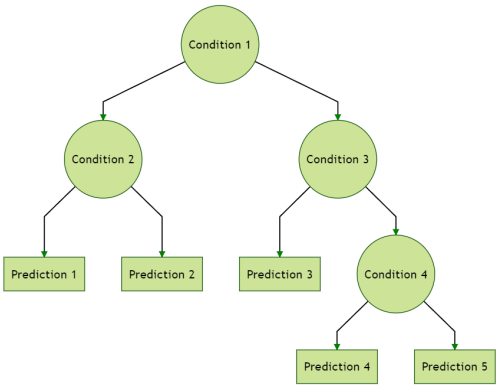
Accelerating AI Inference Using Packed Layouts

Applicant(s): Vidush Singhal & Logan Anderson
Institution: Purdue University
Faculty Advisor: Milind Kulkarni, Aravind Machiry

Introduction

Artificial Intelligence is omnipresent and has far-reaching impacts on the lives of millions of people. However, modeling such intelligence comes with major drawbacks. Once an AI model is trained, inference requires frequent traversals of the machine learning model to compute relevant outputs. Since an inference model, once trained, does not require any further training [28], we can use the same model for each query (for example, decision tree inference [5, 14], search query in chat-bots, etc.). As a result, we must ensure that model inference is fast and efficient (requires lesser energy). To this end, our proposal aims to accelerate and optimize the inference of specific machine learning models, to positively impact the lives of all individuals who will benefit from the power of AI. We identify memory as one of the major bottlenecks for various machine learning models [18, 19, 4, 20] and aim to optimize the data layout of such models to enhance inference time and increase efficiency. ML inference is being deployed on various embedded systems [1, 9, 4] for real-time data analysis; these systems are usually constrained in the amount of memory available for computation and suffer from cache locality. Our proposal aims to enhance the locality of inference of such models and, in the long term, provide support for parsing existing models via our compiler tool-chain and support a GPU backend for faster inference time. We envision developing a compiler tool-chain similar to many existing ML compilers [27, 6] built upon the Gibbon [37, 36, 30] compiler for accelerating machine learning models to provide a general ecosystem for accelerating inference of ML models.



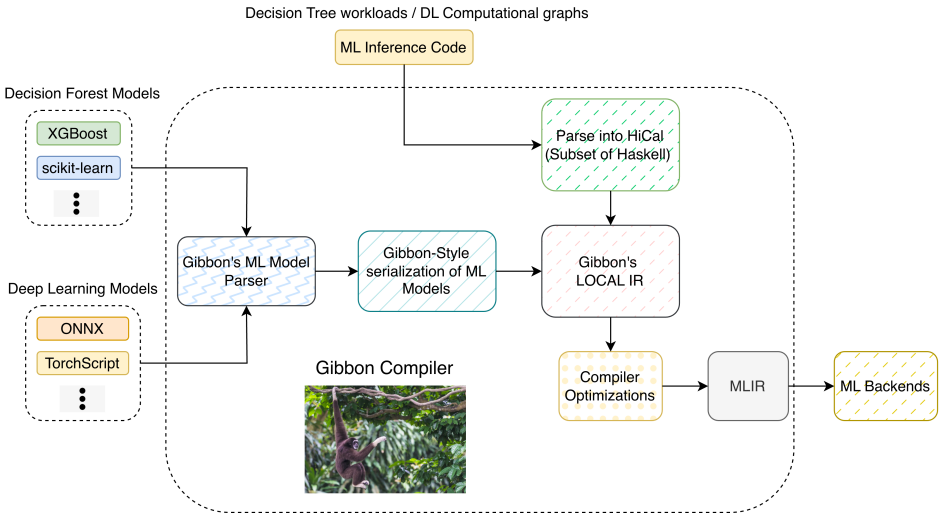
(a) Decision Tree

```
type Condition = Bool
type Prediction = Int
data DecisionTree = Cond Condition DecisionTree DecisionTree
                  | Pred Prediction
```

(b) Haskell ADT

Cond	T	Cond	F	Pred	10	Pred	20	Cond	T	Pred	30	....

(c) Serialized Decision Tree



(d) Proposed Design of the Compiler Framework

Figure 1: A Decision Tree Model (left) including one possible Haskell ADT of the decision tree with its corresponding Gibbon serialization with dummy values. Overall Pipeline of the proposed compiler ecosystem (right).

Background

Gibbon [37, 36, 30] is an experimental compiler that specializes in accelerating tree traversals and operates over a simple high-level language. Programs are written in a subset of Haskell (Hi-Cal), which supports algebraic data types and pattern matching. The compiler serializes ADTs by fixing a traversal order, allowing it to traverse those data types efficiently. This approach is substantially more efficient than standard pointer-based tree traversals due to better spatial locality. Traversals written in Gibbon tend to have a high degree of locality and subsequent performance improvements due to predictable access patterns.

Data layout [7] is a crucial determinant of a program's runtime performance. Poorly organized data can lead to excessive cache misses and unnecessary memory traffic, resulting in substantial slowdowns. This challenge is especially apparent in programs that operate over tree-structured data—such as web-browser engines, numerical simulation pipelines, compiler passes that traverse abstract syntax trees (ASTs) [31, 37, 30], and machine-learning systems that rely on decision forests [20, 18]. These workloads are considered irregular because tree structures do not exhibit the predictable, array-like memory layouts that modern hardware is optimized for. As a result, traversals over tree-shaped data often produce irregular, hard-to-predict memory access patterns.

Improving the data layout of such programs can yield significant performance gains. Gibbon addresses this need by compiling high-level programs that manipulate tree-like algebraic data types (ADTs) into lower-level programs that operate directly on a serialized, layout-

optimized representation of those ADTs. [37, 36, 30] This spares programmers the burden of manual data-layout engineering while producing code with substantially better spatial locality and faster traversal performance due to more efficient use of the cache hierarchy.

Model inference is the stage in which a trained AI model processes new inputs to produce classification or prediction outputs. Because inference does not modify the model's parameters, it typically dominates the runtime cost of deployed systems—for example, chatbots repeatedly performing forward passes on incoming queries. Many widely used machine-learning methods rely on tree-structured computation, including decision-forest models such as XGBoost, etc. [5, 20, 18, 19, 25, 8, 39], TensorFlow-Decision Forest [12], LightGBM [14], Tree-LSTMs [33], etc. In tree boosting, a large model is constructed as a sequence of increasingly refined weak learners, and inference consists of a forward traversal across these trees using a vector-valued input. At internal nodes, the computation determines the traversal path; at leaf nodes, the model combines the input vector with stored parameters or embeddings to produce an output. Because Gibbon's serialized data layouts yield highly efficient tree traversals with predictable memory access patterns, models of this form can benefit substantially when compiled through Gibbon. By reducing cache misses and improving traversal locality, Gibbon has the potential to accelerate inference for a broad class of tree-structured machine-learning models.

Long-term vision of the proposed work / Proposed Ideas

We plan to extend the Gibbon compiler with new capabilities to support the compilation and optimization of machine-learning models. In the near term, our focus is on accelerating decision trees and tree-ensemble models. Because these models are inherently tree-structured, serializing them and enabling direct computation over their serialized representations can significantly improve spatial locality. This is particularly beneficial for embedded platforms, where cache capacity and main memory are severely constrained. In addition, we plan to enhance the Gibbon compiler to automatically identify opportunities for parallelism and SIMD execution during inference.

We plan to have a GPU backend that enables offloading highly parallel inference workloads to GPUs. As a longer-term goal, we also plan to explore using Gibbon to serialize dense, tensor-like computation graphs in order to improve spatial locality for deep learning models by extending Gibbon's LOCAL semantics [36].

The current Gibbon compiler accepts programs written in a Haskell-like language; to significantly broaden its usability, we will add front-end support for standard machine-learning model formats. In the near term, this includes widely used frameworks for decision tree inference, such as XGBoost and scikit-learn [8]. Over a longer horizon, we will extend Gibbon to support dense tensor computation graphs that dominate modern deep learning workloads. This will require extending Gibbon's front end to parse established model representations such as ONNX and TorchScript, enabling trained models to be automatically ingested and systematically converted into Gibbon's packed, serialized representations.

On the backend, Gibbon already generates LLVM IR. We will extend this pipeline to support MLIR [16], allowing Gibbon to directly target backend stacks that are optimized for machine-learning inference and accelerator execution. Figure 1d shows the overall pipeline of the proposed compiler framework.

We will initially focus on accelerating decision tree inference on CPUs by aggressively exploiting data locality, parallelism, and SIMD execution. Achieving this requires a sequence of concrete, near-term compiler enhancements. Specifically, we will transition from array-of-structs (AoS) to structure-of-arrays (SoA) layouts, lower recursive tree traversals into efficient iterative loops, and enable vectorization within these loops. Together, these steps establish a solid foundation for high-performance inference on modern CPUs.

Building on this foundation, we will pursue a set of longer-term extensions that substantially broaden Gibbon's scope and impact. These include supporting multiple front-end model formats (e.g., XGBoost, scikit-learn, and ONNX), enabling serialization of graph-structured models, introducing efficient tensor abstractions, and implementing a GPU backend—such as CUDA—to offload highly parallel computations to GPUs. In developing GPU support for Gibbon, we will draw inspiration from established EDSLs for parallel computation, including Accelerate and Obsidian [3, 32, 22, 15].

Related Work

We distinguish the related work as belonging to two types of learning based models. The first and more closely related to our proposed work and relatively shorter-term goals target decision tree-based learning, and the second type of learning, called deep learning, is a much longer-term goal that we envision requiring many modifications and additions to the Gibbon compiler. Decision tree-based learning and deep learning are the state-of-the-art techniques [4] in machine learning and dominate competitions hosted on websites such as `https://www.kaggle.com/`.

For data without structure, such as images, deep learning seems to work well; however, for structured data, decision-tree ensembles such as Gradient Boosting or Random Forest seem to work best [4].

There is significant work on accelerating machine learning models that are tree-based. For instance, tree boosting [5] is an effective and widely used machine learning method. Such techniques are widely used in applications such as spam detection in emails. Machine learning algorithms that use trees are deployed on embedded devices where they operate on real-time data [4], requiring faster traversals of the tree ensemble. Tree ensembles are important in applications such as astrophysics [2], pedestrian detection [24], 3D face analysis [11], noise signal analysis [29], nano-partical analysis [17, 38] etc. [4] XGBoost [5] is a tree boosting system that showcases an efficient algorithm for performing tree boosting. It uses techniques such as data compression and block sharding (a form of parallelism) to make tree boosting more scalable. LightGBM [14] is another system to make gradient boosting decision trees more efficient. QuickScorer [18] and V-QuickScorer [19] transform sequential tree evaluation to a more cache-friendly process. They use arrays to represent tree nodes and group similar nodes (for instance, same splitting feature *fk*) together. RapidScorer [39] is a framework to speed

up the scoring process of industry-scale tree ensemble models. Kuan et-al [4] showcase a system to optimize the real-time inference of tree ensembles on embedded devices. TreeBeard [25] is a compiler built using MLIR to generate efficient CPU code for decision tree inference. SilvanForge [26] is a schedule-guided re-targeting compiler for decision tree-based models. Treelite [8] is a tree library toolbox to facilitate easy deployment of models, including accelerating prediction performance.

The above mentioned works often treat the model as a special-case data structure, not as a general-purpose tree traversal program. Prior work [21, 4, 18, 19, 39] on decision tree-based learning has explored layout-optimized representations for decision tree inference, executing predictions directly over array-encoded trees to improve cache locality. However, these approaches are narrowly specialized to decision forests and rely on hand-designed data layouts. Separately, compiler frameworks such as Treebeard [25] and SilvanForge [26] generate optimized inference code by compiling model structure into imperative control flow, effectively eliminating the model as a runtime data structure.

In contrast, our work builds on the Gibbon compiler's ability to execute programs directly over serialized algebraic data types. We propose to treat the machine learning model itself as a packed, serialized tree and to perform inference by traversing this representation directly, without reconstructing pointer-based objects or compiling the model away into specialized code. To our knowledge, this is the first approach to apply general packed-data compilation techniques to the inference of tree-structured machine learning models and building a general-purpose compiler framework to do so. As a proof of concept, we would like to write the decision tree inference program in Gibbon's current front-end language, HiCal. We can encode example decision forests in Gibbon-style packed layouts (for instance, an ADT to represent the decision tree as shown in Figure 1b). Gibbon can *mmap* such an encoding from disk and operate on it to perform inference. Such a representation will perform well due to cache locality and allow us to perform principled compiler optimizations on the generated code via the Gibbon compiler.

Tensor, Graph-based machine learning inference

There are various compilers for optimizing deep learning workloads [27, 6, 34, 35]. Throughout these frameworks, we see a few common themes. These works aim to design a language to represent the computation of ML operators and algorithms at a higher level of abstraction to make it easier for programmers to write programs. They try to separate the actual computation from the scheduling. Works such as Halide [27] initially pioneered this idea. To this end, they want to hide the low-level decisions of scheduling, memory, etc., from the user and leave it to the compiler to generate optimized code. They decouple the computational graph of the model from the actual implementation of the operators. Various works implement their own code for these operators, often in a higher-level language, to perform specific optimizations. They focus on extending target support for many ML models and provide automatic and efficient code generation and scheduling of these algorithms on heterogeneous devices.

Although these works optimize the data layout of tensors in the computation graph, our proposed approach is orthogonal. Tensors are only part of the computation graph, even though tensors are contiguously allocated, various nodes of the computation graph may not share any locality depending on how the allocator allocates memory for the computation graph during runtime. The details of the in-memory representation of the computation graph are implementation-based, often involving pointer-based data structures and heap-allocated memory for nodes in the computation graph.

In the long term, we propose to use Gibbon to serialize graph-based models by implementing front-end support to parse deep learning model formats like ONNX into a Gibbon-style serialized ADT. The operator implementation can be done at a higher-level language like HiCal (Gibbon's front-end language). Since we propose building a compiler framework, we can perform principled optimizations to the generated code via the compiler easily. We would also like to extend the Gibbon backend to support offloading computations to GPUs.

One-year horizon of the project, even if the proposal is a multi-year project

As a short-term goal, we would like to take two directions towards our long-term goal of making Gibbon a compiler ecosystem for accelerating machine learning workloads. As Gibbon is currently well-suited for tree traversals, we would like to accelerate decision tree-based inference models [5, 4]. These models are widely used for structured inputs on diverse hardware platforms and embedded devices, which are severely constrained for memory [4]. Thus, optimizing for cache locality will allow faster inference time. To show a proof of concept, we would like to encode example decision trees in Gibbon's current encoding style, that is, an Algebraic Data Type, and write the decision tree inference algorithm in Haskell. An example Decision Tree and its serialized Gibbon encoding is shown in Figures 1a, 1b and 1c. For starters, we can develop a Shell/Python script to parse model formats like XGBoost, scikit-learn, etc., into a Gibbon-style serialized encoding and see the performance improvements of a packed serialized representation of the model. In addition, we can use the Gibbon compiler to perform various optimizations on the generated code, thus leveraging the benefit of a compiler ecosystem. Another important step toward our eventual goal is to efficiently support tensors in Gibbon. This step will enable us to begin working towards the development of a GPU backend that can work for AI, which largely relies on fast and efficient tensor computations [13].

Strength of the team to achieve the proposal milestones

Vidush is a 5th year PhD student and works on compiler research. He has worked on Marmoset [30], which optimizes the data layout of ADTs in Gibbon. He has significant experience working with the Gibbon [37] compiler. In addition, he has worked on other compiler projects like Orchard [31], which fuses and parallelizes tree traversals, Copse [23], which vectorizes decision forest inference in FHE. In addition, he has experience working with AI compiler tool-chains like MLIR and LLVM. Logan is a 1st year PhD student who has previously worked on SparseAuto[10], an auto-scheduler for tensor computations based on time and space complexity, along with optimizing based on cache locality. Combined, our skills will allow the team to successfully achieve the mentioned goals.

References

[1] C. Banbury, V. J. Reddi, P. Torelli, J. Holleman, N. Jeffries, C. Kiraly, P. Montino, D. Kanter, S. Ahmed, D. Pau, U. Thakker, A. Torrini, P. Warden, J. Cordaro, G. D. Guglielmo, J. Duarte, S. Gibellini, V. Parekh, H. Tran, N. Tran, N. Wenxu, and X. Xuesong. Mlperf tiny benchmark, 2021.

[2] C. Bockermann and J. Buss. Fact-tools-processing high-volume telescope data. *Astronomical Data Analysis Software and Systems XXVI*, 521:584, 2019.

[3] M. M. Chakravarty, G. Keller, S. Lee, T. L. McDonell, and V. Grover. Accelerating haskell array codes with multicore gpus. In *Proceedings of the Sixth Workshop on Declarative Aspects of Multicore Programming*, DAMP '11, page 3–14, New York, NY, USA, 2011. Association for Computing Machinery.

[4] K.-H. Chen, C. Su, C. Hakert, S. Buschjäger, C.-L. Lee, J.-K. Lee, K. Morik, and J.-J. Chen. Efficient realization of decision trees for real-time inference. *ACM Trans. Embed. Comput. Syst.*, 21(6), Oct. 2022.

[5] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA, 2016. Association for Computing Machinery.

[6] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, M. Cowan, H. Shen, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy. Tvm: an automated end-to-end optimizing compiler for deep learning. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, OSDI'18, page 579–594, USA, 2018. USENIX Association.

[7] T. M. Chilimbi, M. D. Hill, and J. R. Larus. Cache-conscious structure layout. *SIGPLAN Not.*, 34(5):1–12, May 1999.

[8] H. Cho and M. Li. Treelite: Toolbox for decision tree deployment. In *Proceedings of the SysML Workshop*, 2018.

[9] R. David, J. Duke, A. Jain, V. J. Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, S. Regev, R. Rhodes, T. Wang, and P. Warden. Tensorflow lite micro: Embedded machine learning on tinyml systems, 2021.

[10] A. Dias, L. Anderson, K. Sundararajah, A. Pelenitsyn, and M. Kulkarni. Sparseauto: An auto-scheduler for sparse tensor computations using recursive loop nest restructuring. *Proc. ACM Program. Lang.*, 8(OOPSLA2), Oct. 2024.

[11] G. Fanelli, M. Dantone, J. Gall, A. Fossati, and L. Van Gool. Random forests for real time 3d face analysis. *International journal of computer vision*, 101(3):437–458, 2013.

[12] M. Guillame-Bert, S. Bruch, R. Stotz, and J. Pfeifer. Yggdrasil decision forests: A fast and extensible decision forests library. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '23, page 4068–4077, New York, NY, USA, 2023. Association for Computing Machinery.

[13] Y. Ji, Q. Wang, X. Li, and J. Liu. A survey on tensor techniques and applications in machine learning. *IEEE Access*, PP:1–1, 10 2019.

[14] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: a highly efficient gradient boosting decision tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 3149–3157, Red Hook, NY, USA, 2017. Curran Associates Inc.

[15] B. Larsen. Simple optimizations for an applicative array language for graphics processors. In *Proceedings of the Sixth Workshop on Declarative Aspects of Multicore Programming*, DAMP '11, page 25–34, New York, NY, USA, 2011. Association for Computing Machinery.

[16] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, and O. Zinenko. Mlir: scaling compiler infrastructure for domain specific computation. In *Proceedings of the 2021 IEEE/ACM International Symposium on Code Generation and Optimization*, CGO '21, page 2–14. IEEE Press, 2021.

[17] P. Libuschewski. Exploration of cyber-physical systems for gpgpu computer vision-based detection of biological viruses. 2017.

[18] C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, N. Tonellotto, and R. Venturini. Quickscorer: A fast algorithm to rank documents with additive ensembles of regression trees. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, page 73–82, New York, NY, USA, 2015. Association for Computing Machinery.

[19] C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, N. Tonellotto, and R. Venturini. Exploiting cpu simd extensions to speed-up document scoring with tree ensembles. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, page 833–836, New York, NY, USA, 2016. Association for Computing Machinery.

[20] M. Madhyastha, K. Lillaney, J. Browne, J. Vogelstein, and R. Burns. Pacset (packed serialized trees): Reducing inference latency for tree ensemble deployment, 2020.

[21] M. Madhyastha, K. Lillaney, J. Browne, J. Vogelstein, and R. Burns. Pacset (packed serialized trees): Reducing inference latency for tree ensemble deployment, 2020.

[22] G. Mainland and G. Morrisett. Nikola: embedding compiled gpu functions in haskell. In *Proceedings of the Third ACM Haskell Symposium on Haskell*, Haskell '10, page 67–78, New York, NY, USA, 2010. Association for Computing Machinery.

[23] R. Malik, V. Singhal, B. Gottfried, and M. Kulkarni. Vectorized secure evaluation of decision forests. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, PLDI 2021, page 1049–1063, New York, NY, USA, 2021. Association for Computing Machinery.

[24] J. Marín, D. Vázquez, A. M. López, J. Amores, and B. Leibe. Random forests of local experts for pedestrian detection. In *2013 IEEE International Conference on Computer Vision*, pages 2592–2599, 2013.

[25] A. Prasad, S. Rajendra, K. Rajan, R. Govindarajan, and U. Bondhugula. Treebeard: An optimizing compiler for decision tree based ml inference. In *Proceedings of the 55th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '22, page 494–511. IEEE Press, 2023.

[26] A. Prasad, S. Rajendra, K. Rajan, R. Govindarajan, and U. Bondhugula. Silvanforge: A schedule-guided retargetable compiler for decision tree inference. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, SOSP '24, page 488–504, New York, NY, USA, 2024. Association for Computing Machinery.

[27] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, page 519–530, New York, NY, USA, 2013. Association for Computing Machinery.

[28] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, R. Chukka, C. Coleman, S. Davis, P. Deng, G. Diamos, J. Duke, D. Fick, J. S. Gardner, I. Hubara, S. Idgunji, T. B. Jablin, J. Jiao, T. S. John, P. Kanwar, D. Lee, J. Liao, A. Lokhmotov, F. Massa, P. Meng, P. Micikevicius, C. Osborne, G. Pekhimenko, A. T. R. Rajan, D. Sequeira, A. Sirasao, F. Sun, H. Tang, M. Thomson, F. Wei, E. Wu, L. Xu, K. Yamada, B. Yu, G. Yuan, A. Zhong, P. Zhang, and Y. Zhou. Mlperf inference benchmark, 2020.

[29] F. Saki, A. Sehgal, I. Panahi, and N. Kehtarnavaz. Smartphone-based real-time classification of noise signals using subband features and random forest classifier. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, page 2204–2208. IEEE Press, 2016.

[30] V. Singhal, C. Koparkar, J. Zullo, A. Pelenitsyn, M. Vollmer, M. Rainey, R. Newton, and M. Kulkarni. Optimizing Layout of Recursive Datatypes with Marmoset: Or, Algorithms + Data Layouts = Efficient Programs. In J. Aldrich and G. Salvaneschi, editors, *38th European Conference on Object-Oriented Programming (ECOOP 2024)*, volume 313 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 38:1–38:28, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[31] V. Singhal, L. Sakka, K. Sundararajah, R. Newton, and M. Kulkarni. Orchard: Heterogeneous parallelism and fine-grained fusion for complex tree traversals. *ACM Trans. Archit. Code Optim.*, 21(2), May 2024.

[32] J. Svensson, M. Sheeran, and K. Claessen. Obsidian: a domain specific embedded language for parallel programming of graphics processors. In *Proceedings of the 20th International Conference on Implementation and Application of Functional Languages*, IFL'08, page 156–173, Berlin, Heidelberg, 2008. Springer-Verlag.

[33] K. S. Tai, R. Socher, and C. D. Manning. Improved semantic representations from tree-structured long short-term memory networks, 2015.

[34] P. Tillet, H. T. Kung, and D. Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, MAPL 2019, page 10–19, New York, NY, USA, 2019. Association for Computing Machinery.

[35] N. Vasilache, O. Zinenko, T. Theodoridis, P. Goyal, Z. DeVito, W. S. Moses, S. Verdoolaege, A. Adams, and A. Cohen. Tensor comprehensions: Framework-agnostic high-performance machine learning abstractions, 2018.

[36] M. Vollmer, C. Koparkar, M. Rainey, L. Sakka, M. Kulkarni, and R. R. Newton. Local: a language for programs operating on serialized data. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, page 48–62, New York, NY, USA, 2019. Association for Computing Machinery.

[37] M. Vollmer, S. Spall, B. Chamith, L. Sakka, C. Koparkar, M. Kulkarni, S. Tobin-Hochstadt, and R. R. Newton. Compiling Tree Transforms to Operate on Packed Representations. In P. Müller, editor, *31st European Conference on Object-Oriented Programming (ECOOP 2017)*, volume 74 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:29, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[38] M. Yayla, A. Toma, K.-H. Chen, J. E. Lenssen, V. Shpacovitch, R. Hergenröder, F. Weichert, and J.-J. Chen. Nanoparticle classification using frequency domain analysis on resource-limited platforms. *Sensors*, 19(19), 2019.

[39] T. Ye, H. Zhou, W. Y. Zou, B. Gao, and R. Zhang. Rapidscorer: Fast tree ensemble evaluation by maximizing compactness in data level parallelization. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 941–950, New York, NY, USA, 2018. Association for Computing Machinery.