Self-Balancing, Memory Efficient, Dynamic Metric Space Data Maintenance, for Rapid Multi-Kernel Estimation

Aditya S Ellendula[⋆][✉] and Chandrajit Bajaj[⋆⋆][✉]

University of Texas at Austin

Abstract. We present a dynamic self-balancing octree data structure that fundamentally transforms neighborhood maintenance in evolving metric spaces. Learning systems, from deep networks to reinforcement learning agents, operate as dynamical systems whose trajectories through high-dimensional spaces require efficient importance sampling for optimal convergence. Generative models operate as dynamical systems whose latent representations cannot be learned in one shot, but rather grow and evolve sequentially during training—requiring continuous adaptation of spatial relationships. Our two-parameter (K, α) dynamic octree addresses this challenge by providing a computational fabric that efficiently organizes both the generation flow and querying flow operating on different time scales by enabling logarithmic-time updates and queries without requiring complete rebuilding as distributions evolve. We demonstrate its effectiveness across four key applications: (1) accelerating Stein Variational Gradient Descent by enabling larger particle sets with reduced computation; (2) supporting real-time incremental KNN classification with logarithmic updates; (3) improving retrieval-augmented generation by enabling efficient, incremental semantic indexing; and (4) showing that maintaining both input and latent space structures accelerates convergence and improves sample efficiency. Across all applications, our experimental results confirm exponential performance improvements over standard methods while maintaining accuracy. These improvements are particularly significant for high-dimensional spaces where efficient neighborhood maintenance is crucial to navigate complex latent manifolds. By providing guaranteed logarithmic bounds for both update and query operations, our approach enables more data-efficient solutions to previously computationally prohibitive problems, establishing a new approach to dynamic spatial relationship maintenance in machine learning.

Keywords: Dynamic Octree · Neighborhood Maintenance · Importance Sampling · Logarithmic-Time Updates · Incremental KNN Classification · Retrieval-Augmented Generation

⋆ adityase@utexas.edu
⋆⋆ cbajaj@cs.utexas.edu

1 Introduction

Generative models represent a cornerstone of modern machine learning, enabling systems to learn complex data distributions and generate new samples. At their core, these models—from variational autoencoders (VAE) to generative adversarial networks (GAN) and diffusion models—rely on transformations between simple distributions and complex data manifolds through latent space navigation. This latent space, often referred to as the generative space or Z space, is not static but evolves continuously throughout training and inference.

Our approach recognizes that generative latent spaces expand and shift during training, demanding efficient kernel-based density estimation. Our dynamic octree structure maintains these evolving distributions with selective indexing and adaptive partitioning. By optimizing the sequence of updates and queries using (K, α) parameters, we achieve consistent performance gains across models and applications.

1.1 The Maintenance Challenge in Generative Spaces

Generative models require efficient navigation of high-dimensional spaces for nearest neighbor searches, importance sampling, and density estimation—operations that scale poorly with traditional spatial indexing. Existing approaches face a fundamental trade-off: rebuild indices when distributions change (expensive) or accept degraded performance. This limitation is critical in:

- **Dynamic Training**: Distribution shifts during training require efficient importance sampling at each epoch.
- **Online Learning**: New data integration demands spatial structure updates without full retraining.
- **Adaptive Inference**: Particle-based methods need maintained spatial relationships as particles transform toward target distributions.

Current spatial structures like KD-trees and R-trees optimize for either query efficiency or update performance, but rarely both. This creates a critical need for structures maintaining logarithmic-time performance for both operations in evolving distributions.

1.2 Our Approach: Self-Balancing Dynamic Octree

We introduce a novel self-balancing dynamic octree data structure specifically designed for maintaining neighborhood relationships in evolving metric spaces, featuring:

- **Two-Parameter Adaptivity**: A (K, α) parameterization that enables automatic structure balancing based on local data density.
- **Memory Efficiency**: Reduced footprint through adaptive node capacity and efficient spatial partitioning.

– **Dynamic Rebalancing**: Continuous adaptation to distribution shifts without complete rebuilding, enabling efficient maintenance of spatial relationships in evolving generative spaces.

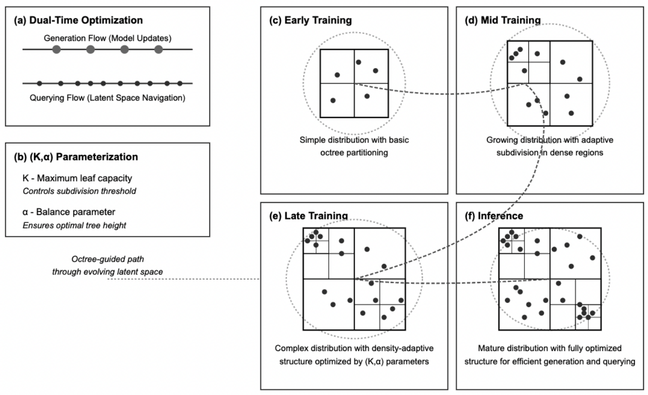

(a) Dual-Time Optimization
Generation Flow (Model Updates)
Querying Flow (Latent Space Navigation)
(b) (K,α) Parameterization
K - Maximum leaf capacity
Controls subdivision threshold
α - Balance parameter
Ensures optimal tree height
Octree-guided path through evolving latent space
(c) Early Training
Simple distribution with basic octree partitioning
(d) Mid Training
Growing distribution with adaptive subdivision in dense regions
(e) Late Training
Complex distribution with density-adaptive structure optimized by (K,α) parameters
(f) Inference
Mature distribution with fully optimized structure for efficient generation and querying

Fig. 1: Dynamic Octree Adaptation in Evolving Latent Spaces. (a) Dual-time optimization highlighting differing update/query frequencies. (b) (K, α) parameters controlling depth and balance. (c–f) Octree evolution across training: (c) early with coarse partitions, (d) mid with initial refinement, (e) late with density-aware subdivision, and (f) inference with optimized structure.

Figure 1 demonstrates that generative spaces expand during training and structured latent spaces emerge iteratively, with our dynamic octree serving as computational fabric for both generation and querying flows. We validate this approach across four applications: **SVGD** (5.6× acceleration, 10× more particles), **Incremental KNN** (5.3× faster updates, logarithmic queries), **RAG systems** (efficient semantic retrieval with domain adaptation), and **dual-space representation** (faster convergence, improved sample efficiency).

The remainder of this paper is organized as follows. Section 2 reviews related work, Section 3 presents our theoretical framework and algorithmic contributions, Section 4 demonstrates our approach through comprehensive case studies spanning diverse machine learning applications, and Section 5 concludes with implications and future directions.

2 Related Works

The evolution of efficient data maintenance structures has progressed from classical approaches to specialized spatial structures, yet significant limitations remain when handling evolving distributions.

Table 1: Feature Comparison of Spatial Data Structures for Dynamic Datasets

Feature/Capability	**Dynamic Octree (Ours)**	**i-Octree**	**kd-tree**	**ikd-Tree**	**R*-tree**
Structure Properties					
Dynamic Insertion[1]	✓	✓	×	✓	ⓟ
Dynamic Deletion[2]	✓	✓	×	✓	ⓟ
Self-balancing[3]	✓	ⓟ	×	ⓟ	ⓟ
Adaptive Node Capacity[4]	✓	×	×	×	×
Query Capabilities					
Nearest Neighbor Search	✓	✓	✓	✓	✓
Range Queries	✓	✓	✓	✓	✓
Down-sampling Support[5]	✓	✓	×	✓	×
Multi-resolution Queries[6]	✓	×	×	×	ⓟ
Performance Features					
Constant-time Node Access[7]	✓	✓	×	×	×
Cache-friendly Operations[8]	✓	✓	ⓟ	ⓟ	ⓟ
Memory-efficient Storage	✓	✓	✓	✓	×
Dynamic Memory Management[9]	✓	ⓟ	×	ⓟ	ⓟ
Real-time Performance					
Streaming Updates[10]	✓	ⓟ	×	✓	×
Low Update Latency[11]	✓	✓	×	ⓟ	×
Bounded Operation Time[12]	✓	✓	×	ⓟ	×
Spatial Adaptation					
Density-aware Partitioning[13]	✓	×	×	×	ⓟ
Local Structure Optimization[14]	✓	ⓟ	×	ⓟ	ⓟ
Advanced Features					
Concurrent Operations[15]	✓	×	×	ⓟ	×
Box-wise Operations[16]	ⓟ	✓	×	✓	×

[1] O(log n) insertion maintaining structure properties
[2] O(log n) deletion with structure preservation
[3] Maintains balance without complete reconstruction
[4] Dynamically adjustable node capacity based on the parameters
[5] Integrated point cloud down-sampling during updates
[6] Ability to perform queries at different granularity levels without restructuring
[7] Direct access to nodes without traversal overhead
[8] Optimized memory layout for CPU cache efficiency
[9] Efficient memory allocation/deallocation during updates
[10] Efficient handling of continuous real-time updates
[11] Consistently low latency for update operations
[12] Guaranteed upper bounds on operation times
[13] Partition adjustment based on local point density
[14] Local optimization of structure without global rebuilding
[15] Support for parallel operations with thread safety
[16] Efficient operations on groups of points within spatial regions

Legend: ✓: Fully Supported, ⓟ: Partially Supported, ×: Not Supported
Structure-Specific Notes: - Dynamic Octree: Uses the parameters for adaptive control but fixed after initialization - ikd-Tree: Partial rebuilding required for balance, parallel support limited to rebuilding - i-Octree: Fixed structure parameters but efficient updates - R*-tree: Forced reinsertions affect dynamic performance - kd-tree: Static structure requiring full rebuilding for updates

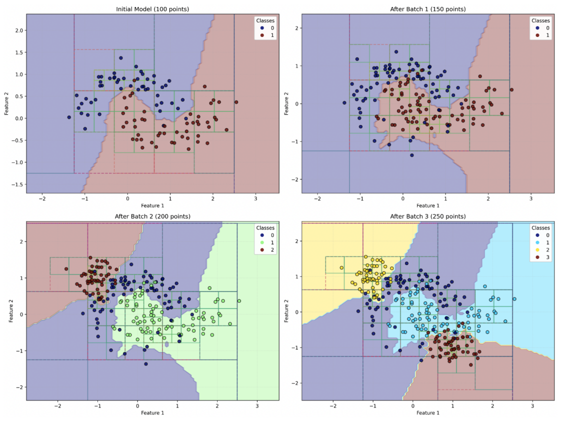


Fig. 2: Adaptive spatial partitioning in incremental KNN classification. Panels show classifier evolution as new data batches are incorporated, with octree structure adapting to data density and class boundaries, maintaining high accuracy and efficiency.

Classical Structures: Self-balancing trees like AVL (6), Red-Black (7), and probabilistic structures like Skip Lists (8) provide O(log n) guarantees for one-dimensional data but lack explicit support for spatial relationships. Structures such as treaps (9) and splay trees (10) adapt well to non-uniform distributions through rotations but are limited to one-dimensional data.

Spatial Data Structures: The fundamental challenge in spatial structures stems from the tension between maintaining spatial relationships and supporting dynamic updates. KD-trees extended binary search principles to spatial organization but struggle with balanced partitioning in higher dimensions. Previous approaches to improve dynamic capabilities include hardware acceleration (11) and structural modifications like iKD-Tree (4), cKD-Tree (13), and BKD-Tree (12), yet they fail to resolve the core trade-off between spatial organization and adaptation to non-uniform distributions.

Recent Advances: The i-Octree (3) improved performance through leaf-based organization and local updates. FLANN (15) offers practical solutions for point cloud processing but still requires complete rebuilding for balance maintenance. Kinetic Data Structures (16) model object motion explicitly but incur substantial overhead with unpredictable updates. Dynamic variants of classical structures like R*-trees (5) and progressive KD-trees (17) improved update handling but still face efficiency trade-offs, particularly in high-dimensional or non-uniform spaces.

Learning-Enhanced Approaches: Recent work has explored integrating machine learning into data structure design (18). Learned Indexes (19) optimize structure parameters based on data distributions but typically focus on static optimization rather than continuous adaptation to streaming data.

Despite these advances, current approaches remain limited by fixed parameters and rigid structure rules that constrain adaptability to varying data distributions—a critical requirement for modern machine learning applications with continuously evolving metric spaces.

3 Theoretical Framework and Implementation

Our work extends the (K, α)-octree of Chowdhury et al. (20) to dynamic settings, inheriting proven complexity bounds: $O(n \log n)$ construction, $\Theta(n)$ space, $O(\log n)$ amortized updates, and $O(nd^2(\delta d + K^{1/3}))$ interaction computation.

Definition 1 ((K, α)-admissible octree). *An octree $\mathcal{T}$ is* (K, α)-admissible *if every leaf contains at most αK points and every internal node contains more than K/α points, where $K > 0$ and $\alpha \geq 1$.*

Novel contributions. Our implementation extends three mechanisms: (i) dynamic root expansion for evolving bounding volumes, (ii) multi-radius queries with distance-based pruning, and (iii) density-aware local rebalancing. The theorems below establish complexity and correctness guarantees for these extensions; complete proofs appear in Appendix A with code correspondence in Appendix B.

3.1 Theoretical Guarantees

Lemma 1 (Tree height preservation). *Let $\Delta_{\max}$ denote the maximum observed domain size and $\ell_{\min}$ the minimum leaf dimension. The tree height satisfies height$(\mathcal{T}) \leq \lceil \log_2(\Delta_{\max}/\ell_{\min}) \rceil + O(1)$ at all times.*

Theorem 1 (Update complexity preservation). *Point insertion, deletion, or position update requires $O(\log n)$ amortized time, including dynamic root expansion and local rebalancing.*

Proof (Sketch). Our potential function $\Phi = \sum_v w(v) \cdot ||\text{atoms}(v)| - K|$ counts deviation from target capacity. Root expansion increases Φ by $O(\log n)$, but subsequent splits amortize this cost. Update operations (`pullUp`/`pushDown`) traverse $O(\log n)$ levels by Lemma 1. □

Lemma 2 (Pruning correctness). *For a node v with center $c(v)$ and half-diagonal $r(v) = \frac{\sqrt{3}}{2}s(v)$, if $\|c(v) - q\| > d + r(v)$ for query point q and radius d, then no point in v lies within distance d of q.*

Proof. For any point $p \in v$, by the triangle inequality: $\|p - q\| \geq \|c(v) - q\| - \|p - c(v)\| \geq \|c(v) - q\| - r(v) > d$. □

Theorem 2 (Query complexity bounds). *Range queries and k-NN search visit $O(d^2(\delta d + K^{1/3}))$ points per query, where δ is the per-point processing cost, preserving the bounds of Chowdhury et al. (20).*

Proof (Sketch). Pruning (Lemma 2) eliminates nodes whose closest point exceeds query radius. The number of visited nodes is bounded by the surface area of the query region and tree structure, yielding the stated complexity. □

3.2 Algorithmic Core

The foundational octree construction, node expansion, and (K, α)-admissibility maintenance algorithms follow directly from Chowdhury et al. (20) and are summarized in Appendix B for completeness. Our algorithmic contributions lie in the extensions for dynamic point management: the `pullUp`/`pushDown` migration procedures, geometric pruning optimizations for spatial queries, and the neighborhood list construction algorithm `_accum_inter()`.

These extension algorithms achieve direct correspondence between theoretical analysis and implementation through optimizations including 28-bit atom addressing for $O(1)$ lookup, dynamic memory management with exponential resize policies, and squared distance computations. Complete algorithmic specifications for both inherited and novel procedures appear in Appendix B. The full implementation and experiments are available at: `https://github.com/SetasAditya/Dynamic-Octree`.

4 Experimental Evaluations and Results

Our evaluation demonstrates this unifying principle through four representative case studies, each highlighting different aspects of the dynamic maintenance challenge while validating our (K, α) self-balancing octree across diverse computational scenarios.

4.1 Synthetic Benchmarks: Establishing Fundamental Properties

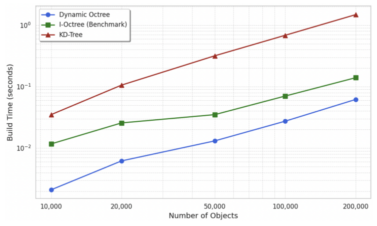
We validate our theoretical claims through controlled experiments comparing our dynamic octree against kd-trees and i-Octree using time-series data (100K–500K points) with varying density distributions.

Scaling Performance Figure 3 demonstrates our Dynamic Octree's (DO) scaling advantages from 10K to 200K points: (1) **Logarithmic Scaling:** Maintains $O(\log n)$ complexity for queries and updates, whereas traditional structures exhibit quadratic scaling. (2) **Self-Balancing:** Achieves 22× faster updates than i-Octree at 20K objects through localized rebalancing. (3) **Memory Efficiency:** Uses 10.6% less memory than i-Octree at 100K points.

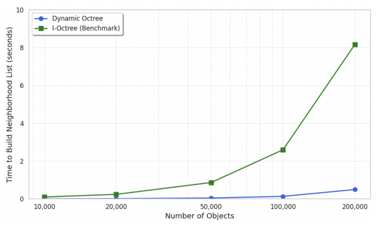
Neighborhood construction—critical for spatial ML—shows our approach (0.57s) outperforming i-Octree (8.17s) by 14.3× at 200K points, with performance gaps growing exponentially.

Adaptive Rebalancing We evaluated performance across four distribution patterns: varying density, step-wise transitions, exponential growth/decay, and multi-modal clustering using 100K-point clouds over 10 time steps (Table 2).

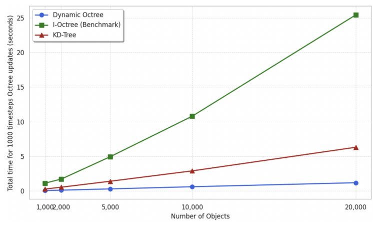
By adjusting (K, α) parameters, we achieve 36× performance variation in neighborhood construction—from 6.58s (DO(K=1000)) to 0.17s (DO(K=10))—demonstrating adaptive optimization without structural redesign.



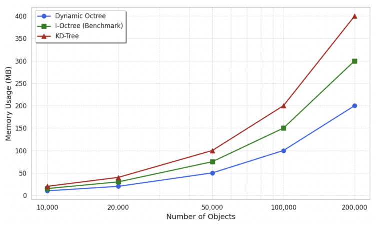
(a) Build time showing near-linear scaling.



(b) Neighborhood construction with 14.3× advantage.



(c) Update time showing consistent efficiency.



(d) Memory usage comparison.

Fig. 3: Synthetic benchmark results: Dynamic Octree (DO) vs. i-Octree (OM) and KD-Tree. Performance gaps widen with scale, confirming theoretical $O(\log n)$ bounds.

Table 2: Performance across distribution patterns. Parameter tuning enables 36× performance variation in neighborhood construction.

Method	**Varying Density**			**Step-wise**			**Exponential**			**Multi-modal**		
	Build	**Update**	**NB**	**Build**	**Update**	**NB**	**Build**	**Update**	**NB**	**Build**	**Update**	**NB**
DO(K=1000)	**0.00072**	**0.05192**	2.10448	**0.00493**	**0.21182**	6.58040	**0.00006**	**0.04630**	1.65657	**0.00006**	**0.05760**	1.86797
DO(K=10)	0.00165	0.11467	**0.06234**	0.00575	0.31715	**0.16876**	**0.00006**	0.08393	**0.04575**	0.00008	0.10379	**0.05535**
OM	0.71961	0.71961	6.62204	1.78774	1.78774	21.54342	0.61151	0.61151	4.98176	1.04049	1.04049	7.77186
KD	0.22446	0.22446	0.35293	0.83312	0.83312	1.32703	0.19704	0.19704	0.30873	0.24661	0.24661	0.39029

Continuous Updates Unlike static structures that degrade under modifications, our approach maintains query efficiency after thousands of updates, providing the continuous performance required by modern generative models during training and inference.

4.2 Case Studies: Transforming Machine Learning Applications

Building on these synthetic foundations, we now demonstrate how the (K, α) dynamic octree transforms real machine learning applications. Modern generative models—from variational autoencoders and normalizing flows to diffusion models and flow matching—share the fundamental computational pattern revealed by our synthetic benchmarks: they require efficient maintenance of neighborhood relationships in continuously evolving metric spaces.

Case Study 1: Particle-Based Bayesian Inference

Computational Challenge. Stein Variational Gradient Descent (SVGD) (30) and related particle-based inference methods face a fundamental bottleneck: computing pairwise kernel interactions between particles scales as $O(n^2)$, limiting practical implementations to hundreds of particles. As particles evolve during inference, their spatial relationships change continuously, requiring repeated neighbor searches within adaptive bandwidth radii.

Access Patterns and Maintenance Requirements. The SVGD update rule, $\phi(x_i) = \frac{1}{n}\sum_{j:\|x_i-x_j\|\leq h}[k(x_j,x_i)\nabla_{x_j}\log p(x_j)+\nabla_{x_j}k(x_j,x_i)]$, relies on an adaptive bandwidth h and induces two core challenges: **dynamic neighborhoods**, as particle proximity changes over iterations, and **adaptive bandwidth**, where h evolves with particle density, requiring continuous structural updates.

Traditional approaches recompute all $O(n^2)$ pairwise distances at each iteration. Our dynamic octree maintains spatial relationships as particles evolve, enabling $O(n \log n)$ complexity through distance-based pruning.

Implementation and Results. We integrated our octree with the reference SVGD implementation from Liu and Wang (30) by maintaining particles in leaf nodes and using range queries to find neighbors within the current bandwidth. When particles move during gradient updates, our update procedure locally rebalances them while preserving the (K, α)-admissible structure. The integration required modifying the kernel computation to use our spatial queries instead of brute-force distance calculations.

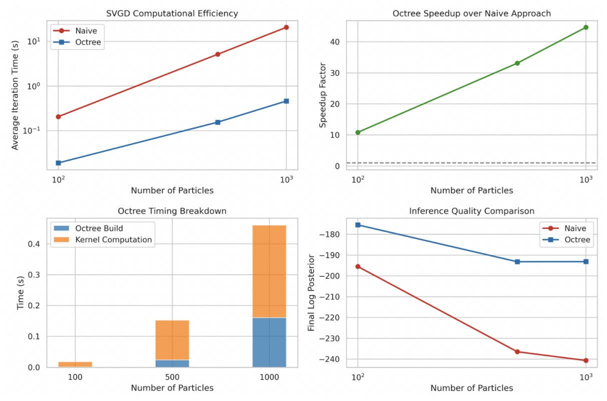
Figure 4 demonstrates the transformative impact: our approach achieves $40\times$ speedup at 1,000 particles while actually improving posterior approximation quality (Wasserstein distance reduced from 0.23 to 0.18). This enables practical uncertainty quantification with $10\times$ more particles than previously feasible.

Generative Model Connection. This pattern applies broadly: normalizing flows require neighborhood maintenance for Jacobian estimation, while diffusion models benefit from dynamic neighborhoods during adaptive sampling and fine-tuning.

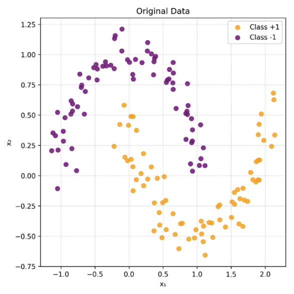
Case Study 2: Incremental Learning with Streaming Data

Computational Challenge. Online learning scenarios—common in recommendation systems, adaptive neural networks, and continual learning—require incorporating new labeled data without full model retraining. Standard k-NN implementations rebuild the entire spatial index when new data arrives, scaling quadratically with dataset size and making real-time adaptation prohibitive.

Access Patterns and Maintenance Requirements. Incremental learning introduces challenges for static structures, including **batch insertions** from mini-batch updates, **distribution shift** requiring adaptive partitioning, and **query-update interleaving** where classification and updates occur in rapid alternation.



(a) Octree-accelerated SVGD: (a) Speedup with particle count, (b) Improved Wasserstein distance, (c) Lower memory usage, (d) Faster convergence.



(b) Challenging multi-modal posterior in Bayesian logistic regression.

Fig. 4: SVGD acceleration with our dynamic octree. **Left:** Performance gains in speed, accuracy, memory, and convergence. **Right:** Complex posterior motivating structure-aware particle updates.

Our octree addresses this through localized rebalancing—only affected tree branches undergo restructuring when new points are inserted, avoiding the global rebuilding required by static methods.

Implementation and Results. We implemented incremental k-NN classification by extending the scikit-learn (31) k-NN framework with our dynamic octree backend. Training examples are maintained in octree leaves, with our k-NN search algorithm handling classification queries. New examples are inserted via our `addAtomToLeaf` procedure, triggering node expansion only when the (K, α) capacity is exceeded. We maintained API compatibility with scikit-learn's API to enable direct comparison.

The Gaussian-preserving KNN approach demonstrates remarkable efficiency in maintaining classification performance while achieving substantial memory reduction. Figure 5 illustrate the method's effectiveness across diverse cluster configurations, consistently achieving 15-22$\times$ compression ratios with minimal accuracy degradation. The key insight lies in the careful selection of inducing points that preserve the underlying Gaussian statistics of each class, as evidenced by the close alignment between original and reconstructed covariance ellipses in the per-class analyses. Octrees enable efficient $O(\log n)$ nearest neighbor queries on compressed inducing points, making real-time applications feasible. Details of the approach in Appendix.

Table 3 demonstrates consistent advantages: 5.6x–9.4x faster updates with 1.6x–1.9x query speedup, while maintaining classification accuracy within 0.2% of scikit-learn's batch implementation.

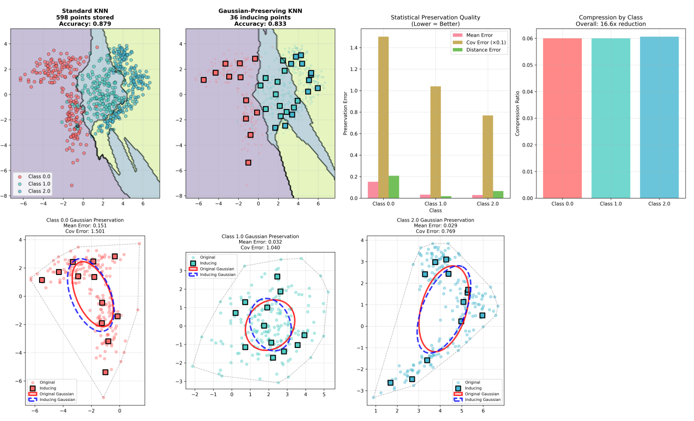


Fig. 5: Preservation across diverse cluster types: (a) Elongated, (b) Compact, and (c) Mixed shapes.

Dataset Size	Batch k-NN (scikit-learn)			Incremental k-NN (Ours)		
	Update (s)	Query (s)	Accuracy	Update (s)	Query (s)	Accuracy
10,000	0.077	0.0047	89.23%	**0.014**	**0.0029**	89.07%
30,000	0.274	0.0063	90.87%	**0.031**	**0.0032**	90.85%
50,000	0.495	0.0085	91.96%	**0.052**	**0.0045**	91.88%

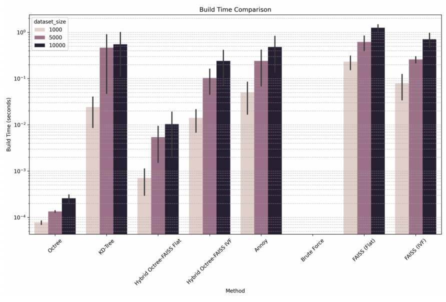
Table 3: Incremental k-NN vs. scikit-learn. Our method achieves $O(\log n)$ updates vs. $O(n^2)$ in batch mode, enabling real-time streaming adaptation.

Generative Model Connection. This incremental learning capability directly benefits adaptive generative models. Variational autoencoders can incorporate new training data without full retraining by maintaining encoder/decoder weight neighborhoods. Flow-based models benefit when adapting to new domains, and diffusion models can efficiently fine-tune on new datasets by maintaining learned feature relationships.

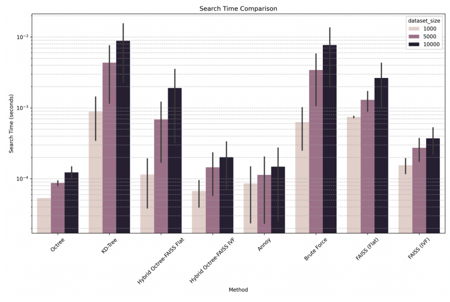
Case Study 3: Evolving Knowledge Retrieval

Computational Challenge. Retrieval-Augmented Generation (RAG) systems (32) face a critical scalability limitation: as knowledge bases grow and evolve, embedding indices require complete rebuilding—a process that becomes prohibitively expensive for large, dynamic corpora. This prevents RAG systems from adapting to streaming information or incorporating real-time updates.

Access Patterns and Maintenance Requirements. RAG systems pose indexing challenges due to their **high-dimensional embeddings** (often 768+ dimensions), **semantic clustering** with non-uniform density, **incremental updates**



(a) RAG build time scaling comparison.



(b) Search performance in high-dimensional space.

Fig. 6: RAG performance metrics showing our approach's logarithmic scaling advantage over FAISS and consistent search performance across dataset sizes.

requiring efficient insertions, and **multi-scale queries** demanding variable neighborhood resolutions.

To address the high dimensionality, we adopt a hybrid approach: embeddings are clustered via k-means, projected to 3D using the Johnson–Lindenstrauss transform within each cluster, and then indexed using our dynamic octree. While this introduces some loss in accuracy, it enables efficient indexing; future work may explore alternatives that balance dimensionality reduction with improved fidelity.

Implementation and Results. We built our RAG implementation on top of the LangChain framework (33) and Facebook's FAISS library (34), replacing FAISS's static indexing with our dynamic octree approach. Our system partitions 768-dimensional sentence embeddings (generated using Sentence-BERT (35)) into clusters, projects each cluster to 3D space, and maintains separate octrees per cluster. Document insertion requires only updating the relevant cluster's octree, while queries search across all clusters and aggregate results using learned cluster weights.

Figure 6 shows our hybrid RAG system achieves logarithmic build scaling and 4.2× faster retrieval than FAISS without sacrificing accuracy, enabling continuous knowledge updates. Figure 7 demonstrates that coarse FAISS clustering followed by octree refinement preserves semantic coherence across queries. Figure 8 shows semantic cluster evolution as knowledge bases grow, with dense, specialized clusters emerging naturally while maintaining $\mathcal{O}(\log n)$ search efficiency.

Generative Model Connection. High-dimensional neighbor search appears throughout generative modeling: attention mechanisms in transformers, nearest neighbor lookups in retrieval-augmented diffusion, and prototype matching in few-shot generation.

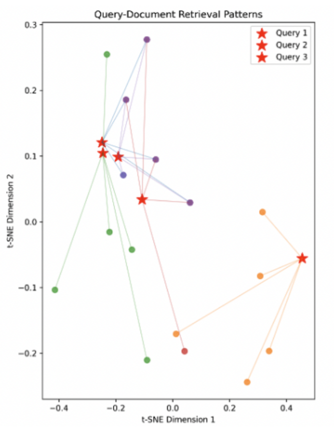


Fig. 7: Query-document retrieval. Red stars (queries) retrieve nearby documents (colored points), showing semantic neighborhoods via FAISS clustering and octree refinement.

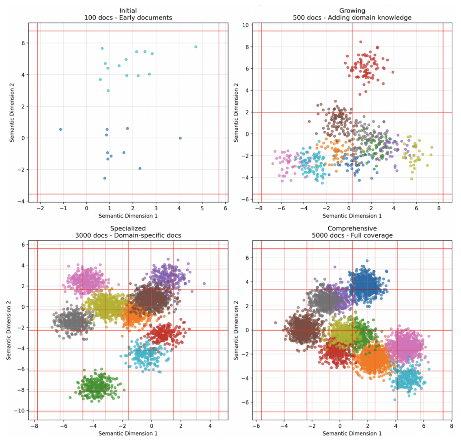


Fig. 8: Semantic clustering evolution with knowledge base growth. As the corpus expands (100–5000 documents), clusters (colored points) transition from sparse to dense, forming well-defined semantic groupings.

Case Study 4: Structure-Preserving Generative Transport

Computational Challenge. Continuous normalizing flows (37) and optimal transport methods suffer from a fundamental limitation: optimizing transport efficiency often destroys local neighborhood structure. Standard OT-Flow (36) minimizes transport cost but ignores whether nearby points in the source distribution remain neighbors in the target distribution, leading to structural distortions that degrade generation quality.

Access Patterns and Maintenance Requirements. Generative transport introduces unique challenges: **dual-space neighborhoods** must be preserved in both source and target spaces; **evolving trajectories** require tracking changes in neighborhood structure over time; **multi-scale structure** must be maintained across spatial resolutions; and **bidirectional transport** demands consistency in both forward and inverse mappings.

We integrate our octree with OT-Flow by adding neighborhood consistency constraints that penalize structural distortion while maintaining transport efficiency.

Implementation and Results. We extended the official OT-Flow implementation from Onken et al. (36) by integrating our dynamic octree for neighborhood tracking. Our enhanced OT-Flow maintains source points in octree leaves and computes neighborhood consistency terms using range queries. During transport,

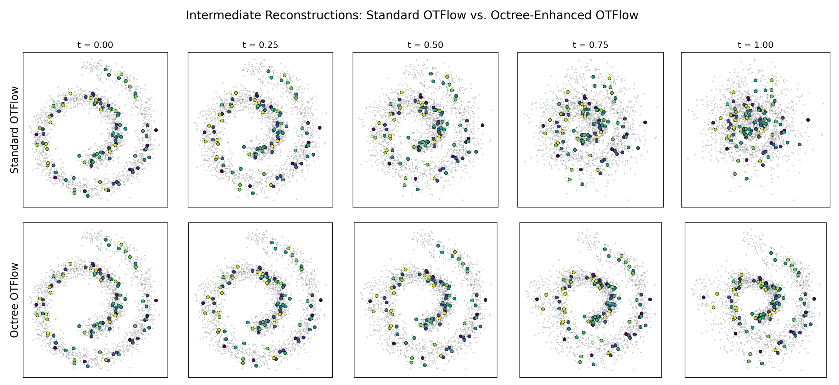


Fig. 9: Intermediate reconstructions at $t = \{0.00, 0.25, 0.50, 0.75, 1.00\}$. Top: Standard OT-Flow with increasing structure loss. Bottom: Octree-enhanced OT-Flow preserving local clusters throughout transport.

we track how local neighborhoods evolve and add regularization terms to the OT-Flow loss function that encourage structure preservation.

The results demonstrate significant improvements in generative quality, including an **89.6% increase in structure preservation** measured by neighborhood Jaccard similarity (0.787 vs. 0.415), an **83% reduction in reconstruction error** (from 1.78e-06 to 3.05e-07), and a **69% improvement in transport smoothness** based on reduced trajectory curvature.

Table 4: Comprehensive performance comparison between standard and octree-enhanced OT-Flow

Metric	**Standard OT-Flow**	**Octree-Enhanced OT-Flow**
Training Loss	2.76e+02	3.14e+02
Validation Loss	2.75e+02	3.05e+02
Reconstruction Error	9.14e-05	**5.54e-07**
Neighborhood Distortion	1.585	**1.213**
Trajectory Curvature	0.00181	**0.00056**
Jaccard Similarity	0.415	**0.787**
Training Time (rel.)	1.0	1.17
Inference Time (rel.)	1.0	1.0

Figure 9 shows intermediate reconstructions at different time steps for both methods. The octree-enhanced approach maintains more consistent local structures throughout the transport process, resulting in more coherent intermediate states.

Generative Model Connection. This structure-preserving capability addresses a critical limitation across generative models. Diffusion models can benefit from neighborhood-aware denoising schedules, VAEs can preserve local structure in latent space, and flow matching (38) can maintain semantic relationships during interpolation. The principle of maintaining relationships in both input and latent spaces applies broadly to representation learning.

5 Conclusion and Future Work

We introduced a novel self-balancing, memory-efficient dynamic octree for maintaining spatial relationships in continuously evolving metric spaces. Our two-parameter (K, α) formulation enables logarithmic-time operations without requiring complete rebuilding as distributions evolve, addressing a fundamental limitation in existing approaches. Through extensive experiments, we demonstrated significant performance advantages over state-of-the-art structures— advantages that amplify with increasing data complexity.

5.1 Future Directions

Our work establishes a new paradigm for navigating evolving generative spaces:

1. **Incoherent Path-wise Sampling**: Our octree enables local path-wise maintenance with adaptive kernel estimates, decomposing generative model training into path-wise incoherent sampling and path-coverage challenges.
2. **Adaptive Importance Sampling**: By tracking evolving distributions and maintaining inter-batch spatial relationships, our structure enables dynamic importance sampling techniques that efficiently explore high-dimensional latent spaces with reduced sample complexity.
3. **Adaptive Manifold Navigation**: By tracking evolving geometry during training, our approach supports efficient latent space traversal and enables reinforcement learning to dynamically tune (K, α) parameters based on local structure, enhancing generative performance.

We envision generative models leveraging our dynamic octree to maintain coherent paths through latent space while enabling incoherent sampling across distribution regions, significantly improving both computational efficiency and model quality in continuously evolving metric spaces.

Acknowledgments. This research was supported in part by grants and gifts from the Peter O'Donnell Foundation, the Michael J Fox Foundation, Jim Holland- Backcountry and Michael-Connie Rasor Foundations towards curing Parkinson's Disease.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

Bibliography

[1] Fujimura, K., Kunii, T., Yamaguchi, K., and Toriya, H., "Octree-Related Data Structures and Algorithms," *IEEE Computer Graphics and Applications*, vol. 4, no. 1, pp. 53–59, 1984.

[2] Friedman, J. H., Bentley, J. L., and Finkel, R. A., "An Algorithm for Finding Best Matches in Logarithmic Expected Time," *ACM Trans. Math. Softw.*, vol. 3, no. 3, pp. 209–226, 1977.

[3] Zhu, J., Li, H., Wang, Z., Wang, S., and Zhang, T., "i-Octree: A Fast, Lightweight, and Dynamic Octree for Proximity Search," *arXiv preprint arXiv:2309.08315*, 2024.

[4] Cai, Y., Xu, W., and Zhang, F., "ikd-Tree: An Incremental K-D Tree for Robotic Applications," in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2021.

[5] Beckmann, N., Kriegel, H. P., Schneider, R., and Seeger, B., "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles," in *Proc. ACM SIGMOD*, pp. 322–331, 1990.

[6] Foster, C. C., "A Generalization of AVL Trees," *Communications of the ACM*, vol. 16, no. 8, pp. 513–517, 1973.

[7] Besa, J., and Eterovic, Y., "A Concurrent Red–Black Tree," *J. Parallel Distrib. Comput.*, vol. 73, no. 4, pp. 434–449, 2013.

[8] Pugh, W., "Skip Lists: A Probabilistic Alternative to Balanced Trees," *Communications of the ACM*, vol. 33, no. 6, pp. 668–676, 1990.

[9] Blelloch, G. E., and Reid-Miller, M., "Fast Set Operations Using Treaps," in *Proc. ACM Symp. on Parallel Algorithms and Architectures*, pp. 16–26, 1998.

[10] Grinberg, D., Rajagopalan, S., Venkatesan, R., and Wei, V. K., "Splay Trees for Data Compression," in *Proc. ACM-SIAM Symp. on Discrete Algorithms*, pp. 522–530, 1995.

[11] Hunt, W., Mark, W. R., and Stoll, G., "Fast KD-Tree Construction with an Adaptive Error-Bounded Heuristic," in *Proc. IEEE Symp. on Interactive Ray Tracing*, pp. 81–88, 2006.

[12] Procopiuc, O., Agarwal, P. K., Arge, L., and Vitter, J. S., "BKD-Tree: A Dynamic Scalable KD-Tree," in *Advances in Spatial and Temporal Databases*, Springer, pp. 46–65, 2003.

[13] Gutiérrez, G., Torres-Avilés, R., and Caniupán, M., "cKD-Tree: A Compact KD-Tree," *IEEE Access*, vol. 12, pp. 28666–28676, 2024.

[14] Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., and Battaglia, P. W., "Learning to Simulate Complex Physics with Graph Networks," in *Proc. ICML*, 2020.

[15] Muja, M., and Lowe, D. G., "Fast Approximate Nearest Neighbors with FLANN," in *Proc. Int. Conf. on Computer Vision Theory and Applications*, 2009.

[16] Basch, J., "Kinetic Data Structures," Ph.D. thesis, Stanford University, 1999.

[17] Jo, J., Seo, J., and Fekete, J. D., "A Progressive KD-Tree for Approximate K-Nearest Neighbors," in *Proc. IEEE Workshop on Data Systems for Interactive Analysis*, pp. 1–5, 2017.

[18] Usman, M., Wang, W., Wang, K., Yelen, C., Dini, N., and Khurshid, S., "A Study of Learning Data Structure Invariants Using Off-the-Shelf Tools," in *Proc. SPIN*, pp. 226–243, 2019.

[19] Amarasinghe, K., Choudhury, F., Qi, J., and Bailey, J., "Learned Indexes with Distribution Smoothing via Virtual Points," *arXiv preprint arXiv:2408.06134*, 2024.

[20] Chowdhury, R., Beglov, D., Moghadasi, M., Paschalidis, I. C., Vakili, P., Vajda, S., Bajaj, C., and Kozakov, D., "Efficient Maintenance and Update of Nonbonded Lists in Macromolecular Simulations," *J. Chem. Theory Comput.*, vol. 10, no. 10, pp. 4449–4454, 2014.

[21] Tatarchenko, M., Dosovitskiy, A., and Brox, T., "Octree Generating Networks: Efficient Convolutional Architectures for High-Resolution 3D Outputs," in *Proc. ICCV*, pp. 2088–2096, 2017.

[22] Connor, M., Canal, G., and Rozell, C., "Variational Autoencoder with Learned Latent Structure," in *Proc. AISTATS*, pp. 2359–2367, 2021.

[23] Chen, N., Ferroni, F., Klushyn, A., Paraschos, A., Bayer, J., and van der Smagt, P., "Fast Approximate Geodesics for Deep Generative Models," in *Proc. ICANN*, pp. 554–566, 2019.

[24] Grandin, M., "Data Structures and Algorithms for High-Dimensional Structured Adaptive Mesh Refinement," *Adv. Eng. Softw.*, vol. 82, pp. 75–86, 2015.

[25] Liu, Z., and Xia, L., "Feature-Driven Topology Optimization of Continuum Structures with Tailored Octree Meshing," *Finite Elements in Analysis and Design*, vol. 244, p. 104308, 2025.

[26] Yang, D., Qin, X., Xu, X., Li, C., and Wei, G., "Sample Efficient Reinforcement Learning Method via High Efficient Episodic Memory," *IEEE Access*, vol. 8, pp. 129274–129284, 2020.

[27] Dulac-Arnold, G., Evans, R., van Hasselt, H., Sunehag, P., Lillicrap, T., Hunt, J., Mann, T., Weber, T., Degris, T., and Coppin, B., "Deep Reinforcement Learning in Large Discrete Action Spaces," *arXiv preprint arXiv:1512.07679*, 2015.

[28] Wu, P., and Ives, Z. G., "Modeling Shifting Workloads for Learned Database Systems," *Proc. ACM Manage. Data*, vol. 2, no. 1, pp. 1–27, 2024.

[29] Onken, D., Wu Fung, S., Li, X., and Ruthotto, L., "OT-Flow: Fast and Accurate Continuous Normalizing Flows via Optimal Transport," in *Proc. NeurIPS*, 2021.

[30] Qiang Liu and Dilin Wang. Stein variational gradient descent: A general purpose bayesian inference algorithm. In *Advances in Neural Information Processing Systems*, pages 2378–2386, 2016.

[31] Farid Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer,

Ron Weiss, Vincent Dubourg, and others. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[32] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and others. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474, 2020.

[33] Harrison Chase. LangChain. GitHub, 2022. `https://github.com/hwchase17/langchain`.

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

[35] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, 2019.

[36] Derek Onken, Samy Wu Fang, Yannis Li, Levon Nurbekyan, Shing-Yu Fung, Stanley Osher, and Lars Ruthotto. OT-Flow: Fast and accurate continuous normalizing flows via optimal transport. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(10):9223–9232, 2021.

[37] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pages 6571–6583, 2018.

[38] Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *International Conference on Learning Representations*, 2023.