Research Agenda

Vision. The breakthroughs in Deep Learning [1] enable unstructured data (e.g., images, videos, and audio) [2] [3] to be represented as high-dimensional feature vectors – typically referred to as *embeddings* – for serving a wide range of AI applications [4] [5] [6] [7]. Recent advancements in Large Language Models (LLMs) [8] [9] have catalyzed the emergence of a new generation of AI applications that rely on these embeddings. Consider generative AI for multimedia at a large animation studio as a concrete example. It is not uncommon for such studios to store hundreds of millions of multimedia files across hundreds of terabytes of storage [10]. Leveraging LLMs to generate new multimedia (e.g., new animation frames or video snippets) based on text descriptions (e.g., "Robots in Paris, France in front of the Eiffel Tower") would involve (Figure 1):

i. generating embeddings for each of the hundreds of millions of multimedia files ahead of time, capturing the core attributes of the media (e.g., "robot," "pets," "cars," ... "Paris," "Rome," "Athens," etc.)
ii. projecting the text description for the multimedia (i.e., "Robots in Paris, France in front of the Eiffel Tower") to the embedding space, i.e., a high-dimensional feature vector,
iii. isolating the multimedia files whose embeddings are "close to" the feature vector corresponding to the text description (e.g., images featuring "robot," "Paris," or "Eiffel Tower"), and
iv. feeding these multimedia files and their embeddings to the generative AI model to create a new image.

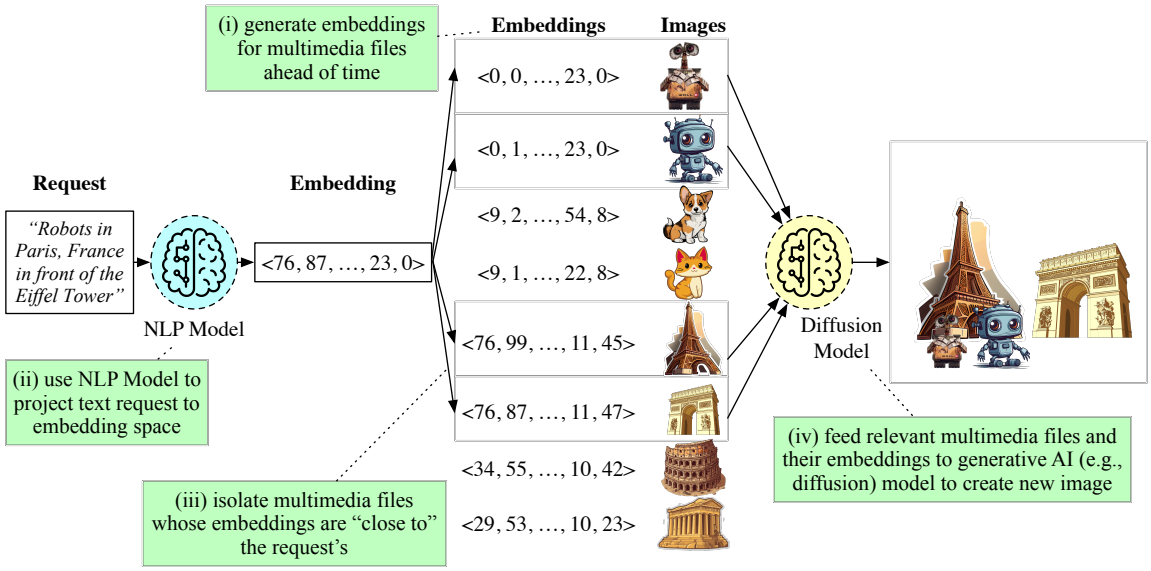

Figure 1. Example outlining how an animation studio may use embeddings along with generative AI to create new animation frames.

Another common use case for embeddings is text-based AI applications. Here, semantically similar sentences should have similar embedded vectors and thus be closer together in the space. A lot of useful tasks can be expressed in terms of text similarity, including semantic search ("How similar is a query to a document in your database?"), spam filtering ("How close is an email to examples of spam?"), content moderation ("How close is a social media message to known examples of abuse?"), or a conversational agent ("Which examples of known intents are closest to the user's message?"). Unsurprisingly, most of the current Retrieval-Augmented Generation (RAG) systems – the bedrock of Large Language Models (LLMs) – are text-based.

As these examples show, a critical component of such AI applications is the efficient analysis of many embeddings. Vector databases and indexing engines [11] [12] [13] provide persistence and long-term memory for these embeddings; indeed, vector query processing on such databases powers many AI applications, ranging from recommendation [4] [5] and recognition [6] to biological information retrieval [7].

However, the needs of emerging AI applications impose new requirements for the vector storage and indexing stack used for catering to such embeddings:

- **Handling massive data volumes and large data dimensionality.** As noted in the example above, the sheer number of embeddings that must be stored and analyze can be quite large, depending on the type of data. For instance, while a 16MB image might generate a single embedding vector, text embeddings are often generated for groups of 4-5 words via embedding models like word2vec [14], resulting in an embedding for every 100B! Moreover, higher match accuracy for AI models is achieved at large vector dimensionalities for these embeddings, e.g., 768 32-bit dimensions in Google's Gemini model [15] and 1536 32-bit dimensions in OpenAI's **`text-embedding-ada-0021`** model [16]. This amounts to over a terabyte of embeddings for an image library in the animation studio example outlined above – or for a single billion-word text document! Storing such a large amount of vector data requires efficient layouts encompassing both memory and persistent storage and space-efficient representations that can support efficient search across the many dimensions, as we outline next.
- **Low-latency search on high dimensionality data.** Searching across large volumes of embedding data differs from traditional database search – it involves finding embedding vectors that are "close to" a searched embedding vector in the high-dimensional latent space. The proximity of vectors is often captured by Euclidean distance [17]; as such, searching over embeddings for a specific vector effectively identifies all embeddings within a fixed Euclidean distance from the searched vector. The typical search latency SLOs for most AI applications is within a few hundred milliseconds, necessitating multi-dimensional index structures that can speed up searches. Unfortunately, traditional index structures further increase the amount of storage required by several orders of magnitude, introducing an unfavorable tradeoff between search and storage efficiency.
- **Embracing heterogeneity in processing architectures.** While most vector storage and analysis approaches are realized on CPUs, in most AI applications, these embeddings are utilized in accelerators such as GPUs and TPUs – as a part of model training or inference serving. As such, in latency-sensitive use cases, ensuring the vector storage stack (e.g., indexes) can be cached and analyzed in these accelerators can improve application performance significantly. Moreover, GPUs provide ample opportunities for leveraging massive data parallelism to speed up multi-dimensional searches, although few (if any) algorithms have been explored. Again, a critical impediment is storage efficiency – accelerators like GPUs are often equipped with high bandwidth memory to sustain their parallelism, which has limited capacity and makes support for multi-dimensional indexes even more challenging.

Unfortunately, existing vector storage and indexing stacks fall short of these requirements along multiple dimensions. Systems that support efficient query semantics across multiple vector dimensions leverage additional index structures with large storage overheads [18] [19]. While such systems perform well for small datasets, they cannot scale to larger datasets as the underlying data structure easily outgrows the memory capacity — e.g., indexes alone can consume 55% of the total memory in real-world in-memory databases [20]. Another emerging class of systems leverages queries on compressed in-memory data structures [21] [22] [23] [24] to scale to larger datasets, but lack support for efficient multi-dimensional search queries required for embeddings. Finally, most storage systems and indexes specifically cater to traditional CPUs; while recent efforts have explored porting simple index structures like hash tables [25] and tries [26] to GPUs, supporting space- and multi-dimensional search-efficient data structures that are efficient to query via both CPUs and GPUs remains an open problem.

Our approach to address these requirements for high-dimensionality vector storage and analysis at massive scales will be to develop novel compressed self-indexes for embeddings that leverage heterogeneous processors to support efficient multi-dimensional search semantics. A self-index refers to a data representation that combines both the raw data (i.e., embeddings) and the indexes that facilitate search (across the various dimensions) into a single data structure, while compression allows for the storage footprint of this data structure to be lower than the raw data itself. We will explore the design of a wide range of novel index structures, ranging from in-memory succinct tries and finite state automata (FSA) to their distributed, persistent, and fusion-based variants. Such an approach will be key to scaling to larger scales and higher dimensionalities that embeddings are veering towards. Moreover, we will also develop novel search algorithms that support low-latency, high-dimensional searches across embeddings encoded within our envisioned compressed self-index across heterogeneous compute units, i.e., both CPUs and GPUs. The success of our

project will drive significant innovations in the emerging AI landscape by opening them to massive data volumes previously inaccessible due to the lack of space and search efficiency.

PI Experience. PI Khandelwal is an expert on data structure, algorithm design, and distributed storage systems. His work addresses challenges in processing, storing, and serving large volumes of data to empower real-world systems --- from sprawling internet services like social media to mission-critical tools in health and medicine. His research formulates such challenges as algorithm and data structure design problems, resulting in practical systems backed by strong theoretical guarantees that have found their way into production. His prior research has resulted in practical distributed cloud storage [27] [28] [29] [30] and compute [31] [32] stacks, including storage systems that support queries directly on compressed data [21] [33] [22] [23] and secure data stores [34] [35] [36].

Preliminary Work. Our initial efforts towards realizing this vision have already taken important strides through our collaboration with Kiran Srinivasan at NetApp. This collaboration has resulted in the design and implementation of Trinity, a system that simultaneously facilitates query and storage efficiency across large volumes of multi-dimensional data (i.e., vectors). Trinity accomplishes this through a new dynamic, succinct, multi-dimensional data structure called MDTrie. MDTrie employs a combination of novel Morton code generalization, a multi-dimensional search algorithm, and a self-indexed trie structure to achieve the above goals. Our evaluation of Trinity for real-world use cases shows that compared to state-of-the-art systems, it supports 7.2 - 59.6× faster multi-dimensional searches and storage footprint comparable to OLAP columnar stores and 4.8 - 15.1× lower than NoSQL stores and OLTP databases. Our work [33] has already been published at EuroSys'24, where it received the Best Student Paper Award.

However, Trinity still falls short of several key requirements for vector storage and indexing stack outlined above. First, it lacks support for significantly higher dimensionality data – Trinity currently supports tens of dimensions, while emerging AI applications require storage and analysis across hundreds to thousands of dimensions. Moreover, Trinity's multi-dimensional search algorithm focuses on *range queries*, i.e., finding all vectors that lie within two extremes for each dimension (e.g., all vectors whose first dimension lies between 24 and 99, second dimension lies between 32 and 44, third dimension lies between …, etc.). Finally, Trinity does not support search or compression on various AI accelerators. We outline our agenda for meeting these requirements to fulfill our vision next.

Research Agenda. We will realize our vision by addressing the following inter-related research questions:

How can we handle larger data volumes for embeddings?

The first research question we seek to answer is how we can improve space efficiency and data representation for encoding terabytes of embedding data. As noted earlier, the amount of embedding data can easily spill over to terabytes, and even Trinity's compact in-memory representation can fail to manage the data efficiently in memory. We plan to explore this research question in two orthogonal directions.

First, we will exploit data representations that further reduce the storage footprint of multi-dimensional data by leveraging data structures that are even more compact than tries while still supporting the same functionality – specifically, compressed and query-efficient finite state automata (FSA) [37] [38]. These data structures would permit the encoding of multi-dimensional data as succinct state machines, which improve on reducing redundancy in trie-based index structure in two key ways: (i) by encoding the index as a graph instead of a tree, states that are reused throughout the graph are encoded as a single node with multiple pointers to them, whereas tree structure would require a creating a copy of the node to avoid loops, and (ii), by using "recurrent" states – where a node has an edge to itself – it can reduce long redundant chains of repeated nodes common in tree-based indexes. We will also explore novel *fusion* encodings [39] for both succinct tries and FSAs, which essentially permit constant time search across keys stored in a node by compressing (typically referred to as *sketching*) the keys so that all can fit into a single machine word, which in turn allows search to be done in parallel.

Second, we will develop novel on-disk data representations of our fusion tries and FSAs and develop efficient external search algorithms that minimize disk IO and search latency. These external search algorithms will operate with tiered in-memory algorithms to permit scaling to terabytes of embedding data without compromising performance.

How can we handle larger vector dimensionalities for embeddings?

The use of space-filling curves permits packing multiple dimensions into a single value in Trinity, permitting the use of a compressed trie structure to search for ranges within these dimensions as if searching for ranges over a single dimension. Specifically, the MDTrie in Trinity encodes *d* bits of this "coalesced" dimension at each level of the trie, where *d* is the total number of dimensions; this permits searching through one bit of each dimension at each level of the trie. This approach can even be extended to succinct FSA data structures, as well as their fusion variants, as outlined in our first research question. However, embedding many dimensions into a single value and indexing it in a trie or FSA data structure imposes scalability restrictions. For instance, the size of each node in the Trinity's MDTrie grows exponentially with the number of dimensions, i.e., $O(2^d)$. While this is acceptable for tens of dimensions with aggressive per-node bit compression (e.g., 15-20 dimensions result in a few kilobytes per node), this approach fundamentally cannot scale to hundreds or thousands of dimensions required for embeddings in modern AI use cases. As such, the second research question we aim to address is how we can scale to much larger embeddings in such cases.

Our key insight in addressing this challenge lies in the observation at extremely high dimensionality, the amount of information in each node is quite sparse, i.e., most of the bits in each node are likely to be zeros. As such, it would be possible to take a *hierarchical* approach to encoding these nodes to scale to high dimensionality – for instance, every *m* bits in the encoded node could have an associated flag bit, which marks if the *m* bits actually contain useful information or are empty (i.e., contain only zeros). While this example highlights a two-level hierarchy, this idea can be extended to *k* levels, where the flag at the i^{th} level indicates if the corresponding *m* bits at the $(i+1)^{th}$ level contain any useful information or not (see Figure 2 for a 2-level representation). This can reduce the space encoding per node from exponential to linear in the number of dimensions, with a worst-case linear addition in space complexity due to the "flag" bits. Fortunately, the flag bits are likely to be compressible and could facilitate efficient search by eliminating looking through node chunks with no useful information (i.e., where the flag bit is set to 0).

Figure 2. Example of hierarchical encoding to scale to higher dimensionality; in this example, the number of bits required to encode the node reduces from 32 to 14.

How can we support multi-dimensional search at low latency?

As noted in our preliminary work, Trinity – along with most tree-based multi-dimensional indexes – focus on per-dimension range queries, rather than multi-dimensional proximity searches required for embeddings in AI applications. Existing approaches to support proximity searches in Euclidean space on multi-dimensional tree structures (e.g., KD-Trees [40]), on the other hand, incur high storage overheads and search latencies since they must index and navigate a large number of dimensions to isolate vectors of interest. In contrast, approximate nearest neighbor (ANN) algorithms (e.g., Inverted Multi-Index [41] and Hierarchical Navigable Small World [42]) focus on proximity based search, but tend to large indexes, and even so, often identify "close" vectors with high margins of error.

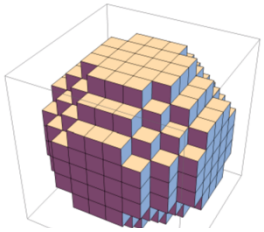

Figure 3. Approximating a hypersphere using multiple hypercubes.

Our approach of encoding multiple dimensions into a single dimension using space-filling curves offers a fundamentally new approach to addressing this challenge, and provides the potential of achieving space- and performance-efficiency, while providing near-perfect accuracy. This relies on our observation that space-filling curves like Morton encoding make it easy to find relevant vectors within well-defined *hypercubes* in n-dimensional space through simple geometric manipulation. We plan to adapt these manipulations to run efficiently on our compressed trie-based and FSA-based representation of Morton-encoded vectors. While proximity searches rely on identifying vectors within a fixed *hypersphere* in n-dimensional space, the same set of vectors can be identified by aggregating the vectors

across several hypercubes that the hypersphere encompasses (see Figure 3). An added benefit of this approach is that each of the smaller hypercubes can be searched independently in parallel, permitting a far more scalable search algorithm.

How can we adapt our search algorithm for GPU acceleration?

Our final research thrust will explore adapting search algorithms to AI accelerators, specifically GPUs. Since most of the AI workloads are run on GPUs, placing the embedding data and searching on it close to or at the GPU is critical for low-latency and high-throughput. While recent efforts have explored porting simple index structures to GPUs [25] [26], none have explored space- and multi-dimensional search-efficient data structures to the best of our knowledge. We will leverage two key insights in facilitating efficient multi-dimensional proximity search algorithms on GPUs. First, our initial efforts in developing Trinity showed that many of the bit compression techniques employed in MDTries benefit significantly from data parallelism – the use of AVX512 instructions improved end-to-end compression and decompression speeds by over 2 - 3× in our experiments. GPUs offer such data parallelism at a much larger scales, with thousands of data parallel cores; as such, we believe they can improve the compression/decompression speeds – and consequently, search speeds – by an order of magnitude or more by carefully designing the corresponding algorithms to leverage data parallelism. Second, we also plan to exploit the inherent parallelism in our search algorithm as well – as we already noted above, decomposing the search in n-dimensional space into a large number of parallel hypercube searches creates significant opportunities for parallelism; we will investigate how these approaches can be realized efficiently on GPUs. Finally, we note that while we will develop novel search algorithms for the GPU, the underlying data representation will remain the same across both CPUs and GPUs, facilitating hybrid searches that exploit all available compute resources.

References Cited

[1] Y. LeCun, Y. Bengio and G. Hinton, "Deep Learning," *Nature,* 2015.

[2] R. Blumberg and S. Atre, "The problem with unstructured data," *DM Review,* 2003.

[3] "Eighty Percent of Your Data Will Be Unstructured in Five Years," 2023. [Online]. Available: https://solutionsreview.com/data-management/.

[4] M. D. A. G. a. S. R. A. S. Das, "Google news personalization: scalable online collaborative filtering," in *WWW*, 2007.

[5] M. Grbovic and H. Cheng, "Real-time personalization using embeddings for search ranking at airbnbM. Grbovic and H. Cheng," in *SIGKDD*, 2018.

[6] F. Schroff, D. Kalenichenko and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *CVPR*, 2015.

[7] K. Berlin, C. C.-S. S. Koren, J. P. Drake, J. M. Landolin and A. M. Phillippy, "Assembling large genomes with single-molecule sequencing and locality-sensitive hashing," *Nature Biotechnology,* 2015.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems,* 2017.

[9] "Introducing ChatGPT," 2023. [Online]. Available: https://openai.com/blog/chatgpt.

[10] T. Coughlin, "Digital Storage At DreamWorks Animation," 2024. [Online]. Available: https://www.forbes.com/sites/tomcoughlin/2017/08/15/digital-storage-at-dreamworks-animation/.

[11] "Zilliz," 2023. [Online]. Available: https://zilliz.com/.

[12] "Pinecone," 2023. [Online]. Available: https://www.pinecone.io/.

[13] "Qdrant," [Online]. Available: https://qdrant.tech/.

[14] M. Kwan, "Finding the optimal number of dimensions for word embeddings," [Online]. Available: https://medium.com/@matti.kwan/finding-the-optimal-number-of-dimensions-for-word-embeddings-f19f71666723.

[15] Google, "Embeddings in the Gemini API," [Online]. Available: https://ai.google.dev/gemini-api/docs/embeddings.

[16] OpenAI, "New and improved embedding model," [Online]. Available: https://openai.com/index/new-and-improved-embedding-model/.

[17] K. Henner, "An intuitive introduction to text embeddings," 2023. [Online]. Available: https://stackoverflow.blog/2023/11/09/an-intuitive-introduction-to-text-embeddings/.

[18] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM,* 1975.

[19] A. Guttman, "R-trees: a dynamic index structure for spatial searching," *SIGMOD,* 1984.

[20] H. Zhang, D. G. Andersen, A. Pavlo, M. Kaminsky, L. Ma and R. Shen, "Reducing the Storage Overhead of Main-Memory OLTP Databases with Hybrid Indexes," *SIGMOD,* 2016.

[21] R. Agarwal, A. Khandelwal and I. Stoica, "Succinct: enabling queries on compressed data," in *NSDI*, 2015.

[22] A. Khandelwal, R. Agarwal and I. Stoica, "BlowFish: dynamic storage-performance tradeoff in data stores," in *NSDI*, 2016.

[23] A. Khandelwal, Z. Yang, E. Ye, R. Agarwal and I. Stoica, "ZipG: A Memory-efficient Graph Store for Interactive Queries," *SIGMOD,* 2017.

[24] H. Zhang, H. Lim, V. Leis, D. G. Andersen, M. Kaminsky, K. Keeton and A. Pavlo, "SuRF: Practical Range Query Filtering with Fast Succinct Tries," *SIGMOD,* 2018.

[25] D. A. Alcantara, A. Sharf, F. Abbasinejad, S. Sengupta, M. Mitzenmacher, J. D. Owens and N. Amenta, "Real-time parallel hashing on the GPU," in *SIGGRAPH Asia*, 2009.

[26] Y. Deng, M. Yan and B. Tang, "Accelerating Merkle Patricia Trie with GPU," *VLDB,* 2024.

[27] A. Khandelwal, R. Agarwal and I. Stoica, "Confluo: Distributed Monitoring and Diagnosis Stack for High-speed Networks," in *NSDI*, 2019.

[28] A. Khandelwal, Y. Tang, R. Agarwal, A. Akella and I. Stoica, "Jiffy: elastic far-memory for stateful serverless analytics," in *EuroSys*, 2022.

[29] M. Vuppalapati, G. Fikioris, R. Agarwal, A. Cidon, A. Khandelwal and E. Tardos, "Karma: Resource Allocation for Dynamic Demands," in *OSDI*, 2023.

[30] S.-s. Lee, Y. Yu, Y. Tang, A. Khandelwal, L. Zhong and A. Bhattacharjee.

[31] H. Zhang, Y. Tang, A. Khandelwal, J. Chen and I. Stoica, "Caerus: NIMBLE Task Scheduling for Serverless Analytics," in *NSDI*, 2021.

[32] H. Zhang, Y. Tang, A. Khandelwal and I. Stoica, "SHEPHERD: Serving DNNs in the Wild," in *NSDI*, 2023.

[33] Z. Mao, K. Srinivasan and A. Khandelwal, "Trinity: A Fast Compressed Multi-attribute Data Store," in *EuroSys*, 2024.

[34] P. Grubbs, A. Khandelwal, M.-S. Lacharite, L. Brown, L. Li, R. Agarwal and T. Ristenpart, "Pancake: Frequency Smoothing for Encrypted Data Stores," in *USENIX Security*, 2020.

[35] M. Vuppalapati, K. Babel, A. Khandelwal and R. Agarwal, "SHORTSTACK: Distributed, Fault-tolerant, Oblivious Data Access," in *OSDI*, 2022.

[36] G. Jia, R. Agarwal and A. Khandelwal, "Length Leakage in Oblivious Data Access Mechanisms," in *USENIX Security*, 2024.

[37] S. Chakraborty, R. Grossi, K. Sadakane and S. R. Satti, "Succinct representations for (Non) deterministic finite automata," in *International Conference on Language and Automata Theory and Applications*, 2021.

[38] N. a. P. N. Cotumaccio, "On indexing and compressing finite automata," in *SODA*, 2021.

[39] M. L. Fredman and D. E. Willard, "BLASTING through the information theoretic barrier with FUSION TREES," in *ACM Symposium on Theory of Computing*, 1990.

[40] P. Ram and K. Sinha, "Revisiting kd-tree for Nearest Neighbor Search," in *SIGKDD*, 2019.

[41] A. Babenko and V. Lempitsky, "The inverted multi-index," in *CVPR*, 2012.

[42] Y. A. Malkov and D. A. Yashunin, "Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 2020.