Accelerating Scientific Workflows with LLM-Driven Compiler Optimizations for Generated High-Performance Hardware

Max Ramstad
Pacific Northwest National Laboratory
Richland, Washington, USA
robert.ramstad@pnnl.gov

Nicolas Bohm Agostini
Pacific Northwest National Laboratory
Richland, WA, USA
nicolas.agostini@pnnl.gov

Antonino Tumeo
Pacific Northwest National Laboratory
Richland, WA, USA
antonino.tumeo@pnnl.gov

CCS Concepts

• **Software and its engineering → Compilers**; • **Hardware → Emerging languages and compilers**; *HW-SW codesign*; • **Computing methodologies → Artificial intelligence**.

Keywords

LLM, MLIR, HLS, EELS, optimizations

1 Introduction

Performance-critical kernels determine throughput, latency, and energy in High-Performance Computing (HPC) and Artificial Intelligence (AI) workloads. Streaming scientific workflows such as machine learning models used in Electron Energy Loss Spectroscopy (EELS) microscopes [3] demand near–real-time processing to guide experiments. At Pacific Northwest National Laboratory (PNNL), scientists develop and deploy such experimental workflows, producing high-throughput data that must be analyzed and visualized quickly to influence experimental decisions. Software-only solutions targeting Central Processing Units (CPUs) or Graphics Processing Units (GPUs) often cannot meet these low-latency requirements, motivating custom accelerators.

High-Level Synthesis (HLS) enables the deployment of custom hardware accelerators on Field-Programmable Gate Arrays (FPGAs) and Application-Specific Integrated Circuits (ASICs), exploiting fine-grained parallelism, pipelining, and low latency [2]. Yet, high-performance HLS designs require expert-crafted transform schedules, a bottleneck in adopting accelerators widely.

The MLIR framework [4] and the SODA toolchain [1, 2] provide modular compilation and hardware-oriented optimizations. However, achieving high-quality schedules still depends heavily on human expertise. We propose using Large Language Models (LLMs) to synthesize MLIR *Transform dialect* [5] schedules, automating high-level code transformations for better HLS outcomes while preserving expert control. This enables faster experimentation, rapid prototyping of scientific applications, and reduced time-to-insight for workflows like EELS microscopy, where low-latency data processing is critical.

2 Methods

We implement an LLM-in-the-loop transformation schedule generator that (i) ingests the original MLIR kernel and compact hardware specifications (e.g., memory sizes/ports, initiation interval, available parallelism), (ii) reasons about locality, parallelism and the user request, and (iii) emits a Transform dialect schedule to be applied to the IR. We achieve semantically correct and relevant schedules with the following elements in the LLM context and prompt:

- `Clarity`: Provide explicit optimization objectives (such as runtime/latency) and constraints.
- `Contextual refinement`: Include clear hardware specifications to limit "noisy" context used by the LLM.
- `Reason-first`: Require structured reasoning, such as chain-of-thoughts [6], before emitting the schedule.
- `Transform dialect primitives`: Include a library of transform dialect operations and use case examples allows for more accurate and reproducible code generation.

Figure 1 summarizes our flow. Given MLIR code and hardware specifications, the LLM produces a schedule that the compiler applies to generate an optimized kernel.

2.1 Challenges and Limitations

While LLM-aided optimization presents exciting prospects, several challenges persist:

- `Code correctness`: Ensuring generated MLIR transformation schedules are syntactically valid and free of bugs remains critical.
- `Optimization quality`: LLMs may produce suboptimal transformations due to limited understanding of nuanced hardware behaviors.
- `Optimal prompting`: Crafting prompts that thoroughly capture the optimization needs of complex kernels can be a tedious and involved process.

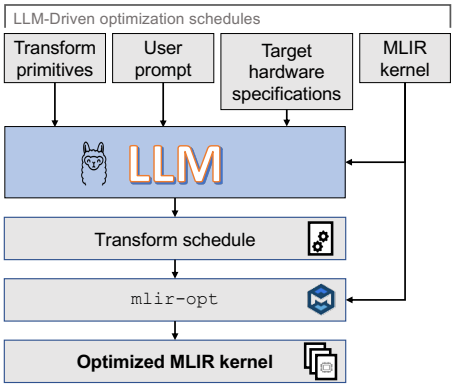

Figure 1: LLM-to-MLIR pipeline: the model receives MLIR and hardware specifications, examples, and the user request to emit a Transform dialect schedule used by the compiler to produce an optimized kernels for HLS.

Strategies for addressing these challenges include integrating human oversight, improving the system's prompt design, and refining the example code given as context.

2.2 Experimental Results and the EELS model

Our primary example in this work comes from optimizing an EELS autoencoder model for FPGA and ASIC targets, targeting low-latency inference. EELS is a powerful technique for nanoscale characterization, offering rich chemical and structural insight from the fine features of its spectra, which can reveal bonding configurations, oxidation states, and subtle variations in electronic structure. However, EELS data is often noisy and high-dimensional, which demand computational models that can perform robust denoising and representation of data prior to classification.

The EELS encoder [3] is a deep neural network with convolutional, ReLU, pooling, and dense layers. It transforms raw microscope signals into compact latent-space tensors that serve as inputs to downstream classification models. This automated representation learning enables rapid identification of material properties such as chemical composition, electronic structure, and bonding environments. Optimizing the EELS encoder is therefore critical for enabling real-time or high-throughput analysis across diverse compute platforms.

Table 1: Runtime speedup due to improved transformation scheduling.

Target	Baseline	Optimized	Speedup
FPGA	87446	37799	2.31x
ASIC	102806	38639	2.66x

From the results presented in Table 1, we can see that the LLM-aided optimization approach yielded significant speedup compared to the baseline MLIR kernel.

3 Conclusion

The integration of LLMs with the MLIR transform dialect marks a promising advance in computation kernel optimization. By automating the generation of high-quality transform dialect code through structured prompts, LLMs significantly reduce development effort while ensuring robust performance across diverse kernels and targets. This approach not only accelerates kernel optimization pipelines but also democratizes access to specialized techniques, allowing developers with varying levels of expertise to contribute to high-performance computing efforts.

As LLMs become increasingly sophisticated, their role in compiler workflows will likely expand, enabling further innovations in kernel design and performance tuning. This work demonstrates that an LLM-driven optimization flow can provide an automated, high-performing alternative to historically manual optimization workflows, paving the way for agentic AI tasks in the compiler and high-performance computing space.

References

[1] Nicolas Bohm Agostini, Serena Curzel, Vinay Amatya, Cheng Tan, Marco Minutoli, Vito Giovanni Castellana, Joseph Manzano, David Kaeli, and Antonino Tumeo. 2022. An MLIR-based Compiler Flow for System-Level Design and Hardware Acceleration. In *2022 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, San Diego, CA, USA, 1–9. https://doi.org/10.1145/3508352.3549424

[2] Nicolas Bohm Agostini, Serena Curzel, Jeff Jun Zhang, Ankur Limaye, Cheng Tan, Vinay Amatya, Marco Minutoli, Vito Giovanni Castellana, Joseph Manzano, David Brooks, Gu-Yeon Wei, and Antonino Tumeo. 2022. Bridging Python to Silicon: The SODA Toolchain. *IEEE Micro* 42, 5 (2022), 78–88. https://doi.org/10.1109/MM.2022.3178580

[3] Jonathan D Hollenbach, Cassandra M Pate, Haili Jia, James L Hart, Paulette Clancy, and Mitra L Taheri. 2023. Embedding theory in ML toward real-time tracking of structural dynamics through hyperspectral datasets. arXiv:2312.05201 [cond-mat.mtrl-sci] https://arxiv.org/abs/2312.05201

[4] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. 2021. MLIR: Scaling Compiler Infrastructure for Domain Specific Computation. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE/ACM, Seoul, Korea (South), 2–14. https://doi.org/10.1109/CGO51591.2021.9370308

[5] Martin Paul Lücke, Oleksandr Zinenko, William S. Moses, Michel Steuwer, and Albert Cohen. 2025. The MLIR Transform Dialect: Your Compiler Is More Powerful Than You Think. 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, 241–254. https://doi.org/10.1145/3696443.3708922

[6] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '22)*. Curran Associates Inc., Red Hook, NY, USA, Article 1800, 14 pages.