Aymane El Jerari
Compiler Class Project

Exploring Pass Selection and Ordering with LLM-Assisted Transformation Schedules using MLIR and IREE

1. Introduction

The goal of this project was to explore the use of Large Language Models (LLMs) to assist with the pass selection and ordering problem in compilers, focusing on the MLIR Transform Dialect. Inspired by recent advancements in LLM-assisted compiler optimizations at Meta, the objective was to leverage the power of LLMs to generate correct and high-performance transformation schedules. The project aimed to use IREE's existing infrastructure for progressive lowering and to create an interface where an LLM could interact with the MLIR Transform Dialect to progressively refine transformation schedules.

This project sought to test the feasibility of using LLMs as "co-pilots" for compiler optimization, a rapidly growing area of research. While the project did not fully achieve its original objectives, it provided invaluable insights and revealed critical challenges in LLM-assisted compilation.

2. Key Challenges and Lessons Learned

1. Ambitious Scope and Time Constraints

> The scope of the project was undeniably ambitious for a month-and-a-half timeline. Bridging three complex systems local LLM setup, the knowledge base and IREE, proved to be a formidable challenge. While I made significant progress on many fronts, fully integrating these components in such a short period was more difficult than expected.

Lesson Learned:

> Breaking down larger projects into smaller, more focused sub-goals is essential. Going forward, I aim to scope this project in a way that allows for progressive iteration and early validation. Despite the initial setback, I now have a much

clearer understanding of how to structure future work in LLM-assisted compilation.

2. Knowledge Base and Retrieval-Augmented Generation (RAG) Setup

A key part of my initial plan was to use a Knowledge Base through Retrieval-Augmented Generation (RAG) to improve the LLM's understanding of the MLIR Transform Dialect. The goal was to enable the LLM to refer to MLIR documentation, and IREE-specific details while generating transformation schedules.

However, setting up a RAG system with a local LLaMA model using LangChain took far more time than anticipated. While I initially aimed to run everything locally, I eventually shifted to using ChatGPT for LLM inference to avoid unnecessary technical overhead. This change introduced new limitations, such as reliance on the context length of the ChatGPT API.

Lesson Learned:

While RAG is a powerful tool, setting it up is non-trivial. Going forward, I plan to dedicate more time to mastering LangChain and LLaMA setup. I now understand the importance of starting with simpler models and progressively scaling up. Despite the challenges, I am still committed to exploring RAG-based LLM assistance for compiler optimizations.

3. Context Size Limitations and LLM Interface Challenges

One of the biggest technical blockers was the context size constraint of the ChatGPT. When I attempted to generate MLIR for models like GPT-2, BERT, or even AlexNet, the size of the MLIR program far exceeded the context window available for LLM inference. This problem was compounded when I attempted to feed MLIR code and error messages back to the LLM for iterative refinement.

To address this, I shifted to simpler feedforward neural networks. However, even with smaller models, the LLM frequently failed to generate correct

transformation schedules. This challenge revealed one of the key limitations of generative models. LLM models can hallucinate incorrect output, particularly in structured domains like compiler transformations.

Lesson Learned:

This experience underscored the importance of modular input design. I now realize that chunking MLIR programs or using strategies like context-aware prompting (e.g., only sending relevant snippets) is critical. I also discovered that LLMs, while powerful, struggle with "hard constraints" like syntax correctness. However, implementing these workarounds lead to a more complex technical implementation which might not necessarily be useful.

4. Learning Curve with IREE

The IREE framework is a sophisticated, full-stack system for MLIR compilation and deployment on multiple backends. While I had prior exposure to MLIR, working with IREE introduced new complexities. Its end-to-end compilation flow requires a solid understanding of MLIR dialects, IR representations, and the backend lowering process.

The progressive lowering from high-level PyTorch tensor dialects down to the linalg dialect was successful, but the larger goal of dynamically injecting LLM-generated transformation schedules into IREE's pipeline proved difficult. Much of my time was spent familiarizing myself with IREE's infrastructure and debugging runtime issues.

Lesson Learned:

Working with large compiler frameworks like IREE requires patience and a clear strategy for onboarding. I now have a deeper appreciation for the complexity of production-grade compilers and the value of strong documentation. While I did not fully master IREE, I learned how to approach large frameworks methodically and gained new debugging skills. I plan to continue exploring IREE in future work, as it remains a critical piece of my research interests.

3. Successes and Accomplishments

While the project did not achieve its initial goal of generating performant, LLM-assisted transformation schedules, it was by no means a failure. The following accomplishments are worth noting:

- **Deeper Knowledge of LLM-Assisted Compilation**: I now have a clearer understanding of the current state of LLM-assisted compiler research. I learned how generative models can fail, where retrieval-augmented generation fits in, and how context size limits affect compiler workflows.
- **Hands-on Experience with IREE and MLIR**: I successfully performed progressive lowering using IREE, taking models from high-level PyTorch IR to lower-level MLIR dialects like linalg. This was a major technical milestone and provided insight into how real-world compilation flows work.
- **Practical RAG System Design**: I gained valuable insight into how RAG systems (using LangChain and LLaMA) can be used to augment LLMs for technical domains like compiler transformations. This knowledge is highly transferable to other research projects involving RAG-based LLMs.
- **Gained New Debugging Strategies**: Working with IREE, MLIR, and LLMs required extensive debugging. I learned how to approach large, multi-component systems, identify root causes, and prioritize issues. This is a skill I will carry forward into future research.

4. Reflection and Future Directions

Although this project did not achieve its original goal, I view it as a major success in terms of personal growth and research direction. Compiler optimization is a field with vast complexity, and working at the intersection of LLMs, MLIR, and IREE revealed challenges that many researchers face.
This project has only deepened my interest in LLM-assisted compiler research. I now have a clearer view of where the key challenges lie and how I can structure future work to achieve success. My next steps are as follows:

- **Continue to Explore LLM-Assisted Compilation**: I plan to refine my approach to LLM-based transformation schedule generation. My next steps include chunking large MLIR inputs and using context-aware prompting. I am also considering a hybrid approach where the LLM generates smaller snippets, which are later validated or corrected using a traditional parser.
- **Master the IREE Framework**: I plan to continue learning the intricacies of IREE, as it is a powerful tool for MLIR compilation. I want to gain a more complete understanding of its internal architecture, from high-level dialects to backend-specific lowering.
- **Iterate on RAG Systems**: I want to continue working on LangChain-based retrieval systems for LLMs. Building a robust knowledge base of MLIR-related content will make future projects more feasible.
- **Scale the Scope of Projects**: This project taught me the importance of scoping research projects effectively. In the future, I plan to break large projects into smaller, modular deliverables that I can iterate on step by step.

5. Conclusion

This project was a valuable learning experience, providing hands-on exposure to the interplay between LLMs, MLIR, and IREE. Although I did not meet the original project goals, I successfully identified critical challenges in the domain of LLM-assisted compilation. I now have a much stronger foundation in this field and remain deeply committed to exploring it as part of my ongoing research. I am excited about the possibilities this field holds. While context size limitations, RAG system setup, and IREE onboarding were formidable hurdles, they have only strengthened my resolve to tackle these challenges. I look forward to continuing this work, building on the knowledge gained from this project, and pushing the boundaries of LLM-assisted compilation research.