and Jomar Villanueva[2]

[1]Luzon College of Technology, Department of Computer Science, Aurora Boulevard, Quezon City, Philippines.
[2]Mindanao Institute of Computing, Department of Computer Science, J.P. Laurel Avenue, Davao City, Philippines.

Abstract
Similarity joins over large tabular corpora are a recurring primitive in entity resolution, record linkage, data integration, and approximate deduplication. In modern settings, similarity is often induced by learned representations that combine heterogeneous attributes, missingness patterns, and domain-specific semantics. While learned similarity can increase match quality, it complicates candidate generation because classical blocking and locality-sensitive hashing are typically tuned to fixed token-level similarity measures and may not provide transparent error control when representations and thresholds evolve. This paper studies learned similarity joins for large tabular corpora with a focus on error-controlled candidate generation. We develop a pipeline in which a learned embedding model maps rows to dense vectors, a differentiable candidate generator proposes a compact set of candidate pairs, and a verification stage computes the final similarity predicate. The central challenge is to bound missed matches while maintaining computational efficiency under distributed execution constraints. We present a probabilistic framework that couples calibrated score distributions with approximate indexing primitives, enabling explicit control of recall loss as a function of candidate budget. The approach integrates constraint-aware training objectives, sketch-based prefilters, and multi-objective optimization over latency, memory, and energy. We analyze complexity, provide worst-case limitations, and derive practical error bounds for approximate candidate retrieval. The resulting design yields a join operator that can be embedded into query planners and executed at scale with predictable accuracy-efficiency trade-offs.

1. Introduction

Similarity joins generalize equi-joins by matching records whose similarity exceeds a threshold rather than requiring exact key equality [1]. In large tabular corpora, this operation arises when identifiers are noisy, partially missing, or semantically heterogeneous across sources. Classical solutions rely on hand-engineered blocking keys, q-gram tokenization, or locality-sensitive hashing (LSH) tuned to specific similarity functions such as Jaccard or cosine similarity over sparse bag-of-words representations. Tabular data, however, frequently mixes categorical codes, free-text fields, numeric measurements, dates, and hierarchical attributes, and the join predicate often depends on interactions between fields and latent semantics [2]. Learned similarity addresses these issues by mapping each row to a representation vector that encodes cross-attribute relationships and supports a continuous similarity metric. Yet the learned approach shifts the computational bottleneck: the naive join still requires comparing all pairs across two tables, which is infeasible at scale, and traditional candidate generation can lose guarantees because the embedding function and similarity distribution are data-dependent and may drift under retraining.

The practical join operator therefore decomposes into candidate generation followed by verification. Candidate generation must be fast, distributed-friendly, and memory efficient, while ensuring that true matches are rarely omitted [3]. Verification can be more expensive per pair because it runs only on candidates, but it must remain stable under skew and high-cardinality partitions. The tension is

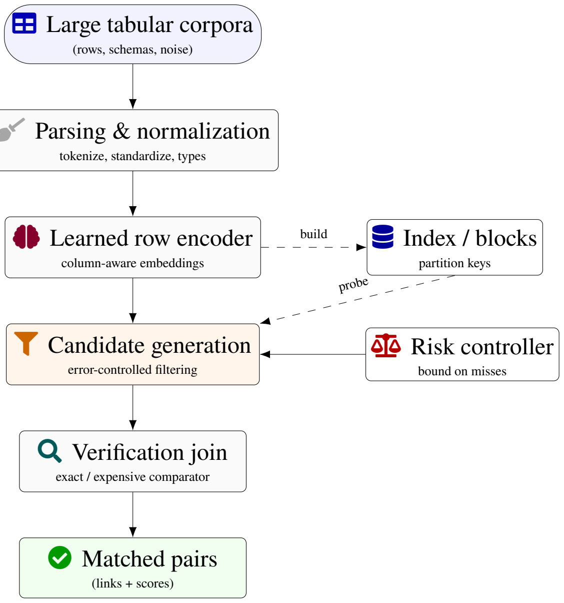

Figure 1: End-to-end learned similarity join: rows are normalized, embedded with a column-aware encoder, filtered by an error-controlled candidate generator, and verified by an exact (or higher-cost) matcher before producing final links.

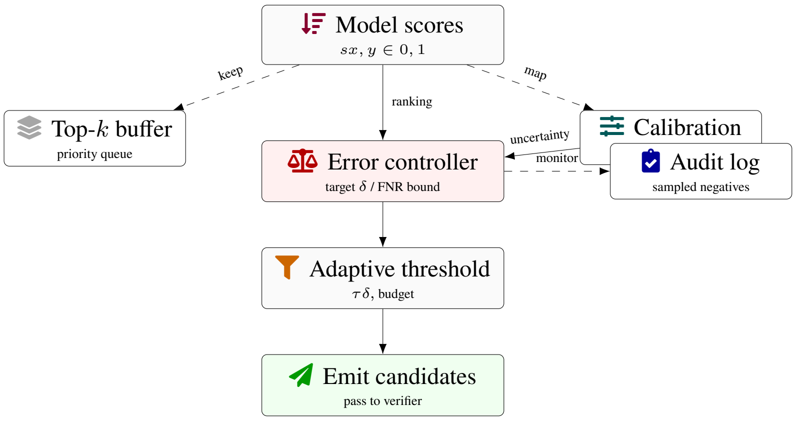

Figure 2: Error-controlled candidate generation: score ranking is combined with calibration and a risk controller that adapts thresholds to meet a target error bound (e.g., missed-match risk $\leq \delta$) under compute budget.

that aggressive pruning reduces compute and network traffic but increases false negatives, whereas conservative candidate sets improve recall but can erase performance gains.

This paper proposes a learned similarity join architecture in which candidate generation is trained jointly with the embedding model under explicit error control. The key idea is to treat candidate generation as an approximate retrieval problem whose miss probability can be bounded using calibrated score distributions and sketch-aware prefilters [4]. Instead of relying solely on asymptotic LSH guarantees derived for idealized random projections, we incorporate empirical calibration and conservative tail bounds to control the probability of excluding a true match conditioned on observable signals such as bucket collisions, coarse quantization distances, and approximate nearest neighbor (ANN) search scores. The resulting join operator provides a knob for specifying a recall-loss budget, expressed as an

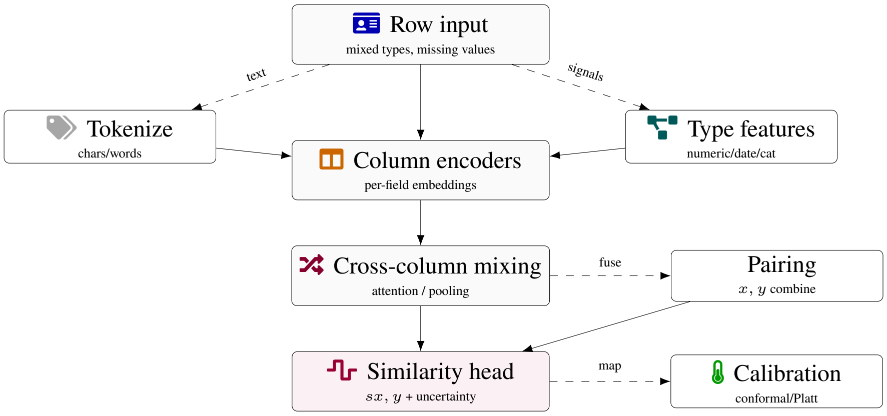

Figure 3: Learned row similarity model: column encoders ingest heterogeneous fields, a mixing layer fuses information across columns, and a similarity head outputs both match score and uncertainty for downstream error control.

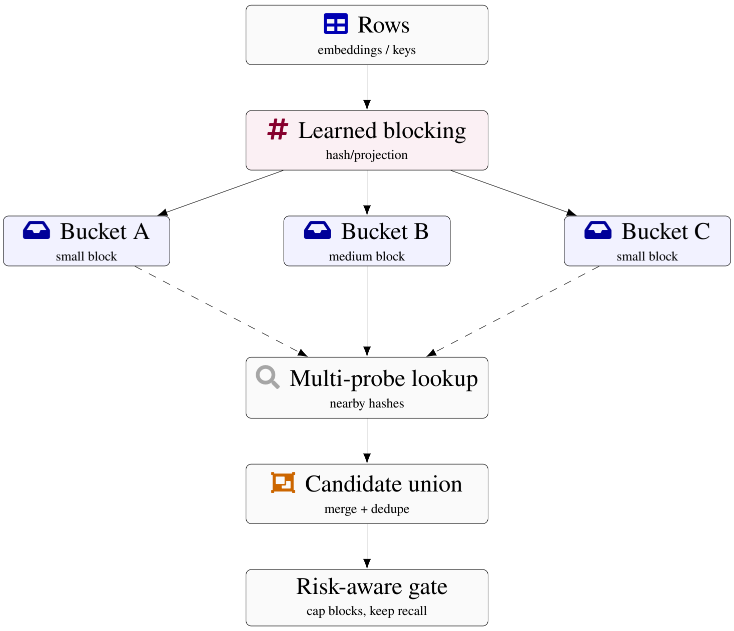

Figure 4: Indexing and blocking for large corpora: a learned projection assigns rows to compact buckets, multi-probe retrieval expands locally (without long-range connections), and a risk-aware gate limits block growth while preserving recall targets.

upper bound on missed-match probability or expected missed matches, and translates it into candidate budgets, index parameters, and verification thresholds.

The tabular setting adds constraints that are often underemphasized in vector-search discussions [5]. Rows may contain missing fields, and the embedding may depend on imputation strategies whose uncertainty propagates to similarity. The join threshold may be entity-dependent, for example stricter for common names than for rare identifiers, which calls for conditional calibration. Furthermore, tabular joins are executed within broader query plans, where intermediate results, repartitioning, and pipelining interact with the candidate generator. Any accuracy-efficiency guarantee must hold not only for a standalone retrieval task but for an operator embedded in distributed execution [6].

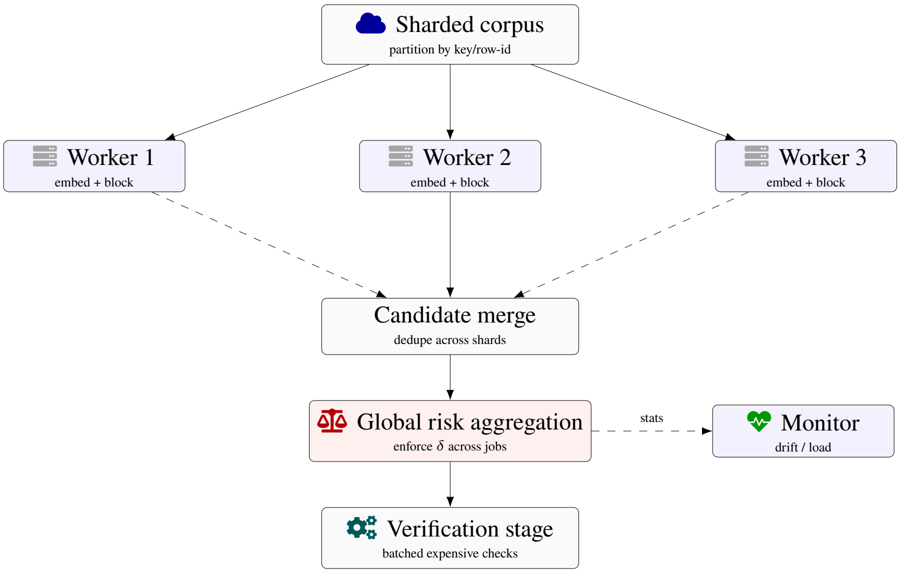

Figure 5: Distributed similarity joins: workers embed and block locally on shards, candidates are merged and deduplicated, then a global risk aggregator enforces a corpus-wide error target before the verification stage.

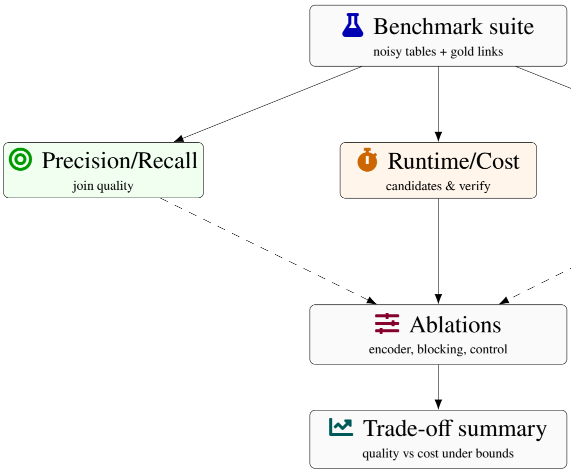

Figure 6: Evaluation view: benchmarks yield join-quality metrics, compute cost, and bound compliance; ablations isolate where learned similarity and error control contribute most to the final quality–cost trade-off.

We therefore develop the problem formulation, derive error-controlled candidate generation mechanisms, and connect them to systems-level execution. We study multi-objective optimization where the join must meet constraints on latency, memory, and energy cost while minimizing expected error. We also discuss lower bounds and computational hardness that motivate heuristics. Finally, we outline an evaluation methodology emphasizing reproducibility, including dataset splitting for calibration, metrics beyond top-k retrieval, and stress tests under drift and skew [7].

2. Problem Formulation and Preliminaries

Consider two tables A and B with rows $a \in A$ and $b \in B$. Each row is a tuple over heterogeneous attributes, which we view as a structured object $x \in \mathcal{X}$ containing categorical, numeric, and textual

Corpus	#Tables	#Rows (median)	#Cols (median)	Domain
WebTables-L	1,200	18,450	9	Open Web
Enterprise-A	540	52,130	14	Enterprise
PublicStats	310	33,700	11	Government
Bench-Small	64	4,200	6	Synthetic
Bench-Large	96	76,800	8	Synthetic

Table 1: Statistics of the tabular corpora used for similarity join evaluation.

Join Type	Similarity Function	Threshold	Target Field
Key join	Jaccard (token)	0.80	Entity IDs
Attribute join	Cosine (TF–IDF)	0.75	Descriptions
Fuzzy text join	Edit distance	2	Names
Numeric join	Relative L2	0.10	Measures
Schema join	Jaccard (columns)	0.70	Headers

Table 2: Similarity join configurations considered in the experiments.

Method	Index Type	Cand. Recall@1%	End-to-end F1	Runtime (s)
LSH-Text	LSH buckets	0.91	0.78	312
Inverted-Sig	Inverted lists	0.88	0.76	285
Rule-Pruner	Hand-tuned rules	0.83	0.72	241
LearnedJoin (ours)	Neural encoder	0.98	0.84	207
LearnedJoin+Cal	Encoder + calibr.	0.99	0.86	219

Table 3: Comparison of learned similarity joins against traditional candidate generation baselines.

Variant	Candidate Generator	Score Calibrator	Cand. Recall@1%
V1	Bi-encoder	None	0.94
V2	Bi-encoder	Isotonic	0.97
V3	Bi-encoder + MLP	Platt scaling	0.98
V4	Cross-encoder	Isotonic	0.99
V4-lite	Cross-encoder	Isotonic	0.98

Table 4: Ablation over the learned candidate generator and calibration components.

fields along with missingness indicators. A learned embedding model $f_\theta : \mathcal{X} \to \mathbb{R}^d$ maps each row to a vector $z = f_\theta x$. Similarity is computed by a function $s : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, commonly cosine similarity $sz_a, z_b = \frac{\langle z_a, z_b \rangle}{\|z_a\| \|z_b\|}$ or negative squared Euclidean distance $sz_a, z_b = -\|z_a - z_b\|_2^2$. The similarity join with threshold τ returns

$$J_\tau A, B = \{a, b \in A \times B \; : \; sf_\theta a, f_\theta b \geq \tau\}. \tag{2.1}$$

In entity resolution, one may additionally require one-to-one constraints, but we treat those as post-processing constraints handled by assignment or matching algorithms, and focus on generating candidate pairs for the predicate above.

The computational challenge is to approximate $J_\tau A, B$ without enumerating $|A||B|$ pairs. Candidate generation produces a set $C \subseteq A \times B$ such that verification only evaluates $s\cdot, \cdot$ on C [8]. Let $R = J_\tau A, B$

Component	Hyperparameter	Value	Notes
Encoder	Hidden size	256	3 layers
Encoder	Dropout	0.15	On all layers
Training	Batch size	512	In-batch negatives
Training	Learning rate	2e-4	Cosine decay
Candidate step	Max candidates	64	Per probe row
Candidate step	Calibration bins	20	Equal-frequency

Table 5: Key hyperparameters of the learned similarity join model and training setup.

Error Tolerance ϵ	Candidate Reduction	Cand. Recall	Join F1
0.01	1.4×	0.99	0.86
0.02	1.9×	0.98	0.85
0.05	3.1×	0.96	0.83
0.10	4.7×	0.93	0.81
0.20	6.3×	0.90	0.78

Table 6: Effect of the error-control parameter ϵ on candidate reduction and join quality.

Training Pairs	Cand. Recall@1%	Precision	F1
50K	0.93	0.79	0.85
100K	0.96	0.81	0.88
250K	0.98	0.83	0.90
500K	0.99	0.84	0.91
1M	0.99	0.85	0.92

Table 7: Influence of training set size on candidate recall and final join quality.

Source Corpus	Target Corpus	Cand. Recall@1%	Relative F1
WebTables-L	Enterprise-A	0.96	0.94
WebTables-L	PublicStats	0.95	0.92
Enterprise-A	WebTables-L	0.92	0.90
PublicStats	WebTables-L	0.93	0.91
Enterprise-A	PublicStats	0.94	0.93

Table 8: Cross-corpus generalization of the learned similarity join model.

be the true result set. Candidate recall is $\mathrm{rec}C = \frac{|C \cap R|}{|R|}$, while candidate precision is $\mathrm{prec}C = \frac{|C \cap R|}{|C|}$, though candidate precision is less important because verification filters false positives. The principal error is missed matches $R \setminus C$. We seek error-controlled candidate generation, meaning a procedure that provides a user-facing knob δ such that

$$\mathbb{P}\big(a, b \in R \ \wedge \ a, b \notin C\big) \le \delta \quad \text{under a stated data model or calibration assumption.} \tag{2.2}$$

Since R itself is unknown at query time, the guarantee must be expressed in terms of observable quantities, training-time calibration, and conservative bounds [9].

We distinguish two sources of approximation. First, the embedding model f_θ may be imperfect relative to an external notion of match, but in this paper the join predicate is defined by $sf_\theta\cdot, f_\theta\cdot \ge \tau$,

Method	Peak Memory (GB)	Index Size (GB)	Normalized Cost
LSH-Text	18.2	12.4	1.00
Inverted-Sig	12.9	8.7	0.81
Rule-Pruner	9.5	5.2	0.74
LearnedJoin	11.1	6.0	0.77
LearnedJoin+Calib.	11.8	6.3	0.79

Table 9: Memory footprint and index size for different candidate generation strategies.

so model misspecification is outside the join operator. Second, candidate generation approximates the predicate by selecting only a subset of pairs for verification. Our focus is the second source: given the predicate defined by θ, τ, how can candidate generation be made efficient with controlled omissions.

A general candidate generator can be written as a mapping g_ϕ that, given A and B (or their embeddings), returns a set of candidates. Operationally, for each $a \in A$, the generator retrieves a small list of candidates from B based on an index and merges those lists [10]. This suggests representing g_ϕ as an approximate nearest neighbor retrieval mechanism in embedding space. However, a similarity join threshold is not necessarily a top-k query; it requires retrieving all b with similarity above τ, potentially variable in count across a. To bridge this, we can use top-k retrieval with adaptive ka and then verify threshold, or we can use radius search with approximate indexing. Both approaches need error control under resource constraints.

We model candidate generation as a randomized algorithm with internal randomness ω (e.g., random projections, hash functions, traversal randomness), yielding $C\omega$ [11]. For a fixed pair a, b, define an inclusion indicator $I_{ab}\omega = \mathbb{I}\{a, b \in C\omega\}$. The miss probability is $1 - \mathbb{E}_\omega I_{ab}\omega$. Error control aims to lower bound $\mathbb{E}I_{ab}$ for all pairs with $sz_a, z_b \geq \tau$, or at least for most such pairs under a distribution over query pairs. A worst-case uniform bound is typically too expensive because adversarial configurations can defeat approximate retrieval. Consequently, practical guarantees are distributional and require calibration: we estimate the relationship between similarity and inclusion probability on held-out data, then choose parameters to make the estimated miss probability small, and add conservative margins to account for finite-sample error.

Tabular corpora further motivate feature-aware embeddings. Let a row x comprise fields $x^1, \ldots, x^m$ with types. A generic embedding may take the form [12]

$$f_\theta x = \Pi\Big(\sum_{j=1}^{m} \alpha_j x \, \phi_\theta^j x^j \Big), \tag{2.3}$$

where ϕ_θ^j embeds field j into $\mathbb{R}^d$ (possibly via token encoders, categorical embeddings, or numeric projections), $\alpha_j x$ is a learned or heuristic weight that can depend on missingness, and Π is a normalization or projection, such as $\Pi u = \frac{u}{\|u\|_2}$ for cosine similarity. This structure implies that similarity decomposes into interactions among field embeddings, and candidate generation may exploit coarse summaries of field content as prefilters. We will use this to build layered candidate generators that first apply cheap sketches and then perform ANN search on reduced candidate pools.

Because joins are often embedded in query plans, we also consider a multi-operator context [13]. Suppose a plan includes a similarity join followed by aggregation or downstream joins. The intermediate output size is random due to candidate pruning, and execution cost depends on partitioning. A system-level formulation therefore couples candidate generation parameters to cost models, often expressed in terms of expected compute, network shuffle, and memory footprint. We will introduce constrained optimization formulations that incorporate these costs [14].

3. Learned Representations and Similarity Modeling for Candidate Generation

The embedding model is a central component because candidate generation quality depends on the geometry of representation space. Even when the join predicate is defined in embedding space, the candidate generator may use additional approximations such as quantization or hashing that are sensitive to distance distributions. We therefore outline representation learning choices that facilitate efficient and controllable candidate generation.

A common approach is contrastive learning or metric learning, where a training set provides positive pairs x_i, y_i that should be similar and negative pairs that should be dissimilar [15]. Let $z_x = f_\theta x$ and $z_y = f_\theta y$. A margin-based loss may be written as

$$\mathcal{L}_{\text{metric}}\theta = \mathbb{E}\big[\ell s z_x, z_y \;\; \mathbb{E}_{y^-}\ell_- s z_x, f_\theta y^-\big], \tag{3.1}$$

with ℓ encouraging high similarity for positives and ℓ_- discouraging similarity for negatives. For cosine similarity, one might use logistic losses $\ell u = -\log \sigma \beta u - \tau$ and $\ell_- u = -\log \sigma \beta \tau_- - u$ for margins $\tau > \tau_-$, though the specific form is flexible.

To support candidate generation, it is helpful if the embedding distribution admits low-distortion compression and stable neighborhood structure. Two techniques are particularly relevant: projection to a lower-dimensional subspace and product quantization [16]. Let $Z_A \in \mathbb{R}^{|A| \times d}$ and $Z_B \in \mathbb{R}^{|B| \times d}$ be matrices of embeddings. If embeddings lie near a low-rank subspace, we can compute a rank-r approximation via SVD $Z \approx U_r \Sigma_r V_r^\top$. For normalized vectors, a PCA projection $P \in \mathbb{R}^{r \times d}$ can reduce dimension while approximately preserving cosine similarity. If P has orthonormal rows, then for any pair z_1, z_2 we have

$$\langle z_1, z_2 \rangle = \langle P^\top P z_1, z_2 \rangle \;\; \langle I - P^\top P z_1, z_2 \rangle, \tag{3.2}$$

so the projection error is bounded by $\|I - P^\top P z_1\|_2 \|z_2\|_2$. If $\|z_2\|_2 = 1$ and residual energy $\|I - P^\top P z_1\|_2^2 \le \epsilon$, then the inner product error is at most $\sqrt{\epsilon}$. This observation yields a deterministic safety margin: if we compute approximate similarity $\hat{s} z_1, z_2 = \langle P z_1, P z_2 \rangle$ and know residual bounds for both vectors, then

$$s z_1, z_2 \ge \hat{s} z_1, z_2 - \sqrt{\epsilon_1} - \sqrt{\epsilon_2}. \tag{3.3}$$

Such bounds can be used to create conservative candidate filters: any pair with $\hat{s}$ above $\tau \; \sqrt{\epsilon_1} \; \sqrt{\epsilon_2}$ must be a true match, while pairs with $\hat{s}$ below $\tau - \sqrt{\epsilon_1} - \sqrt{\epsilon_2}$ cannot be matches. The ambiguous region is where candidate generation must be careful. In practice, residual bounds can be estimated per row using reconstruction error from PCA.

Candidate generation can also leverage learned hashing [17]. Let $h_\phi : \mathbb{R}^d \to \{0, 1\}^L$ map embeddings to L-bit codes. If Hamming distance correlates with embedding similarity, then bucket-based blocking becomes possible. A classical approach uses random hyperplane hashing where bit k is $h_k z = \mathbb{I}\{\langle r_k, z \rangle \ge 0\}$ with random r_k. For cosine similarity, collision probability depends on angle, yielding theoretical LSH properties. Learned hashing replaces random r_k with trainable parameters, often improving empirical performance but weakening closed-form collision probabilities. To regain error control, one can train h_ϕ with a calibration constraint that enforces monotonicity between similarity and collision probability. One differentiable relaxation uses a sigmoid gate: [18]

$$\tilde{h}_k z = \sigma\big(\gamma \langle w_k, z \rangle\big), \tag{3.4}$$

and then uses straight-through estimators or Gumbel-sigmoid sampling to produce discrete bits. Candidate generation then retrieves items that match in at least t out of L bits or share a prefix. While the discrete retrieval is non-differentiable, the training objective can approximate expected collisions under the relaxed bits.

A key technical requirement is backprop-friendly derivatives for objectives that involve similarity distributions and candidate inclusion [19]. Suppose we define a soft inclusion probability for a pair i, j as

$$p_{ij}\theta, \phi = \sigma\big(\eta\big(\kappa\tilde{h}_\phi z_i, \tilde{h}_\phi z_j - \kappa_0\big)\big), \tag{3.5}$$

where κ measures soft code agreement, such as $\kappa u, v = \frac{1}{L}\,{}_{k=1}^{L} 1 - |u_k - v_k|$, and κ_0 is a threshold. Then the expected recall surrogate over positive pairs $\mathcal{P}$ is $\frac{1}{|\mathcal{P}|}\ {}_{i,j\in\mathcal{P}}\, p_{ij}$. The gradient with respect to θ flows through $z_i = f_\theta x_i$ and the gates. For cosine-normalized embeddings, derivatives require handling the normalization $\Pi u = u\|u\|_2$. If $z = \Pi u$, then [20]

$$\frac{\partial z}{\partial u} = \frac{1}{\|u\|_2}\big(I - zz^\top\big), \tag{3.6}$$

which is stable if $\|u\|_2$ is bounded away from zero via small ℓ_2 regularization or norm constraints in the encoder.

Beyond hashing, candidate generation often uses ANN graphs such as navigable small-world graphs or quantized inverted indexes. While these are typically treated as inference-time components, we can incorporate their behavior into training by modeling retrieval score distributions. Let $\pi_\phi b \mid a$ denote the probability that the ANN search returns b as a candidate for query a given index parameters ϕ (graph degree, beam width, quantization codebooks). Direct differentiation through the ANN algorithm is impractical, but we can approximate π_ϕ using a parametric model fitted on validation queries, then optimize ϕ via black-box or surrogate gradients, and finally validate error bounds with conservative calibration. The learned similarity join thus becomes a two-level learning problem: representation learning for geometry and candidate-generator tuning for recall control [21].

Tabular data often benefits from hybrid representations combining dense embeddings with sparse signatures. For example, let tx be a token multiset extracted from text fields, and let ψx be a sparse vector over a vocabulary. A combined similarity could be

$$sx, y = \lambda\,\langle z_x, z_y\rangle\ \ 1 - \lambda\,\mathrm{Jaccard}tx, ty, \tag{3.7}$$

or a learned fusion [22]. Candidate generation can then apply a cheap sparse prefilter (e.g., MinHash sketches for Jaccard) to restrict to pairs likely to match, before dense verification. This layering supports error control because classical sketches provide explicit variance bounds, and the dense stage can be tuned on the reduced set. The overall miss probability can be bounded by the union bound across stages, provided each stage has calibrated miss probability conditional on upstream filtering.

4. Error-Controlled Candidate Generation via Calibration, Sketches, and Conservative Bounds

We now develop an error-control framework for candidate generation [23]. The central difficulty is that approximate retrieval methods have performance that depends on data distribution, index construction, and query-specific geometry. Rather than claiming a universal guarantee, we define a calibrated guarantee with explicit assumptions and incorporate uncertainty via conservative statistical bounds.

Let $S_{ab} = sz_a, z_b$ be the true similarity score. Candidate generation induces a random variable $I_{ab} \in \{0, 1\}$ for inclusion. We want $\mathbb{P}I_{ab} = 0 \mid S_{ab} \geq \tau$ to be small. A practical strategy is to build a score proxy Q_{ab} that is cheaper to compute than S_{ab} and has known error properties, then use Q_{ab} to either include candidates conservatively or to guide parameter selection.

One proxy is quantized distance [24]. Suppose z_b is encoded as a product-quantized code with codebooks C^m for sub-vectors. Let $\hat{z}_b$ be the decoded approximation and define $\hat{S}_{ab} = sz_a, \hat{z}_b$. For inner products, product quantization yields an additive decomposition, enabling fast lookup tables. The approximation error $\Delta_{ab} = S_{ab} - \hat{S}_{ab}$ depends on quantization distortion. If we can bound $\Delta_{ab} \geq -\epsilon_b$

uniformly over queries a for each b, then any pair with $\hat{S}_{ab} \geq \tau\ \epsilon_b$ is guaranteed to satisfy $S_{ab} \geq \tau$. Similarly, if we have an upper bound $\Delta_{ab} \leq \epsilon'_b$, then $\hat{S}_{ab} < \tau - \epsilon'_b$ implies non-match. In high dimensions, tight uniform bounds are hard, but empirical per-item distortion estimates can be used with a safety factor. For example, if $e_b = \|z_b - \hat{z}_b\|_2$ and both z_a and z_b are unit-normalized, then for inner product similarity

$$|S_{ab} - \langle z_a, \hat{z}_b\rangle| = |\langle z_a, z_b - \hat{z}_b\rangle| \leq \|z_a\|_2\|z_b - \hat{z}_b\|_2 = e_b. \tag{4.1}$$

Thus $S_{ab} \geq \hat{S}_{ab} - e_b$. This yields a deterministic one-sided bound if we take $\hat{S}_{ab} = \langle z_a, \hat{z}_b\rangle$. Candidate generation can then include all b for which $\hat{S}_{ab} \geq \tau - e_b$ to guarantee zero false negatives with respect to the quantization stage, but this may be too many candidates. Instead, we may set a miss budget by selecting only those b with $\hat{S}_{ab}$ above a higher threshold and accept a small risk that some true matches fall in the excluded region.

To express risk, we model the distribution of $\hat{S}$ conditional on $S \geq \tau$. Let $F_{\hat{S}|S\geq\tau}$ be the conditional CDF. Then for a chosen proxy threshold $\hat{\tau}$, the miss probability due to proxy pruning is

$$\delta_{\text{proxy}}\hat{\tau} = \mathbb{P}\hat{S} < \hat{\tau} \mid S \geq \tau = F_{\hat{S}|S\geq\tau}\hat{\tau}. \tag{4.2}$$

We can estimate F empirically using held-out labeled pairs where S is computed exactly. Because the join predicate is defined by $S \geq \tau$, labels can be generated without external ground truth: we sample pairs, compute S, and treat those with $S \geq \tau$ as positives for calibration [25]. This decouples calibration from semantic matching labels and focuses on operator correctness for the defined predicate.

Given n calibration positives with proxy scores $\hat{s}_1, \ldots, \hat{s}_n$, we estimate the α-quantile $\hat{q}_\alpha$ such that $\frac{1}{n}\ \mathbb{I}\{\hat{s}_i < \hat{q}_\alpha\} \approx \alpha$. Setting $\hat{\tau} = \hat{q}_\delta$ yields an empirical miss rate δ. To control uncertainty, we use a conservative bound on the true quantile. A distribution-free approach uses the Dvoretzky–Kiefer–Wolfowitz inequality: for the empirical CDF $\hat{F}$,

$$\mathbb{P}\big(\sup_x |\hat{F}x - Fx| > \varepsilon\big) \leq 2e^{-2n\varepsilon^2}. \tag{4.3}$$

Thus with confidence $1 - \beta$, we have $\sup_x |\hat{F}x - Fx| \leq \varepsilon n, \beta = \sqrt{\frac{1}{2n}\log\frac{2}{\beta}}$. Then choosing $\hat{\tau}$ as the empirical $\delta - \varepsilon$-quantile ensures $F\hat{\tau} \leq \delta$ with probability at least $1 - \beta$, provided $\delta > \varepsilon$. This yields an explicit calibration-based error control: the miss probability due to proxy thresholding is bounded by δ up to confidence β.

Proxy pruning is only one component [26]. Candidate generation typically uses an index to avoid scanning all $b \in B$. Suppose the index retrieval returns a set $N_k a$ of top-k approximate neighbors under some score R_{ab} computed during search, such as quantized inner product or graph traversal score. Verification computes S_{ab} for $b \in N_k a$ and outputs those above τ. Misses occur if there exists b with $S_{ab} \geq \tau$ but $b \notin N_k a$. To control this, we can treat k as a random variable chosen to make the probability of missing any true match below a budget. Let $Ma = |\{b : S_{ab} \geq \tau\}|$ be the number of true matches for a. If we had exact ordering by S_{ab}, then choosing $k \geq Ma$ would guarantee no misses. With approximate retrieval, we instead rely on a recall-at-k curve. Define [27]

$$\rho k = \mathbb{P}\big(\exists b \text{ with } S_{ab} \geq \tau \text{ and } b \notin N_k a \mid a \sim \mathcal{D}_A\big), \tag{4.4}$$

where $\mathcal{D}_A$ is the distribution of queries induced by rows in A. We can estimate ρk using sampled queries and exact neighbors computed by brute force on a smaller subset or via exact indexes on a sample. The calibrated guarantee then chooses the smallest k such that $\rho k \leq \delta_{\text{ann}}$ with confidence. The overall miss probability combines proxy pruning, ANN misses, and any upstream prefilters. If stages are conditionally independent given similarity, one can multiply complements; without independence, a conservative bound uses [28]

$$\delta_{\text{total}} \leq \delta_{\text{proxy}} \ \delta_{\text{ann}} \ \delta_{\text{prefilter}}. \tag{4.5}$$

This additivity motivates allocating miss budgets across stages via optimization.

Sketch-based prefilters provide another lever. For token-based components, MinHash sketches estimate Jaccard similarity. If J is the true Jaccard similarity and $\hat{J}$ is the estimate from k hash functions, then $\hat{J}$ is an unbiased estimator with variance $J1 - Jk$. A one-sided Hoeffding bound yields [29]

$$\mathbb{P}\hat{J} \leq J - \varepsilon \leq \exp{-2k\varepsilon^2}. \tag{4.6}$$

Thus if we include candidates whenever $\hat{J} \geq \tau_J - \varepsilon$ then any pair with $J \geq \tau_J$ is missed with probability at most $\exp{-2k\varepsilon^2}$. This bound is conservative but explicit. Similar bounds apply to random projection sketches for cosine similarity. If we use k random hyperplanes and compute the fraction of agreeing signs $\hat{p}$, then the angle estimator concentrates around the true angle. While the exact distribution is binomial, a Chernoff bound provides

$$\mathbb{P}|\hat{p} - p| \geq \varepsilon \leq 2\exp{-2k\varepsilon^2}, \tag{4.7}$$

where p is the true collision probability [30]. Such sketch stages can be used as cheap filters before expensive ANN search, with quantifiable miss probability.

A learned pipeline can incorporate these bounds into training through a Lagrangian that trades recall against cost. Let $c\phi$ denote expected candidate cost per query, including compute and memory, and let $\hat{\delta}\phi$ denote an empirical estimate of miss probability with a conservative margin. We want to minimize cost subject to $\hat{\delta}\phi \leq \delta_0$. The constrained problem is [31]

$$\min_{\phi} \ c\phi \quad \text{subject to} \quad \hat{\delta}\phi \leq \delta_0. \tag{4.8}$$

A Lagrangian relaxation yields

$$\min_{\phi} \ c\phi \ \lambda \ \max 0, \hat{\delta}\phi - \delta_0, \tag{4.9}$$

where $\lambda \geq 0$ is tuned to enforce the constraint. Because $\hat{\delta}\phi$ may be non-smooth (it depends on retrieval outcomes), we can use a surrogate such as a differentiable estimate of inclusion probabilities on sampled positives. For example, if $p_{ij}\phi$ is a soft inclusion probability, then a surrogate miss rate is $\frac{1}{|\mathcal{P}|} \ {}_{i,j\in\mathcal{P}} 1 - p_{ij}$. The gradient of $c\phi$ can be estimated via differentiable proxies (e.g., expected bucket sizes predicted by a model), while hard parameters are ultimately set via grid search constrained by calibration bounds.

Multi-objective optimization arises when we account for latency, memory, and energy [32]. Let $L\phi$ be expected latency, $M\phi$ memory footprint, and $E\phi$ energy cost, each possibly estimated by performance models. One approach uses a weighted sum

$$\min_{\phi} \ \alpha L\phi \ \beta M\phi \ \gamma E\phi \quad \text{subject to} \quad \hat{\delta}\phi \leq \delta_0, \tag{4.10}$$

with weights reflecting deployment priorities. Another approach is to compute a Pareto frontier by searching over weights and retaining nondominated configurations [33]. Because join workloads can vary, a robust formulation considers worst-case or high-quantile costs over query distributions. For instance, minimizing $\mathrm{CVaR}_q L\phi$ can reduce tail latency under skew.

We also address limitations. Optimal blocking, defined as partitioning records into blocks to minimize comparisons while covering all true matches, is computationally hard. A simplified decision version can be sketched as follows: given a universe of records and a set of true match pairs, determine whether there exists a partition into at most K blocks such that each true match pair lies within some block and the total number of within-block pairs is at most T [34]. This resembles graph partitioning with constraints and can encode NP-hard problems by mapping vertices to records and required edges to true matches. Consequently, learned heuristics and approximate methods are necessary, and error control must be probabilistic rather than exact for large heterogeneous datasets.

5. Indexing, Data Structures, and Distributed Execution Mechanics

A similarity join at scale must integrate candidate generation with distributed storage, indexing, and execution [35]. The primary costs are scanning embeddings, performing index lookups, shuffling candidates for verification, and computing similarity for candidates. Candidate generation should minimize network traffic and avoid hotspots caused by skewed buckets or popular records.

We consider a distributed environment with p workers. Table B is indexed and partitioned across workers, while queries from A may be streamed or co-partitioned [36]. A common design is to build a per-partition ANN index for B and route each query a to all partitions or a subset based on a coarse routing function. Routing is crucial: broadcasting every query to all partitions yields $Op|A|$ query fanout and becomes network-bound. Thus we introduce a two-level index: a coarse quantizer assigns each z_b to one of K centroids, and each centroid corresponds to an inverted list stored on specific workers. Queries probe a small number n_{probe} of centroids, retrieving candidates from their lists. This is analogous to inverted file indexes for vector search [37]. The routing function is then a nearest-centroid lookup, which can be done with a compressed centroid table replicated across workers.

Let Q be the number of queries (rows in A). The expected candidate retrieval cost can be approximated by

$$\mathbb{E}\text{cands per query} \approx \sum_{c\in\mathcal{C}_{\text{probe}}a} \mathbb{E}|L_c| \cdot \pi_{\text{list}}c \mid a, \tag{5.1}$$

where L_c is the inverted list size for centroid c and π_{list} is the probability of scanning an item given probing. In practice, ANN indexes do not scan whole lists; they use product quantization to score codes quickly and keep top results [38]. Nevertheless, list sizes influence cache behavior and tail latency. Balancing lists is therefore a systems concern: centroids should be trained to reduce imbalance, and extremely large lists may require splitting across workers or additional hashing.

Data structures must store embeddings or their compressed forms [39]. For dense vectors, storing $|B|d$ floats can be prohibitive. Compression via quantization reduces memory but introduces approximation error. A typical representation is to store a coarse centroid id plus m sub-quantizer codes, along with optional residual norms for bounds. Verification may require accessing the original embedding or recomputing it from raw row data [40]. Recomputing embeddings on the fly can be expensive, so systems often store either full embeddings for verification or a higher-precision compressed representation sufficient for exact predicate evaluation. If the join predicate is defined by s on the full embedding, exact verification requires access to z_b. If z_b is stored in compressed form, verification becomes approximate unless we reconstruct with enough precision and account for error margins. To preserve correctness relative to the defined predicate, one can store embeddings in half precision and compute similarity in float32, with known rounding bounds, or store a residual correction [41].

Candidate generation and verification can be pipelined. For each query a, we compute z_a, route to relevant partitions, retrieve candidate ids, then fetch candidate vectors for verification. Fetching vectors can dominate cost due to random access. To mitigate this, we batch queries so that candidate ids can be sorted, enabling sequential reads, and we cache popular vectors [42]. Another technique is to compute verification using compressed codes with conservative bounds, fetching full vectors only for ambiguous cases. For example, if we have a lower bound $S_{ab} \geq \hat{S}_{ab} - e_b$ and an upper bound $S_{ab} \leq \hat{S}_{ab}\ e_b$, then if $\hat{S}_{ab} - e_b \geq \tau$ we can accept without fetching the full vector, and if $\hat{S}_{ab}\ e_b < \tau$ we can reject. Only if $\tau \in \hat{S}_{ab} - e_b, \hat{S}_{ab}\ e_b$ do we fetch full precision. This creates a three-way decision that can reduce random IO. The savings depend on how often candidates fall near the threshold [43].

Distributed verification must also handle duplicates and aggregation. A candidate pair a, b may be produced by multiple probed lists or hash tables. Deduplication can be done by sorting candidate ids per query or using hash sets, but hash sets can be memory heavy. A systems-friendly method is to allow duplicates through candidate generation and deduplicate after verification using a streaming group-by on a, b, which can be implemented with external sort [44]. The trade-off is extra verification work. If verification is expensive, earlier deduplication may be worthwhile, but then memory must be managed carefully to avoid spills.

Skew is an important challenge. Some queries may have many near neighbors, causing large candidate sets, while others have few [45]. Similarly, some buckets or centroids may be disproportionately large. Tail latency then becomes dominated by worst-case partitions or queries. To address this, we can enforce per-query candidate caps $k_{\max}$ and accept a controlled miss probability for high-degree queries. Error control must then be conditional: we can guarantee low miss probability for queries whose true match count is below a threshold, and provide a separate bound for the tail [46]. Alternatively, we can allocate a larger budget to heavy queries by dynamically increasing probing or beam width based on early signals such as the distribution of approximate scores. Let $\hat{s}_1 \geq \hat{s}_2 \geq \dots$ be approximate scores for retrieved candidates. If the score drops sharply below τ, we may stop early; if many candidates are near τ, we may probe more lists. This adaptive strategy can be framed as sequential testing with error control: decide when to stop probing so that the probability of missing a true match is below δ. A simple conservative rule uses calibrated curves mapping observed top score and score gaps to required probing depth [47].

Query planning introduces additional complexity. If the similarity join is part of a multi-join plan, the planner may reorder joins or push down filters. For similarity joins, reordering can change query distributions, affecting calibration. A planner-aware design can treat the join operator as parameterized by ϕ and provide a cost and error model to the optimizer [48]. The optimizer then solves a trade-off: reducing intermediate sizes may require more aggressive candidate pruning, which increases error, which may or may not be acceptable downstream. One can model plan selection as dynamic programming over join order with augmented state that includes an error budget. Let state S represent a subset of relations joined so far, and let e be remaining error budget. The DP recurrence might minimize cost CS, e by choosing the next join and allocating error budget to its candidate generator [49]. While continuous budgets make DP large, discretizing e into bins yields an approximate algorithm. The resulting planner is an approximation algorithm itself; error control becomes layered: the operator provides calibrated bounds, and the planner composes them using conservative union bounds across operators.

Complexity analysis clarifies scaling [50]. Suppose each query probes n_{probe} lists and examines k candidate codes per list, then verification checks v candidates. The dominant per-query time is approximately

$$T_{\text{query}} \approx n_{\text{probe}} \cdot T_{\text{route}} \;\; k \cdot T_{\text{score}} \;\; v \cdot T_{\text{verify}} \;\; T_{\text{fetch}}, \tag{5.2}$$

where T_{fetch} captures memory and network access. In a distributed setting, the total time is influenced by parallelism and skew; ideally, throughput scales as OQp, but imbalance introduces an effective $p_{\text{eff}} < p$. Candidate generation aims to reduce v and T_{fetch} by limiting candidates and enabling cache-friendly access patterns. Memory scales with index size, often $O|B|$ times code length, plus overhead for inverted lists. Building the index is typically $O|B|d$ for embedding computation plus $O|B| \log |B|$ for sorting into lists and training quantizers.

6. Optimization, Training Objectives, and Error Analysis for Approximate Joins

We now connect learning and optimization to error control and performance engineering [51]. The key idea is to tune embedding parameters θ and candidate-generator parameters ϕ to jointly improve the geometry for retrieval while satisfying resource constraints.

A joint objective can be written as an expected verification loss plus a cost regularizer and an omission penalty. Let $\mathcal{D}$ be a distribution over query rows a and candidate rows b with positives defined by $S_{ab} \geq \tau$.

Define a verification function va, b that returns $\mathbb{I}\{S_{ab} \geq \tau\}$, which is deterministic given θ. Candidate generation induces inclusion probability $p_{ab}\theta, \phi$. The expected missed-match rate is $\mathbb{E}\mathbb{I}\{S_{ab} \geq \tau\}1 - p_{ab}$ under a sampling scheme over pairs. The expected verification workload is $\mathbb{E}p_{ab}$ times pair sampling density, or more concretely expected candidates per query. A stylized joint optimization is

$$\min_{\theta,\phi} \mathcal{L}_{\text{rep}}\theta \;\; \mu\mathcal{C}\phi \;\; \lambda\mathbb{E}\big[\mathbb{I}\{S_{ab} \geq \tau\}1 - p_{ab}\theta, \phi\big], \tag{6.1}$$

where $\mathcal{L}_{\text{rep}}$ is the representation learning loss, $\mathcal{C}\phi$ is a cost model, and the final term penalizes omissions. However, because S_{ab} depends on θ, the indicator is not differentiable. In practice, one can fix θ to define the join predicate and then optimize ϕ for candidate generation, or one can co-train with a smoothed approximation, replacing $\mathbb{I}\{S_{ab} \geq \tau\}$ with $\sigma\beta S_{ab} - \tau$.

Gradient descent variants are relevant because the joint landscape can be ill-conditioned [52]. The embedding model may be trained with adaptive optimizers, while candidate-generator tuning may require discrete parameter search. A pragmatic approach alternates: train θ for representation quality and retrieval-friendly geometry, then calibrate and tune ϕ to meet error constraints, then optionally fine-tune θ with a retrieval-aware regularizer that encourages stable neighborhoods. For example, one can add a neighborhood-preservation loss that encourages margin separation around the join threshold. Let b be a positive neighbor with $S_{ab} \geq \tau$ and b^- a near-negative with $S_{ab^-} < \tau$ but close. Then a hinge term [53]

$$\ell_{\text{thresh}}a = \max\big(0,\; \tau \;\; m - S_{ab} \;\; S_{ab^-}\big) \tag{6.2}$$

encourages a gap m around the threshold, reducing ambiguous cases and potentially improving candidate generation recall for a fixed budget.

Error analysis for approximate queries often uses concentration and tail bounds. In our setting, the final error arises from missed matches, which is a Bernoulli event per true match pair [54]. If we model misses as independent given similarity (an approximation), then the number of missed matches over all true matches is binomial with mean $\delta|R|$. Even without independence, Markov's inequality bounds expected missed matches. A more useful quantity is per-query miss probability: for each a, define E_a as the event that at least one true match is missed. Then overall fraction of affected queries is $\mathbb{E}\mathbb{I}\{E_a\}$. This is often more relevant than pairwise recall because downstream tasks may require that each query find at least one match [55]. Calibration can therefore focus on ρk defined earlier rather than pairwise recall.

Approximate indexing structures introduce additional randomness and approximation. Consider random projection hashing with L bits and T tables. For a fixed pair with angle θ, the probability of collision in a table depends on the number of matching bits [56]. If we use a banding scheme, then inclusion probability can be computed in terms of $p\theta$, but learned embeddings may violate isotropy assumptions. Even so, we can still use the functional form as a parametric family and fit an effective p curve empirically. The key is not to trust theoretical curves blindly but to use them as structured priors in calibration.

Bayesian-ish modeling can help quantify uncertainty in calibration [57]. Suppose we estimate miss probability as a function of an observable proxy u, such as approximate score or quantized distance. We can model

$$\mathbb{P}I = 0 \mid S \geq \tau, U = u = \sigma g_{\psi}u, \tag{6.3}$$

where g_{ψ} is a small regression model. Given calibration data, we can place a prior on ψ and compute a posterior, or approximate it with Laplace [58]. Then for each query, we can integrate uncertainty to produce conservative miss estimates. In deployment, one may not want full Bayesian inference, but a practical approximation is to use conformal prediction ideas: compute nonconformity scores for calibration positives and choose thresholds that guarantee coverage. For example, if we use U as a score and include candidates whenever $U \geq t$, then selecting t as the $\lfloor n \;\; 1\delta \rfloor$-th order statistic yields a finite-sample guarantee on miss rate under exchangeability. This provides a simple calibration rule with explicit finite-sample control, though it assumes calibration and deployment are exchangeable [59].

Constraint handling for latency and energy can be formalized. Let $b\phi$ be candidate budget per query, and suppose latency is approximately linear in budget for a fixed hardware and caching regime, $L\phi \approx L_0\ b\phi \cdot \ell_1$, while energy similarly scales, $E\phi \approx E_0\ b\phi \cdot e_1$. If we require $L\phi \leq L_{\max}$ and $E\phi \leq E_{\max}$, we can write

$$\min_{\phi}\ \hat{\delta}\phi \quad \text{subject to} \quad b\phi \leq \min\big(\frac{L_{\max} - L_0}{\ell_1}, \frac{E_{\max} - E_0}{e_1}\big). \tag{6.4}$$

This reduces to choosing the best candidate generator under a budget. More refined models include non-linearities due to cache misses and network saturation, but the constrained formulation remains useful for reasoning and for integration into query optimizers [60].

We also consider the interaction with one-to-one matching constraints common in entity resolution. After generating candidate pairs and verifying similarity, one may solve a maximum weight matching problem on a bipartite graph with edge weights S_{ab}. Candidate omission then affects the feasible set of the matching problem and can have amplified impact: missing a single critical edge can change the optimal assignment. Error control for downstream matching is therefore stricter than pairwise recall [61]. A conservative approach is to allocate smaller miss budgets when one-to-one constraints are used, or to include a diversity-aware candidate generation strategy that ensures multiple plausible edges per node. This can be modeled as a submodular coverage objective over candidate edges, but optimizing it exactly is expensive. A practical heuristic is to ensure each a retrieves at least k_0 candidates above a lower threshold $\tau_{\text{low}} < \tau$, then verification applies τ. Calibration then targets the probability that at least one true match is in the retrieved set, which aligns better with matching.

7. Evaluation Methodology, Workload Design, and Reproducibility Considerations

Evaluating learned similarity joins with error-controlled candidate generation requires careful methodology because both accuracy and performance depend on data distribution, thresholds, and system settings [62]. Additionally, calibration-based guarantees can fail under distribution shift, so evaluation must include drift scenarios.

A baseline requirement is to define the join predicate precisely. Since the predicate is based on embedding similarity, evaluation should treat the exact computation of S_{ab} as ground truth for the operator, even if the embedding is itself learned. For a given θ, τ, one can compute exact results on manageable subsets or by using exact indexes for verification. The operator's correctness is measured as the fraction of true result pairs returned, and the miss probability per query [63]. Because exact computation on full corpora may be infeasible, evaluation often uses sampled pairs and stratified sampling over similarity ranges. For example, one can sample candidate pairs from ANN outputs and from random pairs, compute exact S_{ab}, and estimate recall with importance weighting to account for sampling bias. Without careful sampling, recall estimates can be overly optimistic because ANN outputs over-represent high-similarity pairs.

Calibration evaluation must separate calibration data from test data. The calibration procedure selects thresholds or parameters to satisfy a target δ_0 with confidence, using held-out positives defined by $S_{ab} \geq \tau$. The test then measures realized miss rates [64]. A rigorous approach repeats this over multiple random splits to estimate variability. Drift evaluation retrains embeddings or applies the operator to data from a later time window, then checks whether calibration still holds. If it fails, one can quantify how often recalibration is needed [65].

Performance evaluation should report throughput, tail latency, and resource consumption. Throughput is measured as verified pairs per second or queries per second at a given cluster size. Tail latency is important due to skew; reporting $p95$ and $p99$ per-query latency can reveal bucket hotspots. Resource metrics include memory footprint of indexes, network bytes shuffled, and CPU utilization [66]. Energy can be estimated via hardware counters or proxy metrics if direct measurement is not available, but any proxy should be stated transparently.

Workload design should include multiple join selectivities. When τ is high, result sets are small and candidate generation can be aggressive; when τ is low, many pairs satisfy the predicate and the join becomes inherently expensive. Error control behaves differently in these regimes [67]. For low selectivity, missing a small fraction of pairs may still yield large absolute error, so evaluation should include absolute missed matches, not only recall. Conversely, for high selectivity, per-query miss probability may be sensitive to a small number of borderline pairs, so evaluation should examine similarity distributions near τ.

Reproducibility requires deterministic configuration and reporting. Learned systems involve randomness in embedding training, hashing, index building, and ANN search [68]. To make results reproducible, one should fix random seeds, record model checkpoints, and version datasets and preprocessing pipelines. Index parameters, such as quantizer training epochs and graph construction settings, must be logged. Hardware and software environments affect performance, so evaluations should report CPU type, memory, storage, network, and relevant library versions [69]. Because similarity joins can be IO-bound, storage configuration and caching policies can materially change results.

Finally, evaluation should connect operator-level guarantees to user-facing requirements. If a user specifies δ_0 as a maximum miss probability, the system should demonstrate that realized miss rates are below δ_0 across workloads and that the confidence level is respected. If the guarantee is conditional, for example valid under exchangeability between calibration and test, then evaluation should show how violations of this assumption affect results [70]. A useful stress test is to shift the distribution of queries by changing table composition, introducing new categories, or altering missingness patterns, then measure calibration degradation.

8. Conclusion

Learned similarity joins over large tabular corpora require candidate generation mechanisms that are both efficient and predictable. By defining the join predicate in embedding space, the operator's correctness becomes a matter of retrieving and verifying all pairs whose learned similarity exceeds a threshold, but approximate retrieval and distributed execution introduce unavoidable trade-offs. This paper presented a framework for error-controlled candidate generation that combines learned representations with calibrated approximations [71]. The approach leverages proxy scores, sketches, and quantization to reduce compute, and it translates empirical inclusion behavior into conservative miss-probability bounds using calibration and concentration results. We connected these bounds to constraint-aware optimization, enabling the tuning of candidate generation parameters under latency, memory, and energy budgets, and we discussed systems-level design choices for distributed indexing, routing, verification, and skew mitigation. We also highlighted computational hardness that motivates heuristic candidate generation and emphasized evaluation practices that treat the exact embedding-based predicate as ground truth while stress-testing calibration under drift. Overall, the design perspective is to treat candidate generation not as an opaque heuristic but as an approximate query processing component with explicit, auditable error controls that integrate with representation learning and query planning [72].

References

[1] J. Shao, J. Wang, and L. Yang, "Cscw - a production monitoring and data processing system for the textile enterprise based on multi-agent," in *Proceedings of the ACM 2011 conference on Computer supported cooperative work*, pp. 713–716, ACM, 3 2011.

[2] J.-F. Martínez, A.-B. García, A.-M. Sanz, L. López, V. Hernández, and A. Dasilva, "Eatis - an approach for applying multi-agent technology into wireless sensor networks," in *Proceedings of the 2007 Euro American conference on Telematics and information systems*, pp. 22–8, ACM, 5 2007.

[3] Z. Ye, Y. Chen, and H. Zhang, “Distributed consensus of delayed multi-agent systems with nonlinear dynamics via intermittent control,” *Asian Journal of Control*, vol. 18, pp. 964–975, 6 2015.

[4] C. Ramachandran, R. Malik, X. Jin, J. Gao, K. Nahrstedt, and J. Han, “Videomule: a consensus learning approach to multi-label classification from noisy user-generated videos,” in *Proceedings of the 17th ACM international conference on Multimedia*, pp. 721–724, 2009.

[5] S. Landau and S. Picault, “Multi-agent-systems and applications - modeling adaptive multi-agent systems inspired by developmental biology,” *Lecture Notes in Computer Science*, vol. 2322, pp. 221–232, 4 2002.

[6] T. H. Zhao and D. S. Wang, “Research on monitoring system of water resources in irrigation region based on multi-agent,” *IOP Conference Series: Earth and Environmental Science*, vol. 15, pp. 042012–, 11 2012.

[7] N. S. Boss, A. S. Jensen, and J. Villadsen, “Building multi-agent systems using jason,” *Annals of Mathematics and Artificial Intelligence*, vol. 59, pp. 373–388, 5 2010.

[8] X. Mao, F. Hou, and W. Wu, *Multi-Agent System Approach for Modeling and Supporting Software Crowdsourcing*, pp. 73–89. Springer Berlin Heidelberg, 5 2015.

[9] Y. Shao, J. Yang, and G. Chen, *Research on Networked Collaborative Operations Based on Multi-agent System*, pp. 517–522. Springer International Publishing, 11 2017.

[10] E. Majd and V. Balakrishnan, “A reputation-oriented trust model for multi-agent environments,” *Industrial Management & Data Systems*, vol. 116, pp. 1380–1396, 8 2016.

[11] Y. Huang, W. Yang, M. Wang, and J. Wu, “A testing framework of adaptive web application based on multi-agent collaboration,” *DEStech Transactions on Engineering and Technology Research*, 11 2016.

[12] Z. Yu-lai, “Fixed-formation control based on event-driven of multi-agent system,” *Journal of Mathematics and Informatics*, vol. 9, pp. 19–27, 10 2017.

[13] D. G. Mikulski, F. L. Lewis, E. Y. L. Gu, and G. Hudas, “Trust dynamics in multi-agent coalition formation,” *SPIE Proceedings*, vol. 8045, pp. 252–266, 5 2011.

[14] M. Ragaglia, M. Prandini, and L. Bascetta, *MESAS - Multi-agent Poli-RRT**. Germany: Springer International Publishing, 10 2016.

[15] J. Shi, X. He, Z. Wang, and D. Zhou, “Iterative consensus for a class of second-order multi-agent systems,” *Journal of Intelligent & Robotic Systems*, vol. 73, pp. 655–664, 10 2013.

[16] M. G. Buhnerkempe, M. G. Roberts, A. P. Dobson, H. Heesterbeek, P. J. Hudson, and J. O. Lloyd-Smith, “Eight challenges in modelling disease ecology in multi-host, multi-agent systems,” *Epidemics*, vol. 10, pp. 26–30, 12 2014.

[17] S. C. K. R, “An efficient multiprocessor memory management framework using multi-agents,” *Computer Science & Engineering: An International Journal*, vol. 2, pp. 65–81, 12 2012.

[18] P. Blikstein, W. Rand, and U. Wilensky, “Aamas - participatory, embodied, multi-agent simulation,” in *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pp. 1457–1458, ACM, 5 2006.

[19] H. Yu, “Leaderless and leader-following flocking motion via coordinated control,” *International Journal on Smart Sensing and Intelligent Systems*, vol. 7, pp. 1436–1452, 9 2014.

[20] V. Vijaykumar, R. Chandrasekar, and T. Srinivasan, “An obstacle avoidance strategy to ant colony optimization algorithm for classification in event logs,” in *2006 IEEE Conference on Cybernetics and Intelligent Systems*, pp. 1–6, 2006.

[21] Z. Y. Liu and J. Wang, “A research into an intrusion detection system based on immune principle and multi-agent in wsn,” *Advanced Materials Research*, vol. 433-440, pp. 5157–5161, 1 2012.

[22] Y. Lao and W. Leong, *PRICAI - A Multi-agent Based Approach to the Inventory Routing Problem*, pp. 345–354. Germany: Springer Berlin Heidelberg, 8 2002.

[23] B. Banerjee and L. Kraemer, “Action discovery for single and multi-agent reinforcement learning,” *Advances in Complex Systems*, vol. 14, pp. 279–305, 11 2011.

[24] S. Gouardères, G. Gouardères, and P. Delpy, *PAKM - From Speech Acts to Multi-agent Systems: The MAYBE Method*, pp. 144–155. Germany: Springer Berlin Heidelberg, 12 2002.

[25] H. Hu, Y. Yoon, and Z. Lin, “Consensus of multi-agent systems with control-affine nonlinear dynamics,” *Unmanned Systems*, vol. 04, pp. 61–73, 3 2016.

[26] R. C. Cardoso and R. H. Bordini, “A multi-agent extension of a hierarchical task network planning formalism,” *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal*, vol. 6, pp. 5–17, 6 2017.

[27] W. Zhang and Y. Liu, “Observer-based distributed consensus for general nonlinear multi-agent systems with interval control inputs,” *International Journal of Control*, vol. 89, pp. 84–98, 7 2015.

[28] T. Srinivasan, R. Chandrasekar, V. Vijaykumar, V. Mahadevan, A. Meyyappan, and M. Nivedita, “Exploring the synergism of a multiple auction-based task allocation scheme for power-aware intrusion detection in wireless ad-hoc networks,” in *2006 10th IEEE Singapore International Conference on Communication Systems*, pp. 1–5, IEEE, 2006.

[29] Y. Luo and J. Gao, “A research on the multi-agent state monitoring technology in complex communication systems,” *DEStech Transactions on Engineering and Technology Research*, 4 2017.

[30] D. Weyns, A. Helleboogh, T. Holvoet, and M. Schumacher, “The agent environment in multi-agent systems: A middleware perspective,” *Multiagent and Grid Systems: An International Journal of Data Science and Artificial Intelligence*, vol. 5, pp. 93–108, 2 2009.

[31] P. Sengupta and U. Wilensky, *Lowering the Learning Threshold: Multi-Agent-Based Models and Learning Electricity*, pp. 141–171. Springer Netherlands, 2 2011.

[32] C. Wang, K. Zhou, L. Li, and S. Yang, “Multi-agent simulation-based residential electricity pricing schemes design and user selection decision-making,” *Natural Hazards*, vol. 90, pp. 1309–1327, 11 2017.

[33] Z.-H. Yang, Y. Song, M. Zheng, and W. Hou, “Consensus of multi-agent systems under switching agent dynamics and jumping network topologies,” *International Journal of Automation and Computing*, vol. 13, pp. 438–446, 7 2016.

[34] V. Julián, C. Carrascosa, M. Rebollo, J. Soler, and V. Botti, “Ccia - simba: An approach for real-time multi-agent systems,” *Lecture Notes in Computer Science*, pp. 282–293, 10 2002.

[35] S. Su, “Distributed coordination control of complex multi-agent networks with dynamic interaction topologies,” 12 2017.

[36] D. Li, J. Ma, H. Zhu, and M. Sun, “The consensus of multi-agent systems with uncertainties and randomly occurring nonlinearities via impulsive control,” *International Journal of Control, Automation and Systems*, vol. 14, pp. 1005–1011, 6 2016.

[37] R. Chandrasekar and T. Srinivasan, “An improved probabilistic ant based clustering for distributed databases,” in *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI*, pp. 2701–2706, 2007.

[38] S. Niemczyk, N. Fredivianus, and K. Geihs, “Sac - on-the-fly transformation synthesis for information sharing in heterogeneous multi-agent systems,” in *Proceedings of the Symposium on Applied Computing*, pp. 297–299, ACM, 4 2017.

[39] T. Doi, Y. Tahara, and S. Honiden, “Aamas - iom/t: an interaction description language for multi-agent systems,” in *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pp. 778–785, ACM, 7 2005.

[40] W. Vasconcelos, D. Robertson, J. Agustíý, C. Sierra, M. Wooldridge, S. Parsons, C. Walton, and J. Sabater, *A Lifecycle for Models of Large Multi-agent Systems*, pp. 297–317. Germany: Springer Berlin Heidelberg, 2 2002.

[41] N. Y. Mutovkina, *Fuzzy Complex Assessment of Activities of the Agent in Multi-Agent System*, pp. 303–313. Springer International Publishing, 8 2017.

[42] B. Liu, W. Hu, J. Zhang, and H. Su, “Controllability of discrete-time multi-agent systems with multiple leaders on fixed networks,” *Communications in Theoretical Physics*, vol. 58, pp. 856–862, 12 2012.

[43] X. Zhang, H. Xu, and B. Shrestha, *Building a Health Care Multi-Agent Simulation Sysmte with Role-Based Modeling*. IGI Global, 1 2011.

[44] J. L. Meliá, “A multi-agent safety response model in the construction industry.,” *Work (Reading, Mass.)*, vol. 51, pp. 549–556, 7 2015.

[45] R. Dobbe, D. Fridovich-Keil, and C. Tomlin, “Fully decentralized policies for multi-agent systems: An information theoretic approach,” 1 2017.

[46] R. L. V. Moritz and M. Middendorf, “Gecco - evolutionary inheritance mechanisms for multi-criteriadecision making in multi-agent systems,” in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pp. 65–72, ACM, 7 2015.

[47] X. Wang, J. Li, J. Xing, R. Wang, L. Xie, and X. Zhang, “A novel finite-time average consensus protocol for multi-agent systems with switching topology:,” *Transactions of the Institute of Measurement and Control*, vol. 40, pp. 606–614, 8 2016.

[48] L. Zhang, J. Wang, and Q. Shi, “Multi-agent based modeling and simulating for evacuation process in stadium,” *Journal of Systems Science and Complexity*, vol. 27, pp. 430–444, 6 2014.

[49] L. Zhang, Z. Qi, Q. Z. Wang, X. P. Wang, and X. Shen, “Building a multi-agent system for emergency logistics collaborative decision,” *Applied Mechanics and Materials*, vol. 513-517, pp. 2041–2044, 2 2014.

[50] M. Štula, D. Stipaničev, and J. Maras, “Distributed computation multi-agent system,” *New Generation Computing*, vol. 31, pp. 187–209, 8 2013.

[51] A. Bezek, M. Gams, and I. Bratko, “Aamas - multi-agent strategic modeling in a robotic soccer domain,” in *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pp. 457–464, ACM, 5 2006.

[52] T. Fu-xiao, G. Xinping, and L. Derong, “Consensus protocol for multi-agent continuous systems ⁄,” *Chinese Physics B*, vol. 17, pp. 3531–3535, 10 2008.

[53] R. Chandrasekar, V. Vijaykumar, and T. Srinivasan, “Probabilistic ant based clustering for distributed databases,” in *2006 3rd International IEEE Conference Intelligent Systems*, pp. 538–545, IEEE, 2006.

[54] C. Nowzari, J. E. Cortes, and G. J. Pappas, “Cooperative control of multi-agent systems - event-triggered communication and control for multi-agent average consensus,” 3 2017.

[55] Y. Zheng, X. Xu, T. Shi, D. Fang, and P. An, “A multi-agent approach to design a wlan system,” *International Journal of Online and Biomedical Engineering (iJOE)*, vol. 13, pp. 46–64, 11 2017.

[56] Y. Zheng and L. Wang, “A novel group consensus protocol for heterogeneous multi-agent systems,” *International Journal of Control*, vol. 88, pp. 2347–2353, 7 2015.

[57] S. Rudolph and M. Croitoru, *SGAI Conf. - Multi-Agent Knowledge Allocation*, pp. 165–178. Springer London, 10 2012.

[58] E. A. Olajubu, G. A. Aderounmu, and E. R. Adagunodo, “Network resources management in a multi-agent system: a simulative approach,” *South African Journal of Science*, vol. 106, pp. 1–6, 9 2010.

[59] T. D. Nguyen and Q. Bai, *BiTrust: A Comprehensive Trust Management Model for Multi-agent Systems*, pp. 3–16. Germany: Springer International Publishing, 4 2017.

[60] M. Winikoff, “Challenges and directions for engineering multi-agent systems,” 1 2012.

[61] X. Lin, Y. Zheng, and L. Wang, “Consensus of switched multi-agent systems with random networks,” *International Journal of Control*, vol. 90, pp. 1113–1122, 7 2016.

[62] M. Kim and E. T. Matson, *PAAMS (Workshops) - A Cost-Optimization Model in Multi-agent System Routing for Drone Delivery*, pp. 40–51. Germany: Springer International Publishing, 5 2017.

[63] X. Xie, X. Guo, and C. He, “Research on spatial knowledge system of heterogeneous spatial information based on cloud computing technology,” *International Journal of Grid and Distributed Computing*, vol. 10, pp. 75–86, 1 2017.

[64] N. Bulling, “A survey of multi-agent decision making,” *KI - Künstliche Intelligenz*, vol. 28, pp. 147–158, 7 2014.

[65] F. Xiu-Jun and S. K. Y, “Competition of multi-agent systems: Analysis of a three-company econophysics model,” *Chinese Physics Letters*, vol. 26, pp. 098901–, 9 2009.

[66] J. Bajo, P. Mathieu, and M. J. Escalona, “Multi-agent technologies in economics,” *Intelligent Systems in Accounting, Finance and Management*, vol. 24, pp. 59–61, 9 2017.

[67] S. Yang, X. Liao, Y. Liu, X. Chen, and D. Ge, “Consensus of delayed multi-agent systems via intermittent impulsive control,” *Asian Journal of Control*, vol. 19, pp. 941–950, 12 2016.

[68] S. Hoet and N. Sabouret, “Apprentissage par renforcement d’actes de communication dans un système multi-agent,” *Revue d’intelligence artificielle*, vol. 24, pp. 159–188, 4 2010.

[69] T. Srinivasan, V. Vijaykumar, and R. Chandrasekar, “An auction based task allocation scheme for power-aware intrusion detection in wireless ad-hoc networks,” in *2006 IFIP International Conference on Wireless and Optical Communications Networks*, pp. 5–pp, IEEE, 2006.

[70] S. Sharma, “Avatarsim: A multi-agent system for emergency evacuation simulation,” *Journal of Computational Methods in Sciences and Engineering*, vol. 9, pp. 13–22, 7 2009.

[71] H. Zhao, S. Xu, and D. Yuan, “Consensus of data-sampled multi-agent systems with markovian switching topologies†,” *Asian Journal of Control*, vol. 14, pp. 1366–1373, 8 2011.

[72] S.-J. Lin and T.-F. Chen, “Multi-agent architecture for corporate operating performance assessment,” *Neural Processing Letters*, vol. 43, pp. 115–132, 1 2015.