Automated Data Selection for Efficient Cost Model Training to Optimize Sparse Matrix Kernels on Emerging Hardware Accelerators

Anonymous Author(s)

Abstract

Sparse matrix computations are critical for many applications in machine learning, computer vision, and scientific computing. However, optimizing sparse kernels, such as Sparse Matrix-Matrix Multiplication (SpMM) and Sampled Dense-Dense Matrix Multiplication (SDDMM), remain challenging as their performance is sensitive to input characteristics and high dimensionality of the scheduling search space. Specifically, this complexity arises from the interplay of factors such as matrix dimensions, sparsity patterns, sparse storage formats, hardware targets, and compiler-specific scheduling primitives, which together create a highly irregular and non-intuitive performance landscape. While prior work has introduced learned cost models to guide the selection of scheduling primitives, these cost models are typically kernel- and hardware-specific, and either require millions of training samples or depend heavily on expert-designed heuristics.

In this work, we frame optimizing sparse matrix kernels as a structured exploration problem and identify key limitations in prior work, including its inability to generalize across kernels and hardware, and to train cost models with limited data samples without relying on expert heuristics. We then propose a solution to automate the data collection effort for cost model training on emerging hardware accelerators. Our method augments a state-of-the-art (SOTA) framework with exploration-aware data sampling and multi-armed bandit-based active learning, enabling data-efficient fine-tuning with minimal manual interventions. Our experimental results demonstrate that these strategies substantially reduce reliance on large training datasets and expert heuristics, while achieving performance comparable to SOTA.

1 Introduction

Sparse matrix computations are at the core of many modern workloads in machine learning (Child et al. (2019); Ye & Ji (2021); Dao et al. (2021)), computer vision (Liu et al. (2015)), scientific computing (Li et al. (2023a)), high-performance computing (Siegel et al. (2010)), and graph analytics (Ashari et al. (2014); Serrano (2019)). Two sparse kernels that are commonly used in these domains are Sparse Matrix-Matrix Multiplication (SpMM) and Sampled Dense-Dense Matrix Multiplication (SDDMM) (Rahman et al. (2021)). However, unlike dense computations, the performance of sparse computations is notoriously hard to optimize. This difficulty stems from several sources, such as sensitivity to the input matrix's sparsity pattern, the choice of sparse storage format, hardware-specific behavior (Won et al. (2023); Ye et al. (2023); Sudusinghe et al. (2025)), inconsistencies across compiler backends, and the large and irregular search space of low-level scheduling primitives. For example, for a matrix with one million rows and columns, the search space explored by WACO (Won et al. (2023)) for SpMM on CPU can include approximately 6.9 million valid scheduling choices.

To address this, previous work has adopted both hand-crafted heuristics (Kjolstad et al. (2017); Hong et al. (2019)) and supervised learning-based cost models (Sun et al. (2021); Won et al. (2023); Sudusinghe et al. (2025)) that guide the search for high-performance implementations. Here, cost models (Baghdadi et al. (2021); Mendis et al. (2019)) serve as efficient surrogates for evaluating a

Submitted to the First Exploration in AI Today Workshop at ICML (EXAIT at ICML 2025). Do not distribute.

workload's performance by estimating its execution time on real hardware. Prior work has developed dedicated cost models for each sparse kernel and target hardware (Sun et al. (2021); Won et al. (2023); Sudusinghe et al. (2025). Compared to heuristic-based methods, these learned cost models consistently deliver better performance and adaptability across a wide range of inputs. For instance, WACO (Won et al. (2023)) trains cost models that predict runtime across diverse sparsity patterns and scheduling primitives, enabling automated search algorithms to identify schedules that lead to better runtime performance compared to frameworks like TACO (Kjolstad et al. (2017)) (detailed in Appendix A) and AsPT (Hong et al. (2019)). However, these models tend to require large-scale datasets, often in the order of millions of measurements, to accurately predict performance (Won et al. (2023)). Furthermore, both model training and search rely on heuristically defined search spaces (e.g., powers-of-two loop tiling) and pre-defined scheduling constraints (Won et al. (2023); Sudusinghe et al. (2025)), making them inherently brittle due to their dependence on expert knowledge.

This problem is exacerbated when targeting hardware platforms such as emerging hardware accelerators (Gerogiannis et al. (2023); Jin et al. (2024)), where collecting a single performance data point can take hours to weeks due to the high cost of running simulations (Sudusinghe et al. (2025)). Yet, the adoption of these accelerators is becoming mainstream (Aananthakrishnan et al. (2023); Gerogiannis et al. (2023); Hegde et al. (2019); Li et al. (2023b); Muñoz-Martínez et al. (2023); Jin et al. (2024)) due to their potential to deliver significant performance improvements over conventional hardware platforms. To tackle the significant overheads associated with data collection, prior work (Sudusinghe et al. (2025)) has typically pretrained cost models on CPU (the conventional hardware platform) data samples and then applied transfer learning to the accelerator domain (the specialized hardware platform) by fine-tuning on a small, expertly curated dataset. This workflow adds additional manual interventions, such as selecting matrix samples for fine-tuning, and managing both homogeneous and heterogeneous scheduling primitives across different backends (Sudusinghe et al. (2025)), which require domain expertise. These challenges underscore a more fundamental problem, which is the absence of automated, generalizable, and data-efficient exploration strategies for model training and auto-tuning when optimizing sparse matrix computations across different hardware platforms.

In this paper, we present a perspective that frames optimizing sparse matrix computations across hardware platforms as a structured exploration problem. These computations require navigating large, complex, and high-dimensional search spaces. This challenge is particularly acute for emerging hardware platforms such as sparse accelerators (Gerogiannis et al. (2023); Sudusinghe et al. (2025)). We investigate the following research question: *Can we design automated strategies to guide data collection during cost model training, strategies that effectively identify high-performing schedules using limited data and with minimal reliance on expert intervention?*

Our contributions are as follows:

- We consolidate and formalize the limitations of prior work, introducing a problem formulation that synthesizes key challenges in sparse matrix computations, including complexities of search spaces of scheduling primitives and data inefficiency in cost model training and auto-tuning.
- We augment a state-of-the-art pipeline with exploration-aware sampling and bandit-based active learning to enable automated data-efficient fine-tuning on emerging hardware accelerators.
- Our automated exploration strategies reduce reliance on large datasets and expert intervention while achieving comparable performance to state-of-the-art methods, laying the foundation for developing scalable and generalizable frameworks for cost model training and fine-tuning.

2 Problem Formulation

In this section, we formalize the challenges that make optimizing sparse matrix computations a complex exploration problem. This difficulty arises due to (i) the complexity of exploring the search space to identify high-performing schedules (Section 2.1), and (ii) the difficulty of training cost models that are both accurate and generalizable, while minimizing data collection overhead and expert interventions (Section 2.2).

2.1 Search Space Complexity in Optimizing Sparse Matrix Computations

Optimizing sparse matrix computations differs fundamentally from optimizing dense computations (Zheng et al. (2020, 2021); Zhai et al. (2023)). Their performance is influenced by several factors: scheduling primitives (i.e., program optimization parameters), input matrix dimensions and sparsity

patterns, and sparse storage formats (e.g., COO, CSR, CSC). These factors interact to produce a highly irregular and input-dependent performance landscape. Each primitive, such as SpMM or SDDMM, exposes a distinct set of low-level scheduling primitives (e.g., loop orderings, loop tiling, vectorization), whose impact can vary widely across hardware platforms. The structure of the non-zero elements in the matrix can significantly affect which schedules yield good performance, even when matrix dimensions and non-zero counts remain constant. This introduces a fine-grained sensitivity that renders static heuristics or fixed parameter selections brittle and non-generalizable.

The complexity increases further when considering multiple hardware targets and compiler backends. The same scheduling primitive (e.g., loop tiling) can lead to vastly different performance outcomes depending on whether the computation is executed on a CPU, a GPU, or a sparse accelerator. Moreover, different compiler backends may support different sets of primitives or apply them in backend-specific ways, resulting in divergent behavior even for semantically similar schedules. This leads to heterogeneous and backend-dependent search spaces. Consequently, any effective exploration strategy must contend not only with the size of the search space but also with its structural and semantic variability across hardware and compiler implementations. Below, we have listed down the factors that influence the structure and complexity of these search spaces. The equations for sparse matrix kernels are expressed using Einstein summation (Einsum) notation (Barr (1991)).

- **Kernel-Specific Scheduling Primitives:** Different kernels (e.g., SpMV, SpMM, SDDMM) expose different sets of low-level scheduling primitives. For example, SpMV involves two loop indices, while SpMM adds a reduction loop, enabling tiling across three dimensions. Moreover, certain kernels may leverage only a subset of the available choices for the scheduling primitives, leading to inherently different search spaces across kernels. These structural differences are directly reflected in how loop nests are organized and scheduled. For example, in WACO (Won et al. (2023)), the sparsity pattern for SpMM ($D_{i,k} = \sum_j A_{i,j} \cdot B_{j,k}$, where A is sparse and B is dense) resides in the input matrix **A**, which is sparse and drives the computation along the row and inner product dimensions. This results in loop tiling and scheduling primitives typically aligned with the access pattern over i, k, and j dimensions, with index layout encoded as `['i1', 'i0', 'k1', 'k0', 'j1', 'j0']`. In contrast, SDDMM ($D_{i,k} = A_{i,k} \cdot \sum_j B_{i,j} \cdot C_{j,k}$, where D and A are sparse, while B and C are dense) computes values only at locations where the output matrix **D** is sparse (Rahman et al. (2021); Gerogiannis et al. (2023)). Here, the sparsity pattern determines which (i, k) pairs are evaluated, while the inner summation over j (used to compute dense inner products) must be efficiently handled. Consequently, SDDMM schedules typically operate over a different loop index layout: `['i1', 'i0', 'j1', 'j0', 'k1', 'k0']`. This divergence implies that each kernel exploits only a subset of choices for the scheduling primitives relevant to its computational pattern. For example, certain permutations or tiling strategies applicable to SpMM may be invalid or suboptimal for SDDMM. As such, even when targeting the same hardware or compiler backend, the effective scheduling search space is kernel-specific, and exploration strategies must adapt accordingly to yeild better results.
- **Input Sparse Matrix:** The dimensions of the matrix and the overall sparsity level (i.e., the percentage of non-zero elements) directly affect which scheduling strategies are performant. Beyond these characteristics, the specific distribution of non-zeros (i.e., the sparsity pattern) can further influence performance, as certain schedules may exploit structured or localized sparsity better than others. This makes schedule selection highly input-dependent.
- **Sparse Storage Format:** The choice of sparse storage format (e.g., COO, CSR, CSC) affects data access patterns, iteration structure, and memory efficiency, directly influencing which scheduling strategies are performant.
- **Hardware Platform:** Hardware architectures differ in their compute models, memory hierarchies, and parallelization strategies. A schedule optimized for CPUs may perform poorly or be invalid on GPUs or accelerators, requiring hardware-aware exploration strategies.
- **Compiler Backend and Constraints:** Compiler backends vary in how they interpret and apply scheduling primitives. Some primitives exhibit homogeneous behavior across backends (e.g., loop tiling), while others are heterogeneous, with backend-specific semantics (e.g., unrolling).

Given the factors outlined above, the goal of exploring the scheduling search space for a sparse kernel is to efficiently identify high-performing schedules from a combinatorially large and highly sensitive set of candidates. For example, in the case of WACO, the search space for a single matrix with one million rows and columns can include approximately 6.9 million candidate schedules ((Won et al.,

2023)). To formalize this, let $\mathcal{S}(k, M, f, h, c)$ denote the configuration space for a sparse kernel k, given an input matrix M, a sparse storage format f, a hardware platform h, and a compiler backend c. The goal of exploration is to identify a schedule $x^* \in \mathcal{S}$ that minimizes the runtime cost $C(x)$:

$$x^* = \arg \min_{x \in \mathcal{S}(k,M,f,h,c)} C(x) \tag{1}$$

In practice, evaluating $C(x)$ requires executing the schedule on hardware or invoking a simulator, which is computationally expensive, particularly during the early design stages of emerging accelerators. For example, collecting a single data point using the SPADE sparse accelerator's simulator can take up to two weeks ((Gerogiannis et al., 2023; Sudusinghe et al., 2025)). This motivates the need for data-efficient exploration strategies that can infer schedules with minimal runtime evaluations.

2.2 Complexities in Cost Model Training

Accurate cost models are critical for enabling efficient exploration of the large search spaces associated with scheduling primitives in optimizing sparse kernels (Adams et al. (2019); Baghdadi et al. (2021); Won et al. (2023); Sudusinghe et al. (2025)). These models serve as surrogates for expensive runtime evaluations, predicting the execution time of candidate schedules and guiding search algorithms toward promising ones. However, training robust and generalizable learned cost models (typically neural network-based) is challenging due to several limiting factors. First, cost models often require large training datasets to achieve acceptable predictive performance (Zheng et al. (2021); Baghdadi et al. (2021); Won et al. (2023)). For example, WACO (Won et al. (2023)) required over 2.1 million data points to train a cost model for a single kernel (SpMM) on CPU. This reliance on large datasets becomes a significant bottleneck when targeting emerging accelerators, where large-scale data collection takes many months to years (Gerogiannis et al. (2023); Sudusinghe et al. (2025)). Second, the quality and diversity of the training dataset are crucial. Existing work either samples schedules randomly or relies on expert-curated heuristics to guide data collection. For instance, COGNATE (Sudusinghe et al. (2025)) selects matrices using a heuristic-based binning strategy for fine-tuning and constrains the number of candidate schedules per matrix to 256. These heuristics introduce both bias and brittleness, causing the model's accuracy to become tightly coupled to the quality of these expertly selected data, and leading to poor generalization outside of them. Third, existing cost models are typically kernel-specific and hardware-specific (Sasaki et al. (2022); Won et al. (2023); Sudusinghe et al. (2025)), requiring training or fine-tuning of separate models for each distinct kernel or hardware platform. Finally, most cost models for sparse kernels follow a supervised learning paradigm (Adams et al. (2019); Zheng et al. (2021); Baghdadi et al. (2021); Sasaki et al. (2022); Won et al. (2023); Sudusinghe et al. (2025)). They lack mechanisms for actively sampling data points or learning on uncertain regions of the search space. As a result, they remain data-hungry, static, and inefficient in complex optimization landscapes. Below, we have listed down the factors that influence the efficiency, generalization, and robustness of a learned cost model during training.

- **Dataset Size:** Many models require millions of labeled data samples to reach high accuracy (Zheng et al. (2021); Baghdadi et al. (2021); Won et al. (2023), which is impractical at scale, especially for emerging hardware accelerators.
- **Dataset Diversity and Quality:** Narrowly sampled datasets or heuristic-driven subsets can lead to overfitting and poor generalization to unseen matrices or schedules.
- **Model Architecture:** The capacity of the model (e.g., neural network parameters, representation encoding) directly affects its ability to capture complex, nonlinear performance relationships.
- **Transferability Across Domains:** Models trained for specific kernels, hardware platforms, or sparsity regimes often generalize poorly to others, requiring repeated training or fine-tuning.
- **Learning Paradigm:** The use of static supervised learning limits data efficiency. Alternative paradigms like active learning or transfer learning may improve sample efficiency and robustness.
- **Loss Function:** The training objective influences whether the model learns to predict absolute performance or relative rankings across configurations (Kaufman et al. (2021)). For example, choosing between regression loss and ranking loss leads to different downstream tasks.

We formalize the outcome of cost model training as an optimization objective, where prediction quality depends on the model architecture $\mathcal{NN}$ (e.g., a neural network), hyperparameters ϕ (e.g., learning rate, optimizer, batch size), dataset $\mathcal{D}$ size and quality Q, loss function $\mathcal{L}$, and training strategy T (e.g., supervised learning, active learning). Let $\mathcal{K}$ denote the set of sparse kernels, and consider a

specific kernel $k \in \mathcal{K}$. For kernel k, we define the dataset as $\mathcal{D}_k = \{(M_j, x_i, C_k(M_j, x_i))\}_{j=1,\ i=1}^{J,\ I}$, where M_j is an input sparse matrix, x_i is a candidate schedule, and $C_k(M_j, x_i)$ is the runtime cost of applying schedule x_i to matrix M_j under kernel k. Our objective is to select a training subset $\mathcal{D}_k^{\text{train}} \subseteq \mathcal{D}_k$ that minimizes the expected loss on a separate validation set $\mathcal{D}_k^{\text{val}}$, i.e.,

$$\mathcal{D}_k^{\text{train}*} = \arg \min_{\mathcal{D}_k^{\text{train}} \subseteq \mathcal{D}_k} \mathbb{E}_{(M_j, x_i, C_k(M_j, x_i)) \sim \mathcal{D}_k^{\text{val}}} \left[\mathcal{L}(\mathcal{NN}_\phi(M_j, x_i), C_k(M_j, x_i)), |\mathcal{D}_k|, Q(\mathcal{D}_k), T) \right] \tag{2}$$

The loss $\mathcal{L}$ can vary based on the learning objective, such as minimizing absolute prediction error (e.g., mean squared error) or optimizing schedule ranking (e.g., pairwise ranking loss).

3 Methodology

Building on the formalization in Section 2, we introduce strategies designed to address search space complexities and data efficiency challenges of cost model training. Our solutions (Figure 1) are centered on learning structural representations of matrices and leveraging them to guide exploration. We achieve this by decoupling matrix representation learning from the performance model.

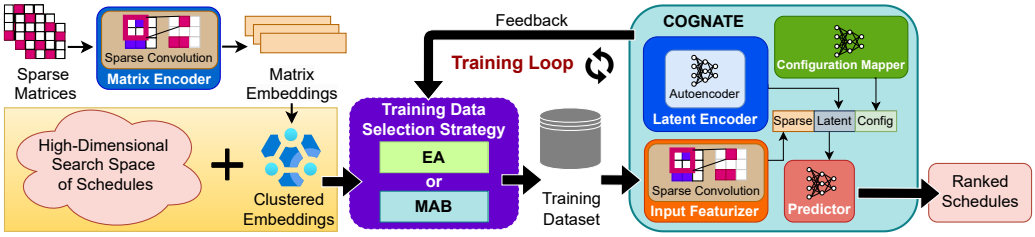

Figure 1: An overview of how our data collection strategies are integrated with COGNATE

3.1 Matrix Representation Learning via Autoencoder Pretraining

We decoupled the sparse convolutional neural network (SCNN) (Graham & Van der Maaten (2017)) from the performance model by first pretraining an autoencoder to learn sparsity patterns from input matrices without using any runtime data samples. The resulting structural representations can then be leveraged to guide the data collection process for cost model training. Once pretraining is complete, we extract latent embeddings from the encoder component and apply k-means clustering to group matrices based on structural similarity. From each cluster, we select representative matrices during the data collection process, ensuring structural diversity in the training set. This removes the need for expert-designed heuristics to curate training subsets. In addition to improving data efficiency and generalization by avoiding overfitting to a narrow set of sparsity patterns, this approach also reduces the burden of collecting performance data. Since the SCNN-based encoder is pretrained on a large collection of input sparsity patterns, the performance model benefits from a meaningful initialization, leading to faster convergence and improved robustness during cost model training.

Autoencoder Training. Given a sparse matrix in COO format, we construct a sparse tensor with binary features and pass it through a stack of strided sparse convolutional layers to obtain a compact latent embedding using the encoder. Two decoders are trained jointly: one reconstructs a 3D shape vector (log-scaled rows, columns, and non-zeros), and the other predicts a 32×32 downsampled dense representation of the matrix's sparsity pattern, obtained via area-based interpolation. We train the autoencoder by minimizing the sum of mean squared errors between the predicted and target shape vectors and sparsity patterns. Once trained, the encoder functions as a matrix feature extractor that captures structural representations in the input sparsity patterns. These learned embeddings can then be reused in downstream tasks such as cost model training and matrix clustering.

3.2 Automated Data Collection for Emerging Sparse Accelerators

Sparse accelerators often operate under extreme constraints, limiting the feasibility of large-scale data collection (as detailed in Section 2.2). Existing fine-tuning pipelines, such as COGNATE (Sudusinghe et al. (2025)), depend heavily on expert-curated subsets of input matrices and schedules. While effective, such reliance introduces inductive bias and hampers generalization and scalability to

unseen sparsity patterns or novel hardware platforms. To address this, we introduce two exploration-driven techniques for data-efficient training: *exploration-aware sampling* and *multi-armed bandit (MAB)*-based active learning (Slivkins et al. (2019)). These are implemented independently and offer potential solutions for developing highly accurate cost models with minimal data requirement.

Exploration-Aware (EA) Sampling. We implement a reward-driven strategy that avoids manual dataset curation under a small data budget. The approach begins by clustering a large input matrix pool using autoencoder-derived embeddings and initializing training with a small, uniformly sampled subset of input matrices, one from each cluster. Each selected matrix is initially assigned a fixed number of schedules (e.g., 10). In subsequent training epochs, additional samples are allocated based on recent performance. We compute a sampling probability p_M for each eligible matrix M using a softmax over a score that balances exploitation and exploration:

$$p_M = \frac{\exp\left(\alpha(1 - s_M) + (1 - \alpha)s_M + b_M\right)}{\sum_{M'} \exp\left(\alpha(1 - s_{M'}) + (1 - \alpha)s_{M'} + b_{M'}\right)} \tag{3}$$

Here, $s_M \in [0, 1]$ is a normalized performance score based on recent model accuracy, and $b_M = 0.2$ provides a fixed bonus for cold-start matrices (i.e., those not yet used). The hyperparameter $\alpha \in [0, 1]$ balances exploration and exploitation. The matrices are drawn from this distribution and allocated additional training samples, subject to a maximum number of active matrices and per-matrix schedule caps. The score s_M is updated using an exponential moving average:

$$s_M^{(t+1)} = (1 - \eta)s_M^{(t)} + \eta r_M \tag{4}$$

where r_M is the reward signal, typically derived from rank prediction quality (e.g., ordered pair accuracy), and η is a smoothing constant. The matrices not selected in an epoch receive zero reward, promoting long-term exploration. To ensure structural diversity, new matrices are introduced only when the active set is under capacity, with cluster-level quotas to maintain balance. This exploration-aware policy enables data-efficient discovery of sparsity patterns and thier schedules.

Multi-Armed Bandit (MAB)-Based Active Learning. We also explore a MAB-inspired strategy that reduces the data footprint by restricting training to a fixed set of input matrices. Specifically, we cluster a large input matrix pool and select a small, representative subset of 25 matrices, five from each cluster, as fixed arms. These matrices remain constant throughout training, while their associated schedules are incrementally explored over time. At each epoch, the model ranks matrices using an Upper Confidence Bound (UCB) score (Carpentier et al. (2011)):

$$\text{UCB}(M) = \frac{R_M}{N_M} + \sqrt{\frac{2 \log T}{N_M}} \tag{5}$$

Here, R_M is the cumulative reward for matrix M, N_M is the number of schedules sampled from it so far, and $T = \sum_M N_M$ is the total number of scheduling attempts across all matrices. The reward R_M is derived from the model's ranking accuracy on recent schedules, encouraging allocations to matrices where the cost model exhibits greater uncertainty or poor ordering. Using UCB scores, we prioritize matrices with high expected gain or unexplored potential. During each epoch, a small budget of new schedules (e.g., 5) is allocated to the top-ranked matrices according to their UCB scores. This allows the training process to focus on matrices that yield poor cost model predictions.

4 Evaluation

We evaluate our exploration strategies in the context of fine-tuning cost models for optimizing sparse kernels on emerging accelerators. Specifically, we augment the COGNATE pipeline (Sudusinghe et al. (2025)), a state-of-the-art framework that leverages transfer learning to adapt cost models from CPUs to sparse accelerators. Our goal is to assess whether automated data exploration strategies introduced in Section 3 can reduce expert interventions while maintaining or improving performance.

Experimental Setup. We use real-world sparse matrices from the SuiteSparse Matrix Collection (Davis & Hu (2011)), which offers a diverse range of workloads with varying sparsity patterns. Following COGNATE, we first pretrain separate cost models for SpMM and SDDMM using schedule data collected on an Intel Xeon Gold 6348 CPU. We then fine-tune each model using performance data gathered using the simulator of the SPADE sparse accelerator Gerogiannis et al. (2023).

Implementation. We implemented our cost model using PyTorch, employing sparse convolutional layers from MinkowskiEngine Choy et al. (2019) to encode the matrix structure. For autoencoder pretraining, we downsampled sparse matrix representations using OpenCV's area-based interpolation.

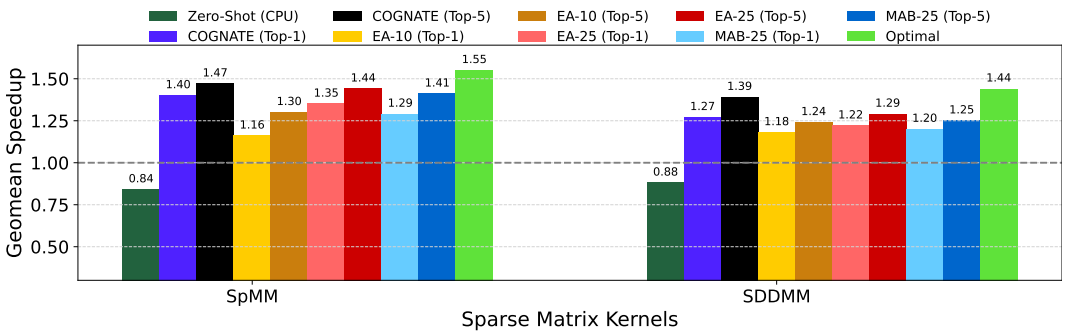

Figure 2: Geomean speedups of EA, MAB, COGNATE, and Zero-Shot, normalized to the baseline.

Baselines. We compare our strategies against COGNATE's fine-tuning framework, which relies on expert-selected matrices and constraints. All evaluations are conducted for the SPADE accelerator. We report speedups over SPADE's default implementation, which applies its native scheduling rules without learned cost models or search, to isolate the impact of our automated data collection strategies.

Experiments. To ensure comparability with COGNATE, we begin by selecting 100 matrices using our autoencoder-based clustering method. Each matrix is paired with 100 randomly sampled schedules, forming the CPU pretraining dataset. For fine-tuning on SPADE, we consider all remaining input matrices as candidates and restrict the total number of fine-tuning data samples to 500, consistent with COGNATE's low-data regime. We evaluate three automated exploration strategies: (1) **EA-10**, an exploration-aware approach that selects at most 10 matrices and emphasizes schedule diversity; (2) **EA-25**, which increases matrix diversity by selecting up to 25 matrices; and (3) **MAB-25**, a multi-armed bandit strategy that uses a fixed set of 25 matrices as arms. We evaluate on a held-out set of 715 real-world matrices from the SuiteSparse Matrix Collection (Davis & Hu (2011)).

Results and Discussion. Figure 2 shows the fine-tuning performance of our exploration strategies on both SpMM and SDDMM kernels, measured as Top-1 and Top-5 speedups over SPADE's default implementation. The expert-guided baseline, **COGNATE** (Sudusinghe et al. (2025)), achieves strong performance with 1.40× (Top-1) and 1.47× (Top-5) speedups. Our automated strategies deliver comparable results under the same data budget with no manual data curation. **EA-10**, which emphasizes schedule diversity across 10 matrices, reaches 1.16× (Top-1) and 1.30× (Top-5). Expanding to 25 matrices, **EA-25** improves performance to 1.35× and 1.44×, closely matching COGNATE. **MAB-25**, a bandit-based strategy using fixed matrix arms, achieves 1.29× and 1.41× speedups. A similar trend is observed for the SDDMM kernel. These consistent trends across both SpMM and SDDMM indicate that our exploration strategies generalize across kernels with differing characteristics and scheduling constraints. Our experimental results validate our hypothesis that automated exploration strategies can replace expert heuristics in training cost models for optimizing sparse matrix kernels. EA-25 demonstrates the benefit of matrix-level diversity, while MAB-25 shows that uncertainty-driven allocation can yield robust gains even with a fixed matrix pool. Together, these results demonstrate that combining unsupervised matrix representation learning with exploration-guided sampling paves a scalable and data-efficient path for developing data-efficient cost models that could effectively identify high-performing schedules with minimal expert interventions.

5 Conclusion

In this paper, we presented automated strategies for guiding the data collection effort during cost model training for optimizing sparse matrix computations on emerging accelerators. By leveraging unsupervised structural representations and introducing exploration-aware sampling and a bandit-based active learning approach, we eliminate the need for manual dataset curation while achieving performance comparable to expert-driven baselines such as COGNATE under the same data budget. Our results demonstrate the potential of automated exploration strategies as a viable alternative to expert interventions, offering a scalable and data-efficient path for identifying high-performing schedules for sparse matrix computations across increasingly complex hardware–software environments.

References

Aananthakrishnan, S., Abedin, S., Cavé, V., Checconi, F., Du Bois, K., Eyerman, S., Fryman, J. B., Heirman, W., Howard, J., Hur, I., et al. The intel programmable and integrated unified memory architecture graph analytics processor. *IEEE Micro*, 43(5):78–87, 2023.

Adams, A., Ma, K., Anderson, L., Baghdadi, R., Li, T.-M., Gharbi, M., Steiner, B., Johnson, S., Fatahalian, K., Durand, F., et al. Learning to optimize halide with tree search and random programs. *ACM Transactions on Graphics (TOG)*, 38(4):1–12, 2019.

Ashari, A., Sedaghati, N., Eisenlohr, J., Parthasarath, S., and Sadayappan, P. Fast sparse matrix-vector multiplication on gpus for graph applications. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 781–792. IEEE, 2014.

Baghdadi, R., Merouani, M., Leghettas, M.-H., Abdous, K., Arbaoui, T., Benatchba, K., et al. A deep learning based cost model for automatic code optimization. *Proceedings of Machine Learning and Systems*, 3:181–193, 2021.

Barr, A. H. The einstein summation notation. *An Introduction to Physically Based Modeling (Course Notes 19), pages E*, 1:57, 1991.

Carpentier, A., Lazaric, A., Ghavamzadeh, M., Munos, R., and Auer, P. Upper-confidence-bound algorithms for active learning in multi-armed bandits. In *International Conference on Algorithmic Learning Theory*, pp. 189–203. Springer, 2011.

Chen, T. and Guestrin, C. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Shen, H., Cowan, M., Wang, L., Hu, Y., Ceze, L., et al. {TVM}: An automated {End-to-End} optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp. 578–594, 2018.

Child, R., Gray, S., Radford, A., and Sutskever, I. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Choy, C., Gwak, J., and Savarese, S. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 3075–3084, 2019.

Dao, T., Chen, B., Liang, K., Yang, J., Song, Z., Rudra, A., and Re, C. Pixelated butterfly: Simple and efficient sparse training for neural network models. *arXiv preprint arXiv:2112.00029*, 2021.

Davis, T. A. and Hu, Y. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1–25, 2011.

Gerogiannis, G., Yesil, S., Lenadora, D., Cao, D., Mendis, C., and Torrellas, J. Spade: A flexible and scalable accelerator for spmm and sddmm. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ISCA '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400700958. doi: 10.1145/3579371.3589054. URL `https://doi.org/10.1145/3579371.3589054`.

Graham, B. and Van der Maaten, L. Submanifold sparse convolutional networks. *arXiv preprint arXiv:1706.01307*, 2017.

Hegde, K., Asghari-Moghaddam, H., Pellauer, M., Crago, N., Jaleel, A., Solomonik, E., Emer, J., and Fletcher, C. W. Extensor: An accelerator for sparse tensor algebra. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 319–333, 2019.

Hong, C., Sukumaran-Rajam, A., Nisa, I., Singh, K., and Sadayappan, P. Adaptive sparse tiling for sparse matrix multiplication. In *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming*, pp. 300–314, 2019.

Jin, H., Yue, Z., Zhao, Z., Du, Y., Deng, C., Srivastava, N., and Zhang, Z. Vesper: A versatile sparse linear algebra accelerator with configurable compute patterns. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

Kaufman, S., Phothilimthana, P., Zhou, Y., Mendis, C., Roy, S., Sabne, A., and Burrows, M. A learned performance model for tensor processing units. *Proceedings of Machine Learning and Systems*, 3:387–400, 2021.

Kjolstad, F., Kamil, S., Chou, S., Lugato, D., and Amarasinghe, S. The tensor algebra compiler. *Proc. ACM Program. Lang.*, 1(OOPSLA), oct 2017. doi: 10.1145/3133901. URL `https://doi.org/10.1145/3133901`.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Li, J., Ren, S.-g., Li, Y., Yang, L., Yu, Y., Ni, R., Zhou, H., Bao, H., He, Y., Chen, J., et al. Sparse matrix multiplication in a record-low power self-rectifying memristor array for scientific computing. *Science Advances*, 9(25):eadf7474, 2023a.

Li, Z., Li, J., Chen, T., Niu, D., Zheng, H., Xie, Y., and Gao, M. Spada: Accelerating sparse matrix multiplication with adaptive dataflow. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 747–761, 2023b.

Liu, B., Wang, M., Foroosh, H., Tappen, M., and Pensky, M. Sparse convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 806–814, 2015.

Mendis, C., Renda, A., Amarasinghe, S., and Carbin, M. Ithemal: Accurate, portable and fast basic block throughput estimation using deep neural networks. In *International Conference on machine learning*, pp. 4505–4515. PMLR, 2019.

Muñoz-Martínez, F., Garg, R., Pellauer, M., Abellán, J. L., Acacio, M. E., and Krishna, T. Flexagon: A multi-dataflow sparse-sparse matrix multiplication accelerator for efficient dnn processing. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pp. 252–265, 2023.

Ragan-Kelley, J., Adams, A., Sharlet, D., Barnes, C., Paris, S., Levoy, M., Amarasinghe, S., and Durand, F. Halide: Decoupling algorithms from schedules for high-performance image processing. *Communications of the ACM*, 61(1):106–115, 2017.

Rahman, M. K., Sujon, M. H., and Azad, A. Fusedmm: A unified sddmm-spmm kernel for graph embedding and graph neural networks. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 256–266. IEEE, 2021.

Sasaki, Y., Takahashi, K., Shimomura, Y., and Takizawa, H. A cost model for compilers based on transfer learning. In *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 942–951. IEEE, 2022.

Serrano, M. J. Efficient implementation of sparse matrix-sparse vector multiplication for large scale graph analytics. In *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–7. IEEE, 2019.

Siegel, J., Villa, O., Krishnamoorthy, S., Tumeo, A., and Li, X. Efficient sparse matrix-matrix multiplication on heterogeneous high performance systems. In *2010 IEEE International Conference On Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS)*, pp. 1–8. IEEE, 2010.

Slivkins, A. et al. Introduction to multi-armed bandits. *Foundations and Trends® in Machine Learning*, 12(1-2):1–286, 2019.

Sudusinghe, C., Gerogiannis, G., Lenadora, D., Block, C., Torrellas, J., and Mendis, C. Cognate: Acceleration of sparse tensor programs on emerging hardware using transfer learning. In *International Conference on machine learning (ICML)*. PMLR, 2025. To appear.

Sun, Q., Liu, Y., Yang, H., Dun, M., Luan, Z., Gan, L., Yang, G., and Qian, D. Input-aware sparse tensor storage format selection for optimizing mttkrp. *IEEE Transactions on Computers*, 71(8): 1968–1981, 2021.

Won, J., Mendis, C., Emer, J. S., and Amarasinghe, S. Waco: Learning workload-aware co-optimization of the format and schedule of a sparse tensor program. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS 2023, pp. 920–934, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450399166. doi: 10.1145/3575693.3575742. URL https://doi.org/10.1145/3575693.3575742.

Ye, Y. and Ji, S. Sparse graph attention networks. *IEEE Transactions on Knowledge and Data Engineering*, 35(1):905–916, 2021.

Ye, Z., Lai, R., Shao, J., Chen, T., and Ceze, L. Sparsetir: Composable abstractions for sparse compilation in deep learning. ASPLOS 2023, pp. 660–678, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450399180. doi: 10.1145/3582016.3582047. URL https://doi.org/10.1145/3582016.3582047.

Zhai, Y., Zhang, Y., Liu, S., Chu, X., Peng, J., Ji, J., and Zhang, Y. Tlp: A deep learning-based cost model for tensor program tuning. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 833–845, 2023.

Zheng, L., Jia, C., Sun, M., Wu, Z., Yu, C. H., Haj-Ali, A., Wang, Y., Yang, J., Zhuo, D., Sen, K., et al. Ansor: Generating {High-Performance} tensor programs for deep learning. In *14th USENIX symposium on operating systems design and implementation (OSDI 20)*, pp. 863–879, 2020.

Zheng, L., Liu, R., Shao, J., Chen, T., Gonzalez, J. E., Stoica, I., and Ali, A. H. Tenset: A large-scale program performance dataset for learned tensor compilers. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. URL https://openreview.net/forum?id=aIfp8kLuvc9.

A Background, and Related Work

Sparse Tensor Algebra. Tensor Algebra Compiler (TACO) (Kjolstad et al. (2017)) was the first compiler capable of generating code for arbitrary compound tensor algebra expressions involving dense and sparse inputs. TACO decomposes a kernel into two abstractions: an iteration graph, which encodes loop dependencies, and a format abstraction, which defines tensor storage as a per-dimension combination of dense or compressed levels. From these structures, the compiler generates a single fused loop nest for the entire expression. Because TACO delegates schedule selection to the user, its performance hinges on manual tuning or exhaustive template-based search. Nevertheless, it introduced the crucial idea that sparse algorithms, schedules, and formats can be decoupled and reconciled during code generation. This idea of decoupling was initially introduced for dense computations in Halide (Ragan-Kelley et al. (2017); Adams et al. (2019)).

Sparse Tensor Cost Modeling and Auto-Tuning. WACO Won et al. (2023) extends this paradigm by automating both format and schedule selection for sparse matrix computations, building on the foundational abstractions introduced by TACO (Kjolstad et al. (2017)). WACO employs a Sparse Convolutional Neural Network (SCNN) feature extractor (Graham & Van der Maaten (2017)), WACONet, trained on approximately 2.1 million $\langle$`matrix`, `schedule`, `runtime`$\rangle$ data tuples per kernel per model, covering multiple sparse matrix kernels such as SpMV, SpMM, and SDDMM. At runtime, the input matrix is embedded and used to query a k-nearest-neighbor graph over a pre-sampled set of schedules to select the fastest candidate via on-device profiling. By coupling storage and traversal decisions, WACO outperforms the default TACO implementation. Unlike TACO, however, WACO's cost model is kernel-specific, and each kernel is trained and tuned independently.

Learned Cost Models for Compiler Optimization. Learned cost models have also been extensively studied in the context of optimizing dense computations. Early approaches such as those in TVM (Chen et al. (2018)) leveraged gradient-boosted trees (e.g., XGBoost (Chen & Guestrin (2016))) to model schedule performance. More recent systems have explored the use of deep learning to improve generalization and prediction accuracy (Baghdadi et al. (2021); Zheng et al. (2020); Sasaki et al. (2022); Zhai et al. (2023); Kaufman et al. (2021)). These models enable effective search over large schedule spaces, often outperforming traditional auto-tuners by using learned performance predictors to guide optimizations. The underlying principles such as feature-based embeddings, surrogate modeling, and data-driven schedule selection in optimizing dense computations, have strong parallels to sparse cost modeling. Leveraging these ideas, alongside new techniques, may offer promising directions for overcoming the current limitations in sparse matrix optimization frameworks.