Portable Dynamic Tiling for Sparse Tensor Accelerators

Sai Gautham Ravipati
sgautham@stanford.edu
Stanford University, USA

Fredrik Kjolstad
kjolstad@stanford.edu
Stanford University, USA

Priyanka Raina
praina@stanford.edu
Stanford University, USA

Olivia Hsu
owh@andrew.cmu.edu
Stanford University/Carnegie Mellon University, USA

ABSTRACT

This work introduces a compiler and its abstraction for dynamic tiling of sparse tensor applications on accelerators. Our abstraction expresses dynamic tiling as data-dependent updates to global iteration-space tile boundaries based on structural tile properties. Because these updates depend only on sparse structural metrics (e.g., nonzero counts) rather than expression-specific traversal logic, tiling policies are decoupled from tensor expressions, data layouts, and hardware backends. This view of tiling enables defining a strategy once and reusing or varying it across workloads and code generators. We validate the abstraction by generating a Dynamic Reflexive Tiling-style algorithm that is within 1% of the original handwritten baseline, demonstrating that policy-level specifications can match expert-designed implementations.

1 INTRODUCTION

Recent hardware trends show a growing number of accelerators targeting data-dependent and irregular workloads [5, 20], such as sparse tensor algebra [7, 8, 15, 16, 19, 22, 26, 27]. These accelerators are typically constrained by limited on-chip memory and rely on host CPUs to tile and orchestrate data movement across the memory hierarchy. While prior work [1, 6, 11, 23, 25] has addressed this challenge for mapping dense workloads to accelerators, the problem is substantially more complex for sparse tensor applications. First, tiling code must operate over sparse loop nests [14], which are significantly harder to optimize than dense ones. Second, performance depends strongly on the tiling algorithm itself, since tile structure directly affects load balance and accelerator utilization. As a result, naive tiling of sparse data usually produces unevenly populated tiles, leading to severe load imbalance and under-utilization.

Prior work [2, 3, 10, 13] on sparse tensor orchestration partly addresses the first challenge, but relies on static tiling, which produces unevenly populated tiles for sparse data. To address this, hand-designed dynamic tiling algorithms [9, 12, 17, 18, 21, 24] have been proposed, which adapt tile shapes based on data properties at runtime. These algorithms adapt tiles based on the tiled-data properties (e.g. number of nonzeros) but are typically specialized to a particular application (e.g., SpMM), sparsity pattern, and hardware accelerator, making them brittle, non-composable, and difficult to reuse. Adapting an existing dynamic tiling strategy to a new application or accelerator often requires rewriting substantial low-level traversal and control logic, even when the tiling policy itself is unchanged.

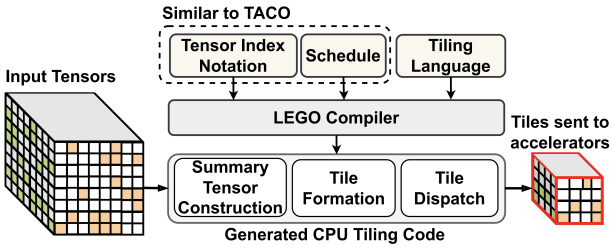

Figure 1: Overview of the LEGO compiler system.

To address this gap, we propose **LEGO**, a compiler that generates dynamic tiling algorithms for sparse tensor accelerators. LEGO decouples tiling policy from tensor algebra expressions, sparse data structure formats, and hardware-specific code generation by introducing an abstraction in which users specify tiling policies as data-dependent actions that update iteration-space tile boundaries under user-defined sparse structural property conditions. LEGO is portable along two axes: (1) tiling policies are reusable across tensor expressions, data layouts, and hardware backends and (2) dispatch code, which converts tile data into accelerator-specific layouts is structurally separated from the tiling logic and reusable across tiling policies. Together, these properties enable rapid generation of tiles that satisfy architectural constraints to improve accelerator utilization, and significantly reduce the effort required to deploy dynamic tiling algorithms across applications and accelerators, which we believe is general enough to handle the space of prior dynamic tiling algorithms proposed.

2 SYSTEM OVERVIEW

Fundamentally, dynamic tiling algorithms for sparse tensor programs adapt tile boundaries based on data-dependent properties in order to satisfy architectural constraints. A tile represents a subregion over iteration indices whose boundaries are adjusted based on structural properties of a tensor. These structural properties—such as nonzeros (nnz) or active rows—represent the sparsity structure and are used to estimate architectural metrics like buffer occupancy or memory traffic. A typical dynamic tiling algorithm begins with an initial tile, evaluates these properties, and iteratively adjusts boundaries until architectural constraints are satisfied. Prior systems [9, 12, 17, 18, 21] tightly couple boundary adaptation logic,

LATTE '26, March 23, 2026, Pittsburgh, PA, USA

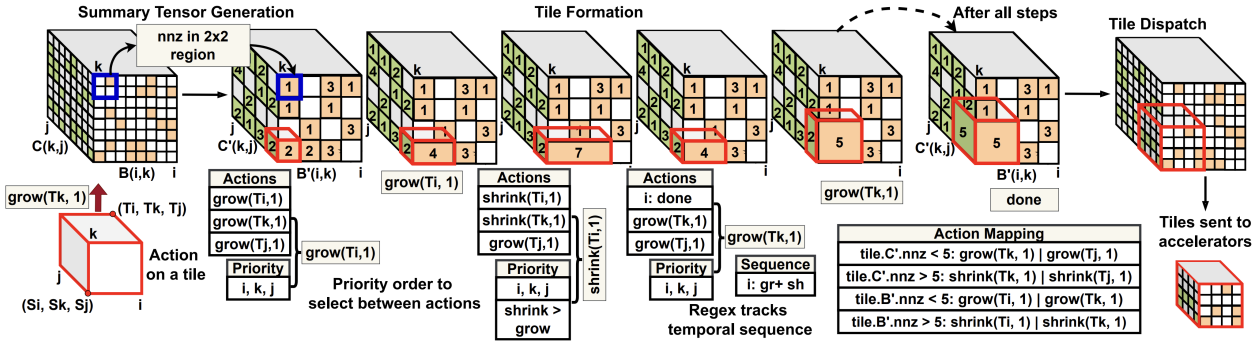

Figure 2: Dynamic tiling for SpGEMM, $A(i,j) = B(i,k) \times C(k,j)$. The global iteration space spans the indices $[i,k,j]$, with matrices residing on the $[i,k]$ and $[k,j]$ faces. The tile boundaries $[T_i, T_k, T_j]$, shown in red, are adapted at each step based on nnz counts.

structural property evaluation, and sparse traversal strategy into monolithic implementations.

Leveraging the generalized view of tiling described above, LEGO introduces a high-level language that allows performance engineers to specify dynamic tiling policies to generate the monolithic code as its output. Instead of hard-coding how tiles are constructed, users specify what makes a tile acceptable using conditions on structural properties and actions that update iteration-space boundaries. The compiler then composes operand-level specifications across tensors and synthesizes a complete tiling algorithm for a given expression. LEGO's design enables reuse of a tiling strategy across different expressions and exploration of multiple tiling strategies for the same expression, without rewriting low-level logic.

The code generated by LEGO retains the structure of prior dynamic tiling systems, but generalizes it across tensor expressions and tiling strategies. Because adapting tile shapes requires repeated traversal of sparse data structures—which can be expensive—the compiler first generates code to build summary tensors: coarser sparse representations that store structural statistics over uniform regions of the original tensors, similar to micro-tiling in prior work [12, 21]. Tiles are then formed by operating over these summary tensors, evaluating structural properties, and applying actions to adjust tile boundaries. Once a tile satisfies the architectural constraints, its boundaries are projected back onto the original tensors to gather data and dispatch the tile to the accelerator. Fig. 1 illustrates the overall system and interaction between generated code fragments. LEGO targets host-side CPU orchestration: it generates the software that constructs summary tensors, discovers tiles, and dispatches them. Hardware based tiling (e.g., RTL generation) is outside the current scope, though the abstraction is in principle compatible with systems that tile on-hardware [21], and adapting the tiling language to emit hardware descriptions is a direction for future work. The remainder of this section focuses on tile-formation code generation from the input language.

2.1 Tile Formation via Dynamic Tiling

Unlike traditional dynamic tiling algorithms, which hard-code tile formation as hand-written sparse iteration loop nests over summary tensors with embedded control logic, LEGO lifts tile formation into a reusable, operand-level abstraction that can be composed across tensor expressions. Tile formation is expressed as condition-to-action mappings from structural properties to updates of tile boundaries. The compiler composes these rules across operands and synthesizes sparse co-iteration loop nests to collect structural properties, and predicates that select boundary update actions.

Fig. 2 illustrates a Dynamic Reflexive Tiling (DRT) [21]-style algorithm applied to SpGEMM. In DRT, tile boundaries are updated adaptively based on the number of nonzeros within each tile. In our input language, this behavior for a general matrix is expressed using condition-to-action rules and regular expressions that constrain the temporal sequence of actions.

```
alg1(X(i1,i2), cap, δ): # gr: grow, sh: shrink
    tile.X.nnz < cap - δ -> gr(Ti1,1) | gr(Ti2,1)
    tile.X.nnz > cap     -> sh(Ti1,1) | sh(Ti2,1)
    i1: [gr(Ti1,1)+ sh(Ti1,1)]; i2: [gr(Ti2,1)+ sh(Ti2,1)]
```

This specification can be reused across tensor operands and expressions (e.g., `B.bind(alg1,capacityB,deltaB)`), enabling reuse of tiling policies across applications. The action-mapping table in Fig. 2 is derived by binding operands of SpGEMM to this algorithm. The language also supports other rank-specific structural properties such as `tile.B.nne(i)` and `tile.B.nne(k)`, which encode row- or column-level sparsity. These constructs enable users to express architectural constraints for diverse tiling algorithms. The compiler automatically generates code to collect these properties during summary tensor construction.

To resolve conflicts when multiple actions are feasible, a user-defined priority order selects among same-direction updates, while `shrink` actions take precedence when directions conflict. Regular expressions are used to capture temporal patterns common in prior algorithms [12, 21] (e.g., "grow repeatedly, then shrink once"). Critically, these rules specify *what* makes a tile complete, not how to traverse sparse data structures to find one. The compiler derives the traversal structure and state management needed to evaluate conditions and apply updates efficiently.

3 RESULTS

To evaluate this abstraction, we generate a DRT [21]-style dynamic tiling algorithm and compare accelerator runtimes against DRT on the ExTensor simulator [8] (Fig. 3). Tiles produced by the generated algorithm run within 1% of the original code, indicating

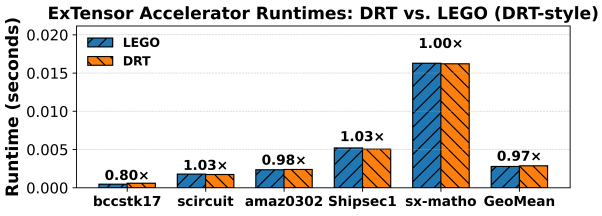

Figure 3: ExTensor runtimes on Suitesparse [4] data.

that the abstraction preserves the essential algorithmic behavior of expert-designed dynamic tilers. Remaining differences arise from variations in iteration-space traversal strategy [14], which we leave as a future implementation improvement. While the current evaluation targets a single accelerator backend, the separation between tiling policy and dispatch code is designed to enable re-targeting to other accelerator backends.

4 CONCLUSION

LEGO demonstrates that dynamic tiling for sparse workloads can be expressed as a reusable, policy-level abstraction rather than as hand-written, expression-specific algorithms. By separating what constitutes a valid tile from how tiles are discovered, LEGO enables portable, composable tiling strategies that match the performance of expert-designed implementations. We believe this shift—from bespoke dynamic tilers to compiler-generated ones—opens the door to more general orchestration frameworks for irregular, data-dependent workloads and reduces a key barrier to deploying specialized accelerators in practice. For accelerator architects, LEGO can also serve as an exploration tool: by rapidly generating and evaluating different tiling strategies, designers can better understand the trade-offs between computation cost, tiling cost and host-side data movement cost, thereby better informing architectural decisions early in the design process.

REFERENCES

[1] Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, C. K. Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Shunting Zhang, Michael Suo, Phil Tillet, Xu Zhao, Eikan Wang, Keren Zhou, Richard Zou, Xiaodong Wang, Ajit Mathews, William Wen, Gregory Chanan, Peng Wu, and Soumith Chintala. 2024. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 929–947. https://doi.org/10.1145/3620665.3640366

[2] Manya Bansal, Olivia Hsu, Kunle Olukotun, and Fredrik Kjolstad. 2023. Mosaic: An Interoperable Compiler for Tensor Algebra. *Proc. ACM Program. Lang.* 7, PLDI, Article 122 (June 2023), 26 pages. https://doi.org/10.1145/3591236

[3] Stephen Chou, Fredrik Kjolstad, and Saman Amarasinghe. 2018. Format Abstraction for Sparse Tensor Algebra Compilers. *Proc. ACM Program. Lang.* 2, OOPSLA, Article 123 (October 2018), 30 pages.

[4] Timothy A. Davis and Yifan Hu. 2011. The university of Florida sparse matrix collection. *ACM Trans. Math. Softw.* 38, 1, Article 1 (Dec. 2011), 25 pages. https://doi.org/10.1145/2049662.2049663

[5] Xingran Du, Joel S. Emer, and Daniel Sanchez. 2025. Hopps: Leveraging Sparsity to Accelerate Automata Processing. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (Rotterdam, Netherlands) *(ASPLOS '25)*. Association for Computing Machinery, New York, NY, USA, 96–111. https://doi.org/10.1145/3676642.3736126

[6] Kayvon Fatahalian, Daniel Reiter Horn, Timothy J. Knight, Larkhoon Leem, Mike Houston, Ji Young Park, Mattan Erez, Manman Ren, Alex Aiken, William J. Dally, and Pat Hanrahan. 2006. Sequoia: programming the memory hierarchy. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing* (Tampa, Florida) *(SC '06)*. Association for Computing Machinery, New York, NY, USA, 83–es. https://doi.org/10.1145/1188455.1188543

[7] Courtney Golden, Axel Feldmann, Joel Emer, and Daniel Sanchez. 2025. Quartz: A Reconfigurable, Distributed-Memory Accelerator for Sparse Applications. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture (MICRO '25)*. Association for Computing Machinery, New York, NY, USA, 929–943. https://doi.org/10.1145/3725843.3756035

[8] Kartik Hegde, Hadi Asghari-Moghaddam, Michael Pellauer, Neal Crago, Aamer Jaleel, Edgar Solomonik, Joel Emer, and Christopher W. Fletcher. 2019. ExTensor: An Accelerator for Sparse Tensor Algebra. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture* (Columbus, OH, USA) *(MICRO '52)*. Association for Computing Machinery, New York, NY, USA, 319–333. https://doi.org/10.1145/3352460.3358275

[9] Changwan Hong, Aravind Sukumaran-Rajam, Israt Nisa, Kunal Singh, and P. Sadayappan. 2019. Adaptive sparse tiling for sparse matrix multiplication. In *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming* (Washington, District of Columbia) *(PPoPP '19)*. Association for Computing Machinery, New York, NY, USA, 300–314. https://doi.org/10.1145/3293883.3295712

[10] Olivia Hsu, Alexander Rucker, Tian Zhao, Varun Desai, Kunle Olukotun, and Fredrik Kjolstad. 2025. Stardust: Compiling Sparse Tensor Algebra to a Reconfigurable Dataflow Architecture. In *Proceedings of the 23rd ACM/IEEE International Symposium on Code Generation and Optimization* (Las Vegas, NV, USA) *(CGO '25)*. Association for Computing Machinery, New York, NY, USA, 628–643. https://doi.org/10.1145/3696443.3708918

[11] Yuka Ikarashi, Gilbert Louis Bernstein, Alex Reinking, Hasan Genc, and Jonathan Ragan-Kelley. 2022. Exocompilation for productive programming of hardware accelerators. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (San Diego, CA, USA) *(PLDI 2022)*. Association for Computing Machinery, New York, NY, USA, 703–718. https://doi.org/10.1145/3519939.3523446

[12] Anirudh Jain, Pulkit Gupta, and Thomas M. Conte. 2025. RASSM: Residue-based Acceleration of Single Sparse Matrix Computation via Adaptive Tiling. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1* (Rotterdam, Netherlands) *(ASPLOS '25)*. Association for Computing Machinery, New York, NY, USA, 907–923. https://doi.org/10.1145/3669940.3707219

[13] Fredrik Kjolstad, Peter Ahrens, Shoaib Kamil, and Saman Amarasinghe. 2019. Tensor Algebra Compilation with Workspaces. In *2019 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. 180–192. https://doi.org/10.1109/CGO.2019.8661185

[14] Fredrik Kjolstad, Shoaib Kamil, Stephen Chou, David Lugato, and Saman Amarasinghe. 2017. The tensor algebra compiler. *Proceedings of the ACM on Programming Languages* 1, OOPSLA (2017), 1–29.

[15] Kalhan Koul, Olivia Hsu, Yuchen Mei, Sai Gautham Ravipati, Maxwell Strange, Jackson Melchert, Alex Carsello, Taeyoung Kong, Po-Han Chen, Huifeng Ke, Keyi Zhang, Qiaoyi Liu, Gedeon Nyengele, Zhouhua Xie, Akhilesh Balasingam, Jayashree Adivarahan, Ritvik Sharma, Christopher Torng, Joel S. Emer, Fredrik Kjolstad, Mark Horowitz, and Priyanka Raina. 2025. Onyx: A 12-nm Programmable Accelerator for Dense and Sparse Applications. *IEEE Journal of Solid-State Circuits* (2025), 1–13. https://doi.org/10.1109/JSSC.2025.3604724

[16] Kalhan Koul, Maxwell Strange, Jackson Melchert, Alex Carsello, Yuchen Mei, Olivia Hsu, Taeyoung Kong, Po-Han Chen, Huifeng Ke, Keyi Zhang, Qiaoyi Liu, Gedeon Nyengele, Akhilesh Balasingam, Jayashree Adivarahan, Ritvik Sharma, Zhouhua Xie, Christopher Torng, Joel Emer, Fredrik Kjolstad, Mark Horowitz, and Priyanka Raina. 2024. Onyx: A 12nm 756 GOPS/W Coarse-Grained Reconfigurable Array for Accelerating Dense and Sparse Applications. In *2024 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*. 1–2. https://doi.org/10.1109/VLSITechnologyandCir46783.2024.10631383

[17] Süreyya Emre Kurt, Aravind Sukumaran-Rajam, Fabrice Rastello, and P. Sadayyappan. 2020. Efficient Tiled Sparse Matrix Multiplication through Matrix Signatures. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–14. https://doi.org/10.1109/SC41405.2020.00091

[18] Xintong Li, Zhiyao Li, and Mingyu Gao. 2025. HYTE: Flexible Tiling for Sparse Accelerators via Hybrid Static-Dynamic Approaches. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA '25)*. Association for Computing Machinery, New York, NY, USA, 1613–1626. https://doi.org/10.1145/3695053.3731044

[19] Zhiyao Li, Jiaxiang Li, Taijie Chen, Dimin Niu, Hongzhong Zheng, Yuan Xie, and Mingyu Gao. 2023. 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* (Vancouver, BC, Canada) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 747–761. https://doi.org/10.1145/3575693.3575706

[20] Quan M. Nguyen and Daniel Sanchez. 2021. Fifer: Practical Acceleration of Irregular Applications on Reconfigurable Architectures. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture* (Virtual Event, Greece) *(MICRO '21)*. Association for Computing Machinery, New York, NY, USA, 1064–1077. https://doi.org/10.1145/3466752.3480048

[21] Toluwanimi O. Odemuyiwa, Hadi Asghari-Moghaddam, Michael Pellauer, Kartik Hegde, Po-An Tsai, Neal C. Crago, Aamer Jaleel, John D. Owens, Edgar Solomonik, Joel S. Emer, and Christopher W. Fletcher. 2023. Accelerating Sparse Data Orchestration via Dynamic Reflexive Tiling. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (Vancouver, BC, Canada) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 18–32. https://doi.org/10.1145/3582016.3582064

[22] Subhankar Pal, Jonathan Beaumont, Dong-Hyeon Park, Aporva Amarnath, Siying Feng, Chaitali Chakrabarti, Hun-Seok Kim, David Blaauw, Trevor Mudge, and Ronald Dreslinski. 2018. OuterSPACE: An outer product based sparse matrix multiplication accelerator. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 724–736.

[23] Jonathan Ragan-Kelley, Andrew Adams, Dillon Sharlet, Connelly Barnes, Sylvain Paris, Marc Levoy, Saman Amarasinghe, and Frédo Durand. 2017. Halide: decoupling algorithms from schedules for high-performance image processing. *Commun. ACM* 61, 1 (Dec. 2017), 106–115. https://doi.org/10.1145/3150211

[24] Lucas Wilkinson, Kazem Cheshmi, and Maryam Mehri Dehnavi. 2023. Register Tiling for Unstructured Sparsity in Neural Network Inference. *Proc. ACM Program. Lang.* 7, PLDI, Article 188 (June 2023), 26 pages. https://doi.org/10.1145/3591302

[25] Xuan Yang, Mingyu Gao, Qiaoyi Liu, Jeff Setter, Jing Pu, Ankita Nayak, Steven Bell, Kaidi Cao, Heonjae Ha, Priyanka Raina, Christos Kozyrakis, and Mark Horowitz. 2020. Interstellar: Using Halide's Scheduling Language to Analyze DNN Accelerators. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 369–383. https://doi.org/10.1145/3373376.3378514

[26] Guowei Zhang, Nithya Attaluri, Joel S. Emer, and Daniel Sanchez. 2021. Gamma: leveraging Gustavson's algorithm to accelerate sparse matrix multiplication. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA) *(ASPLOS '21)*. Association for Computing Machinery, New York, NY, USA, 687–701. https://doi.org/10.1145/3445814.3446702

[27] Zhekai Zhang, Hanrui Wang, Song Han, and William J Dally. 2020. SpArch: Efficient architecture for sparse matrix multiplication. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 261–274.