PURPOSE
OSCAR Workshop

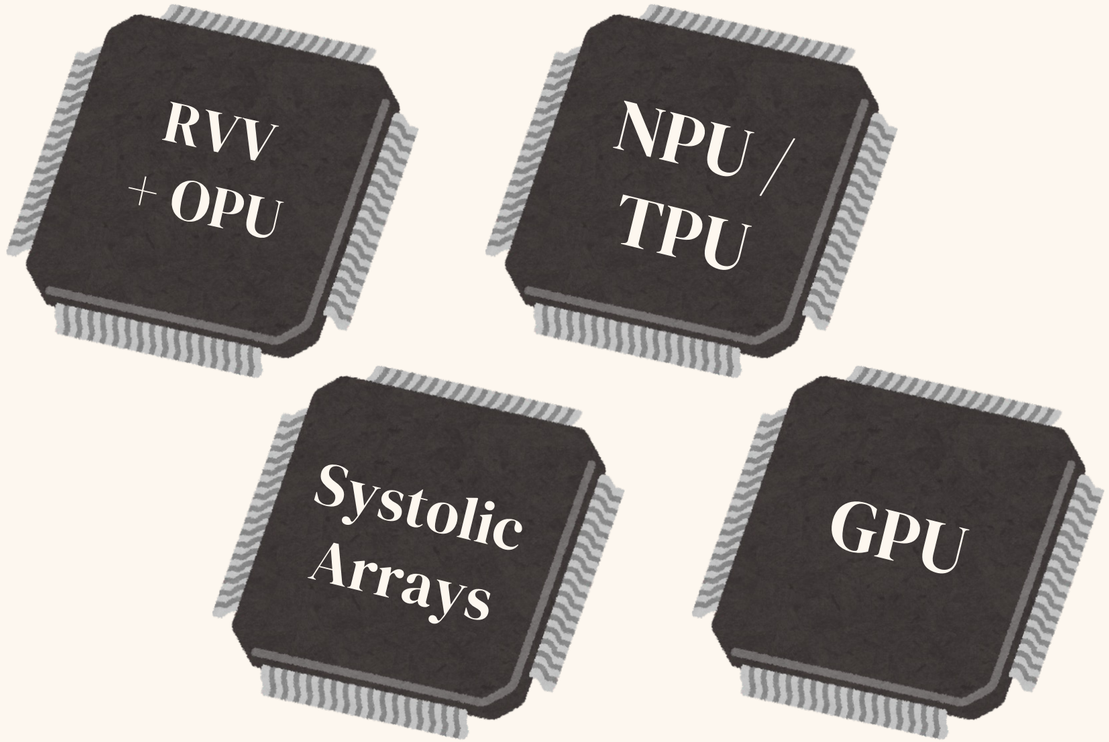
DATE
06/28/2026

Agile Compiler Infrastructure for HW/SW Co-Design

Agustin N. Coppari Hollmann
Kris Dong
Dima Nikiforov
Ashvin Verma
Sophia Yakun Shao

PROJECT PART OF
SLICE

PRESENTED BY
Agustin N. Coppari Hollmann



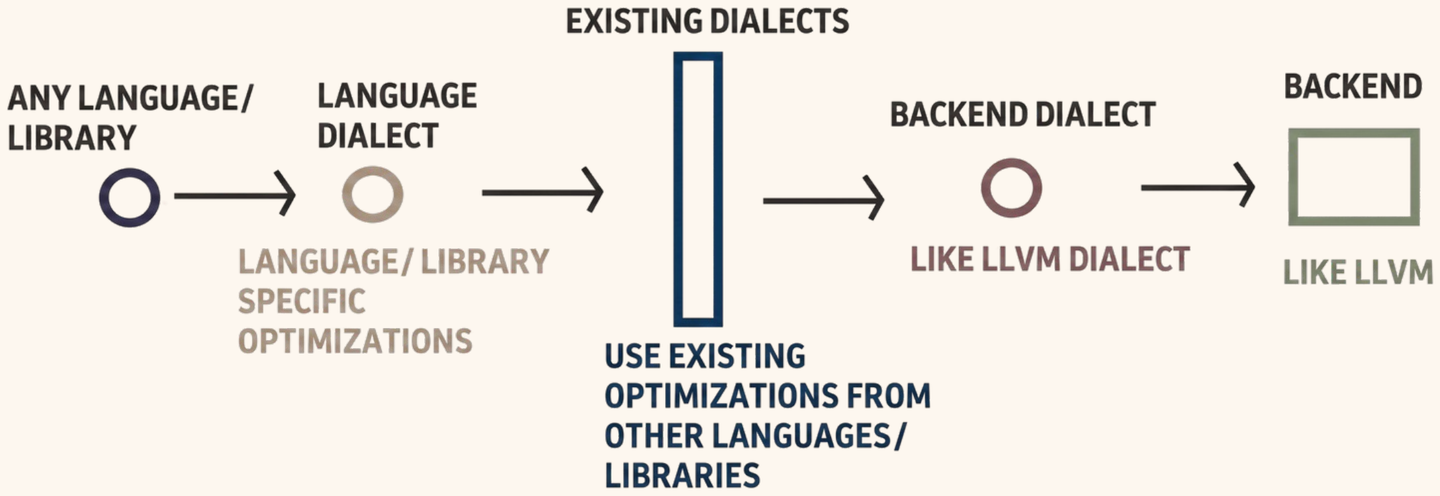
Many Targets → Lots of opportunities

Compilers for AI

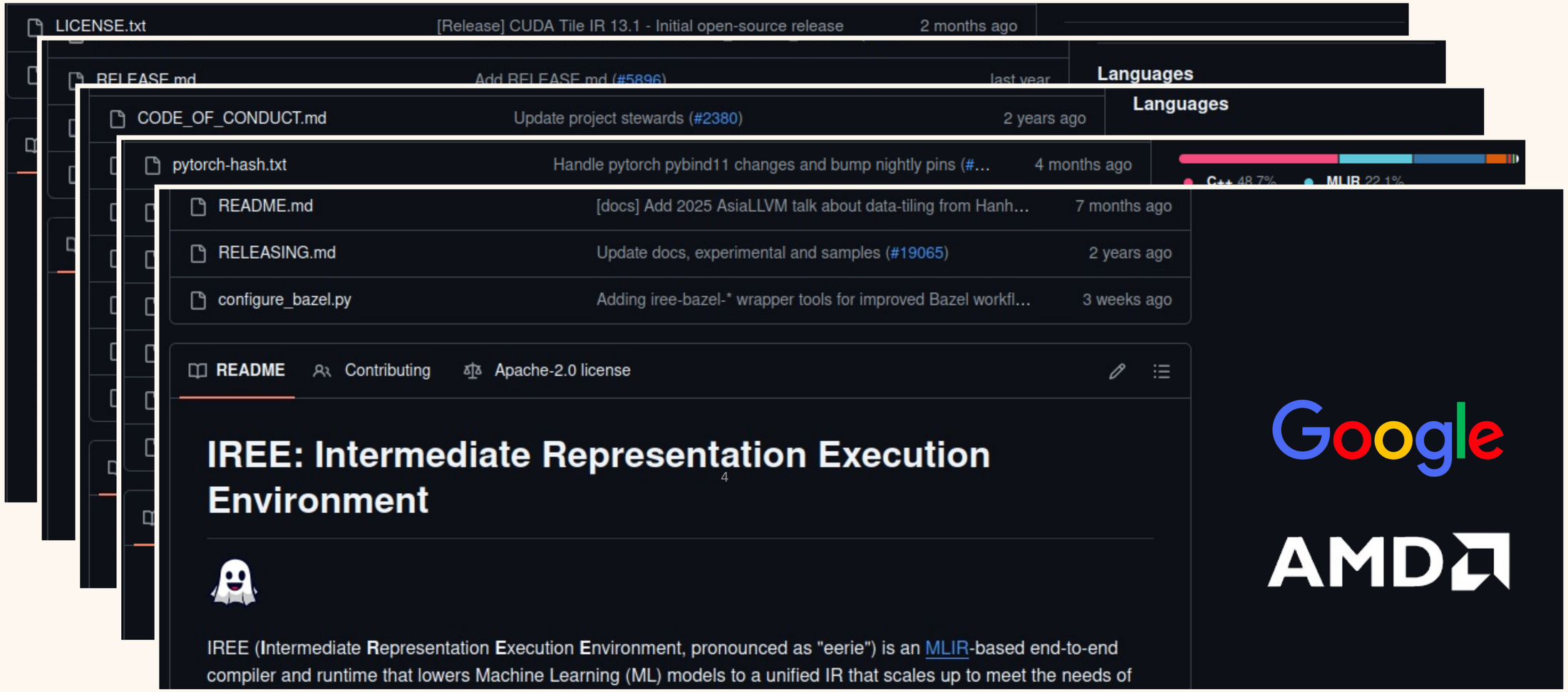
What is MLIR?

Multi-Level Intermediate Representation (MLIR) [1]

- **1 IR** for **many "Dialects"** (custom operations, types, and attributes)
- Supports **progressive lowering** with Dialects "Co-existing"
- (Primary use) Bridges **ML Frameworks to HW Targets**



Where is MLIR used?



Qualcomm Meta Apple
arm Google NVIDIA AMD

MLIR Compilers

Triton

- **Kernel based**
- **Optimized for GPU**
- **Tile-based abstraction**
- **Widely adopted for GPU**
- **CPU implementation much less mature** (even less for RISC-V)

OpenXLA

- **Macro-level optimization**
- **Optimized for TPU**
- **Plug-in** your **propiaretary drivers / backend compiler** as a black-box
- **Not a code generator** on its own for your **ASIC**

IREE [2]

- **Code-gen + Kernels based**
- Can generate **bare-metal exec.**
- Allows you to **include your own pipelines and MLIR dialects**
- **High customization** comes with a **cost of maintaining it**

Less ← → **More**

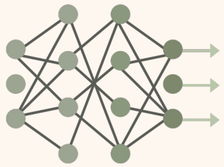
Customizable / Research friendly

Your average experience with ML compilers

- Sit between SW and HW
- Deal with new HW
- Deal with the newest models
- Does their best to do research
- Has to enable SW X on HW Y

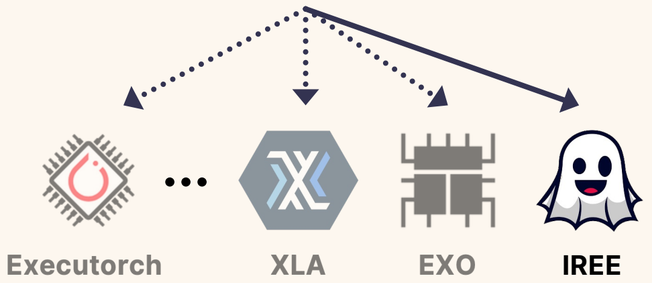
Your Favorite Architecture & ISA

E.g.: RISC-V



ML Model(s)

(Partial) Spec



Need to change sifnificant infra...

Model not supported...

Quantization not supported...

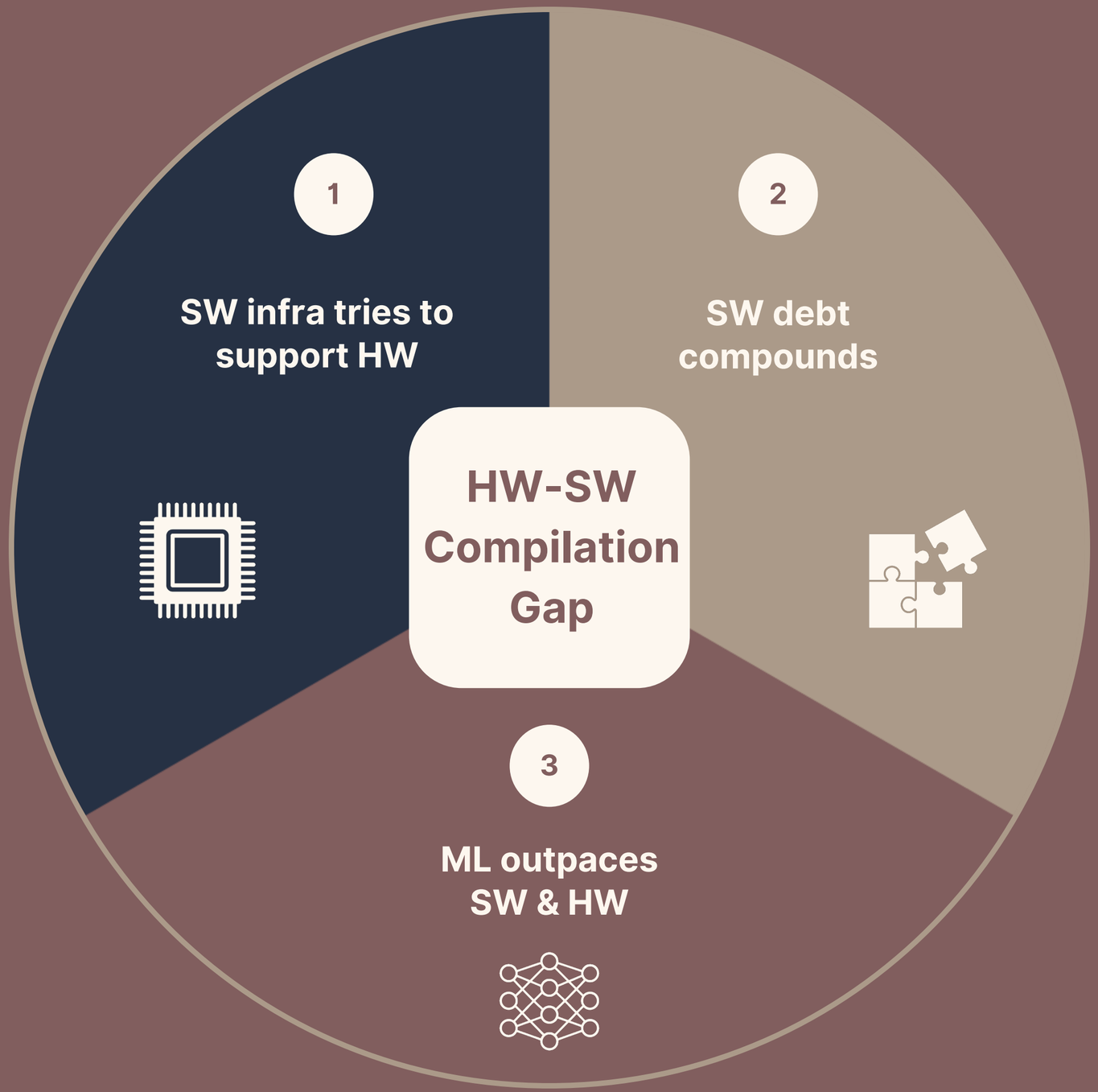
Iteration painfully slow...

Introduce... New IR

Introduce... new target

Compile... Latest VLA

Compile... Latest Quantization


1
SW infra tries to support HW
2
SW debt compounds
HW-SW Compilation Gap
3
ML outpaces SW & HW

SW / HW

Models change quickly

New ops / quantization schemes appear

Generated kernels overfit to shapes

Capture can erase structure

.
.
.

Local kernel wins do not imply E2E wins

Target bringup requires stack-wide changes

Compiler development is slow

DSE needs faithful workload facts, not guesses

.
.
.

Targets are increasingly specialized

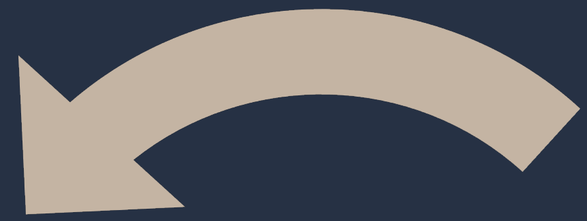
Each target exposes different SW interaction

Simulation fidelity is expensive

Developed for yesterday's algorithms

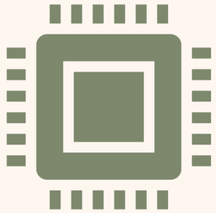
.
.
.

SW



HW

Why Merlin and for who?



Expose multiple entry points for the compiler to evolve with the HW

Native integration with Zephyr RTOS multicore

Build to interact with LLMs and agents from the ground-up

You need for your SW stack to no fall behind your HW

You need a runtime friendly for HW simulation

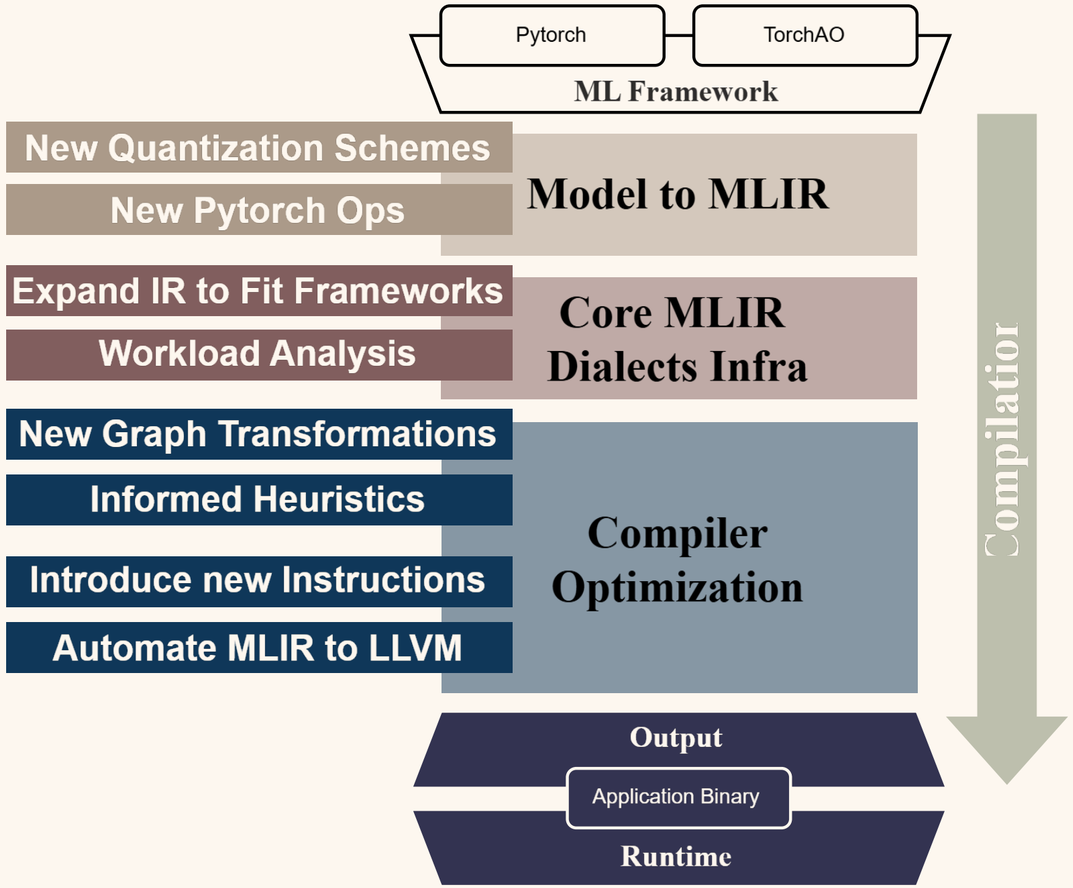
You want to iterate closer to the speed ML does

HW Co-design

Lightweight MLIR compilation

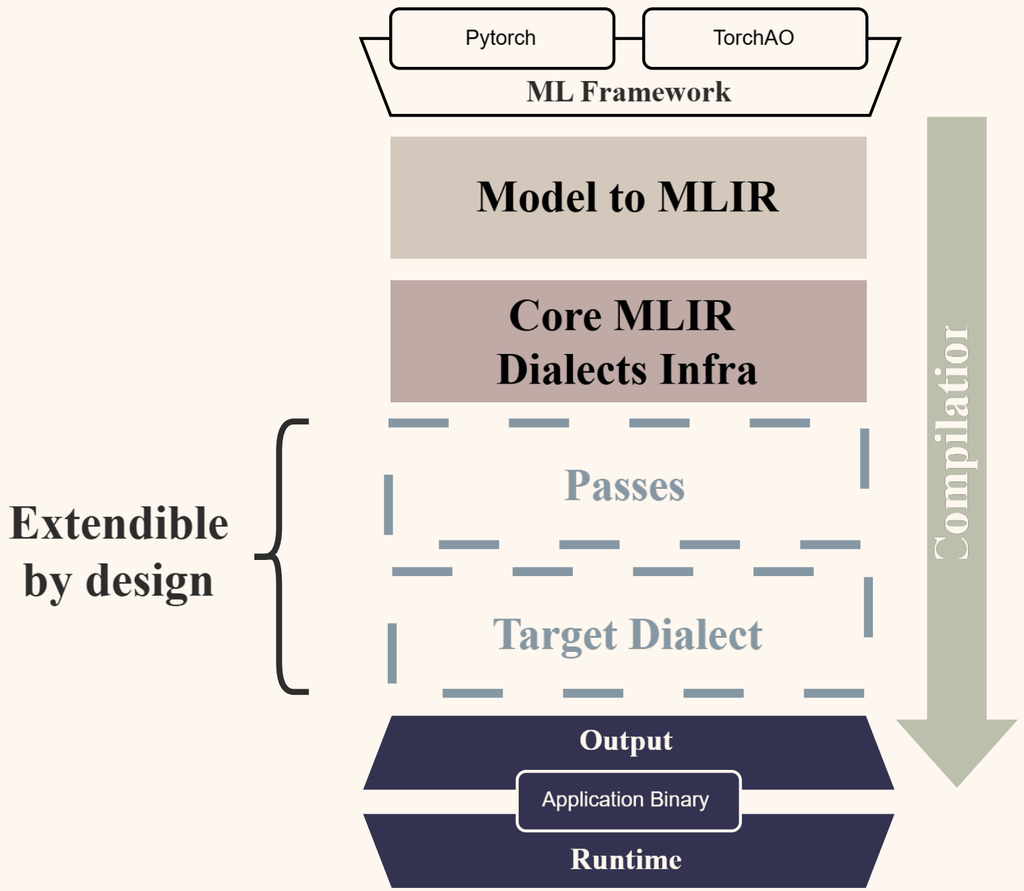
Bring up flexibility

Full-stack agile development



Improvement Targets
Capture / Normalization
Generated
Output / Runtime Interaction
Facts / Context
IR / Feedback
Infrastructure
Input / Editable Artifacts

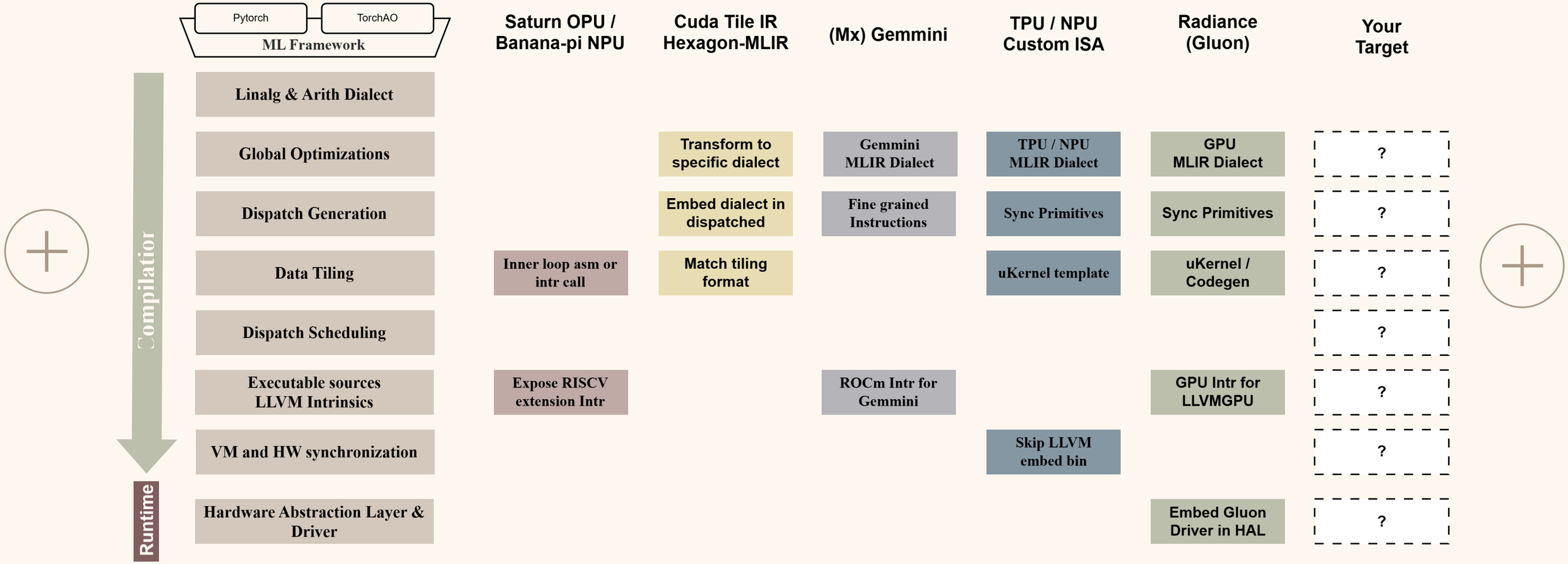
Full-stack agile development



Improvement Targets
Capture / Normalization
Generated
Output / Runtime Interaction
Facts / Context
IR / Feedback
Infrastructure
Input / Editable Artifacts

Bring up of your target in MLIR

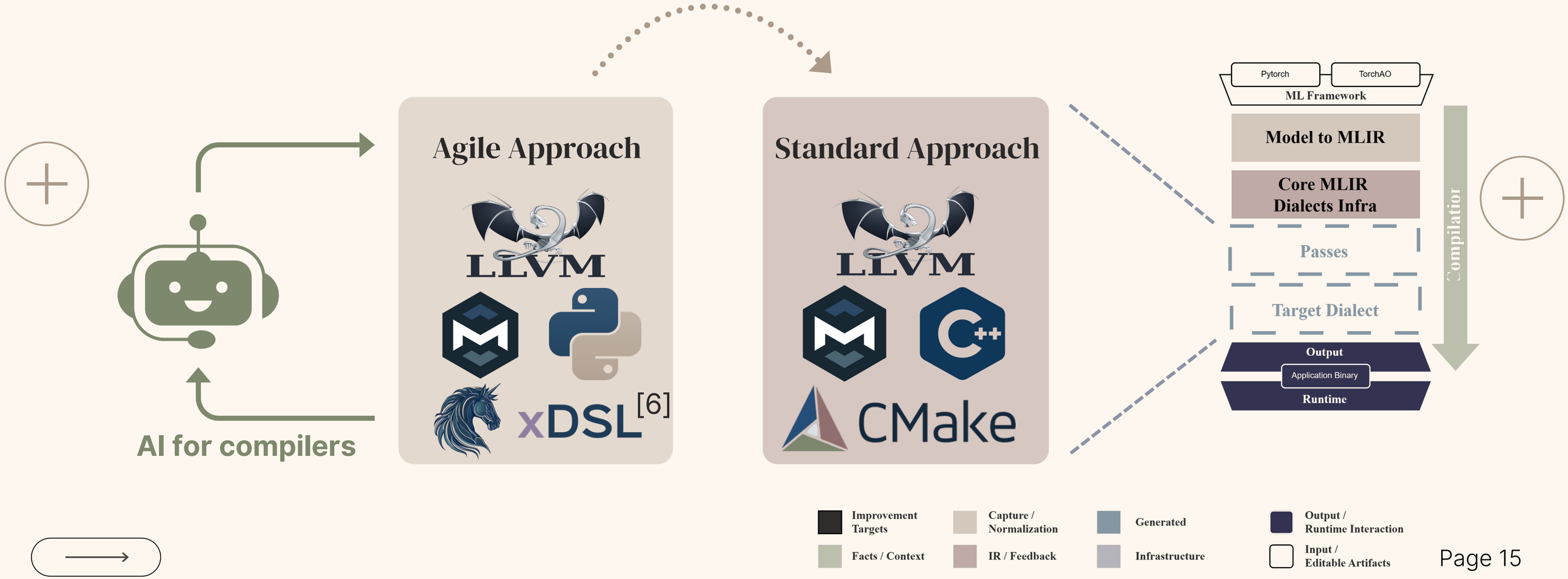
Writing a compiler for your HW target



MLIR dialect target bring up

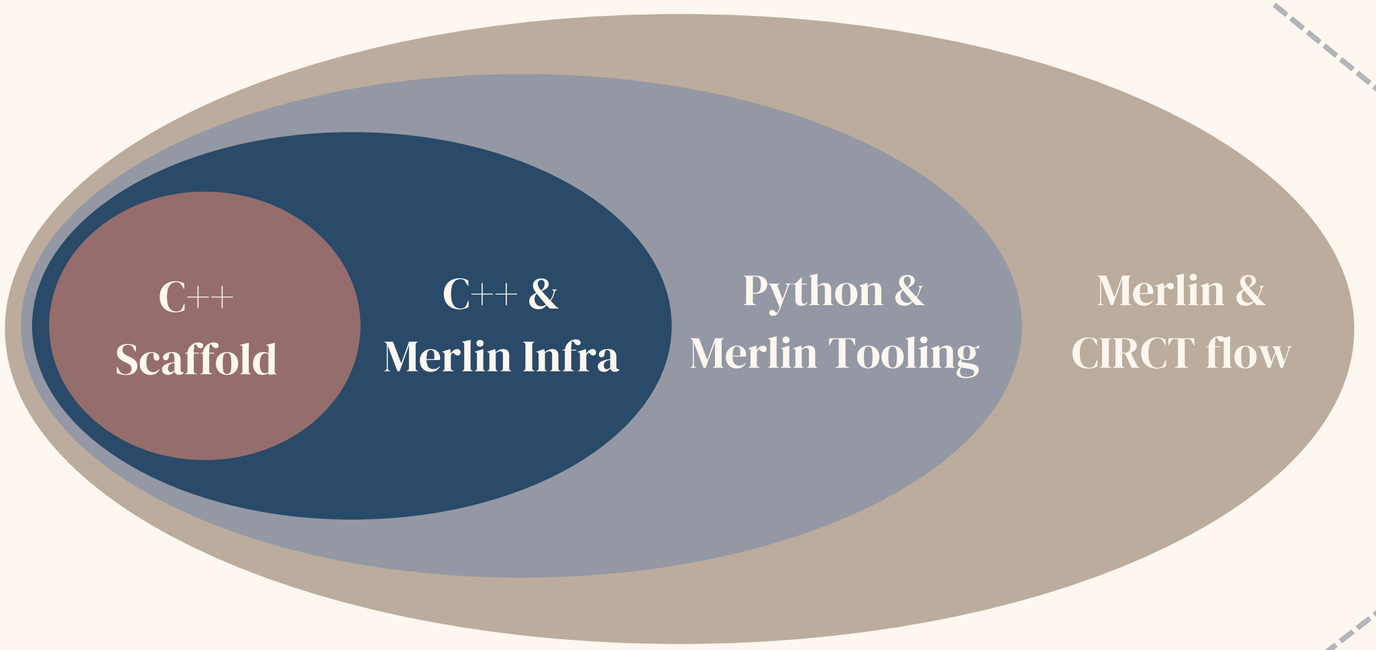
We support multiple ways to integrate your dialect & passes

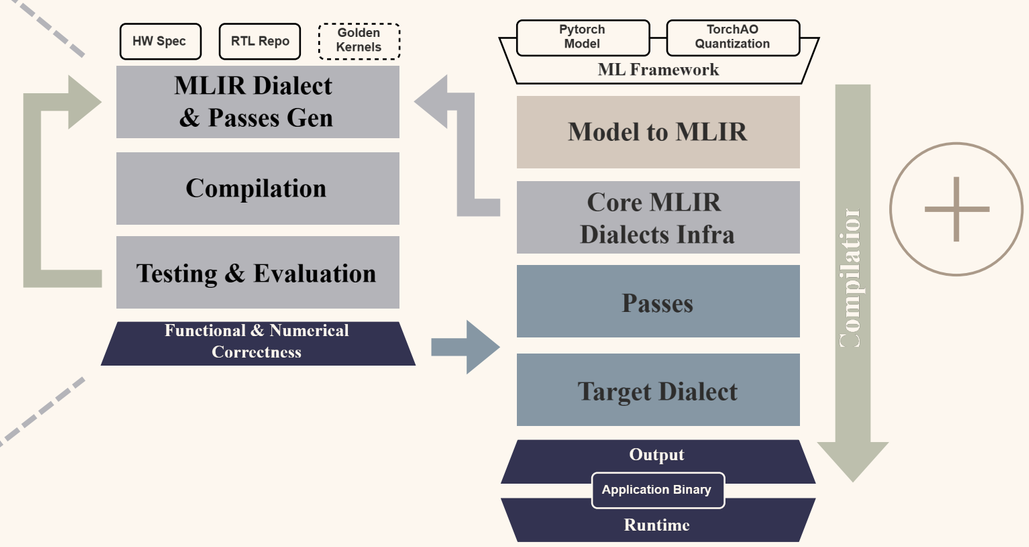
But which approach should you use?



Agentic experiments:

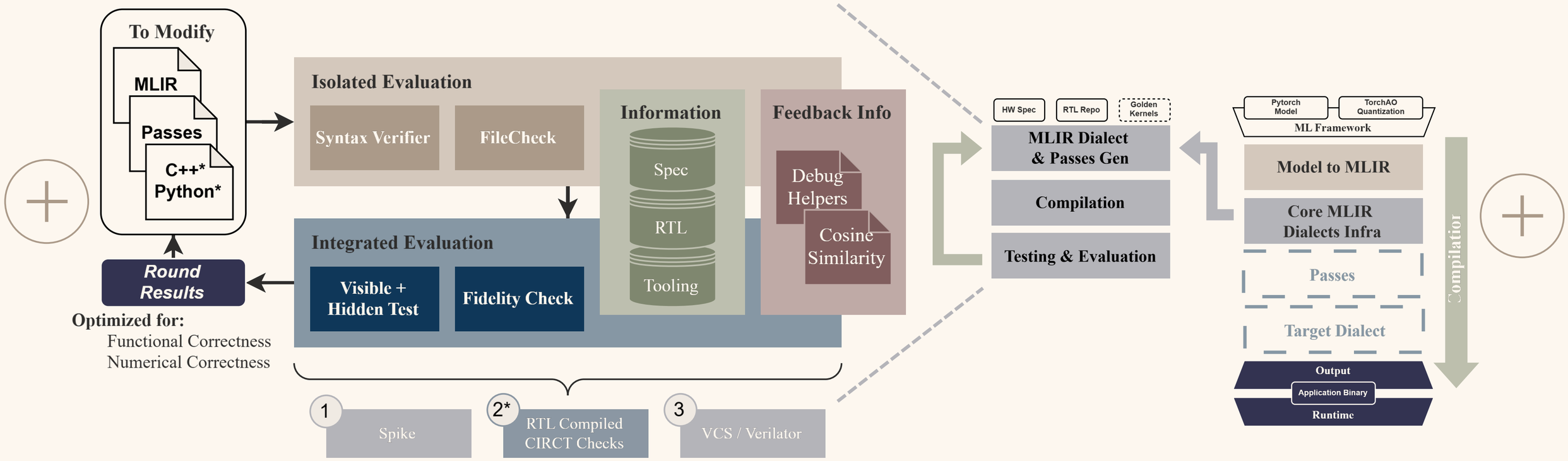
[3] Gemmini case study



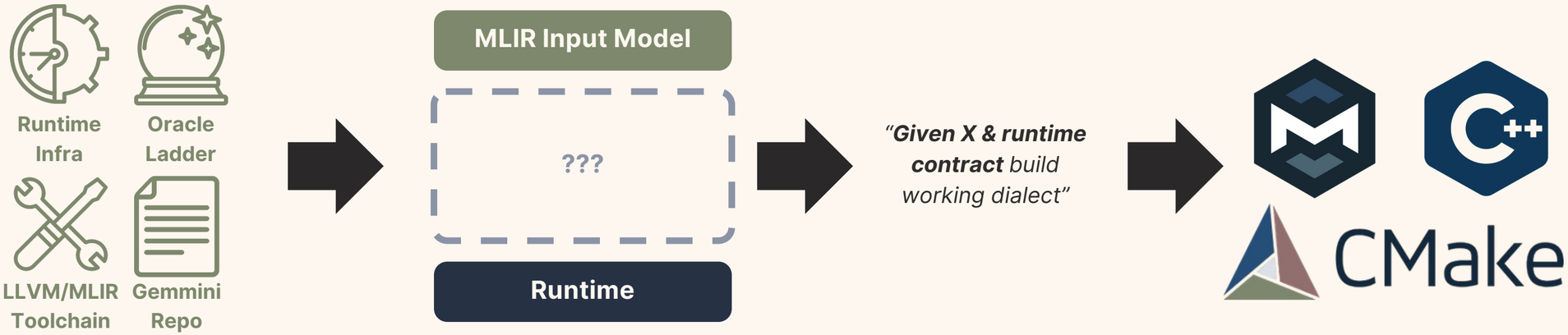


Task → Same
Model → Same
Grading → Same
Sandbox → Same

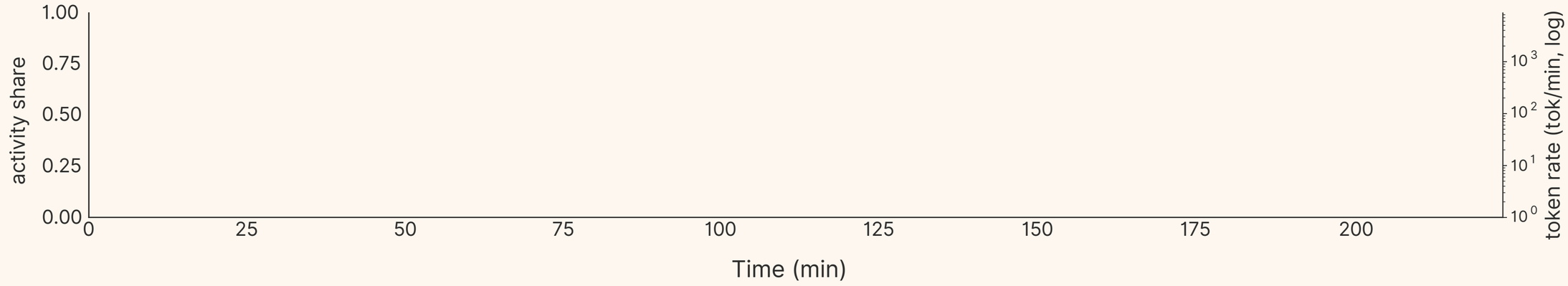
Agentic MLIR generation loop



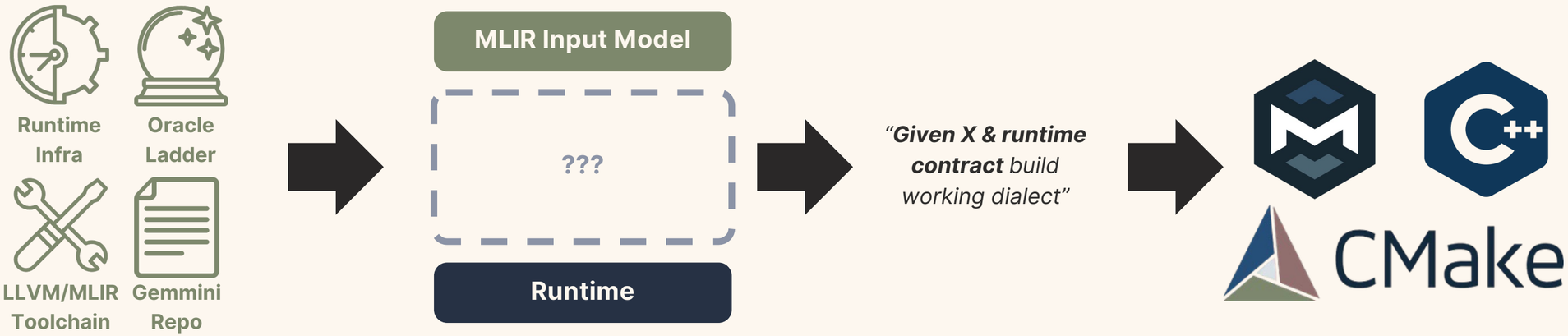
Experiment results:
C++ Scaffold



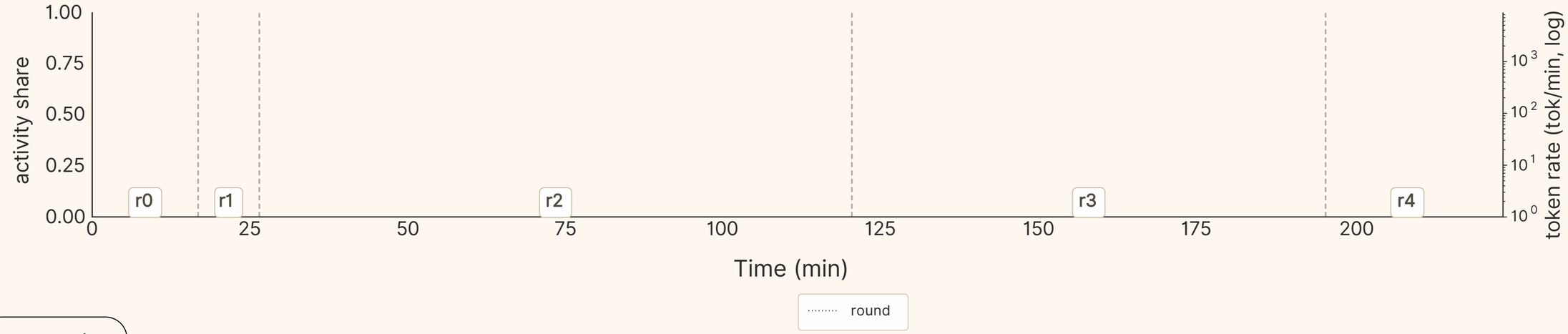
raw C++ (from scratch)



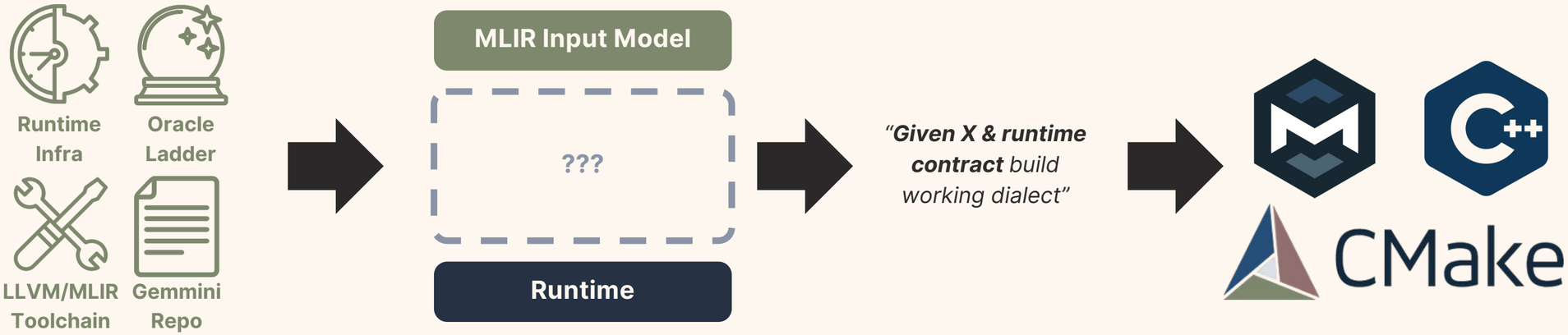
Experiment results:
C++ Scaffold

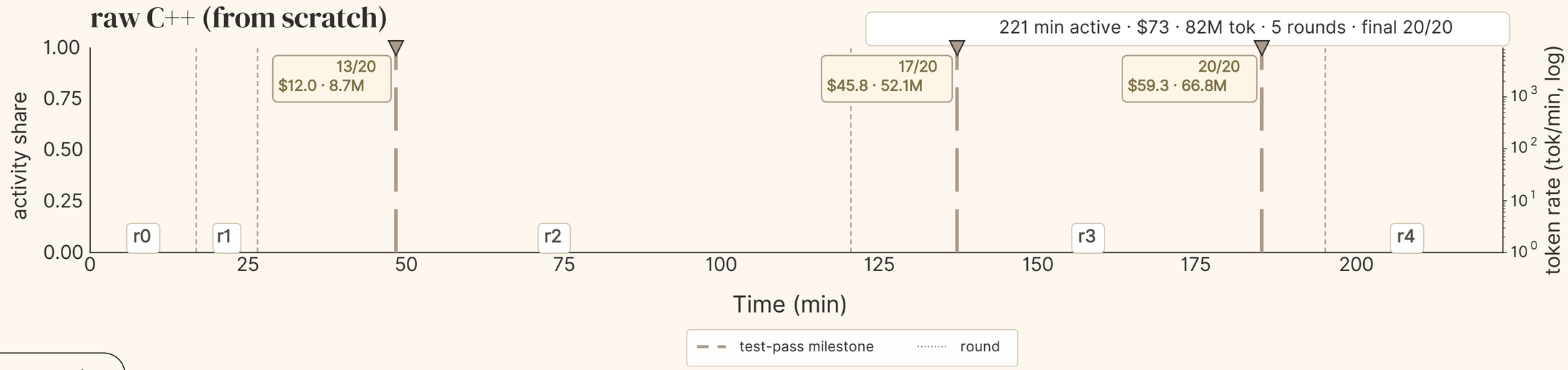


raw C++ (from scratch)

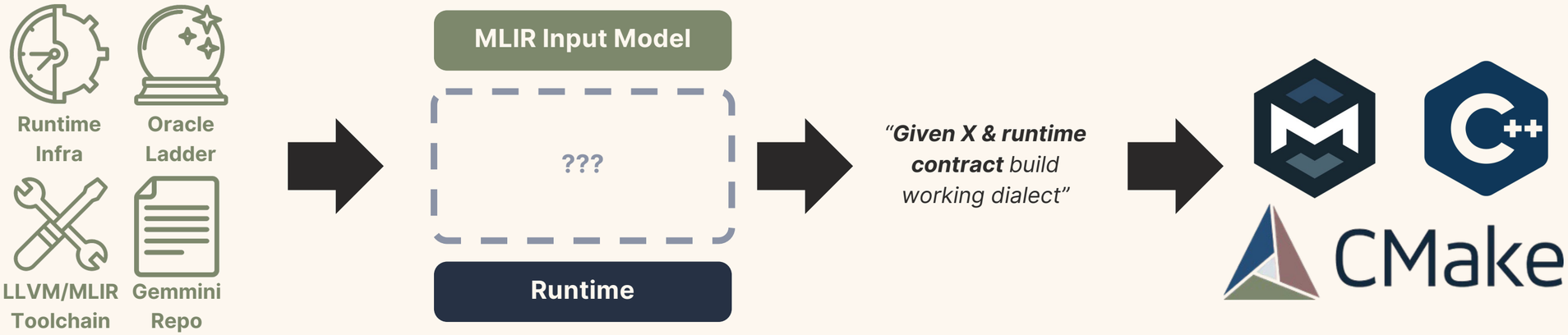


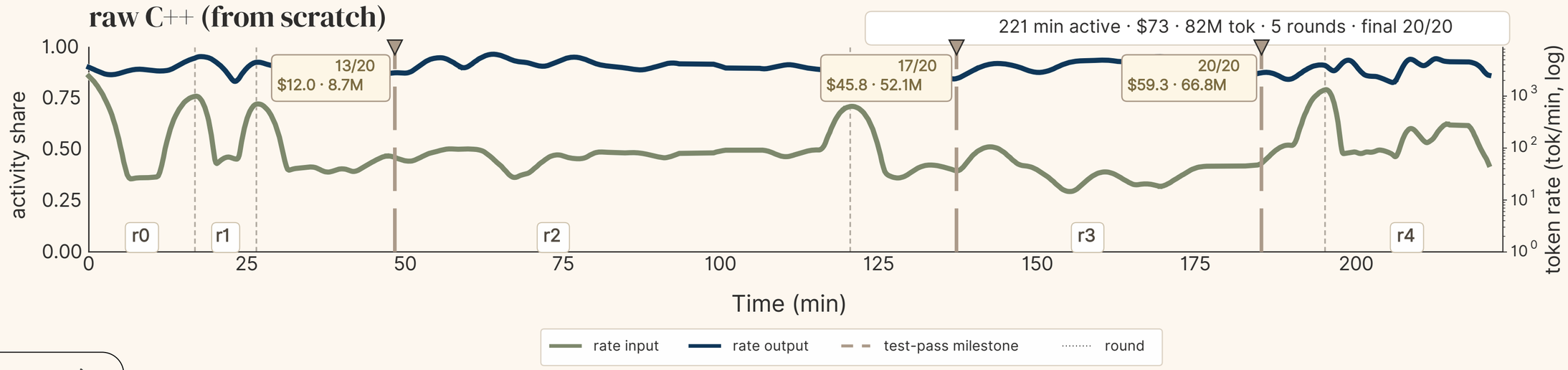
Experiment results:
C++ Scaffold



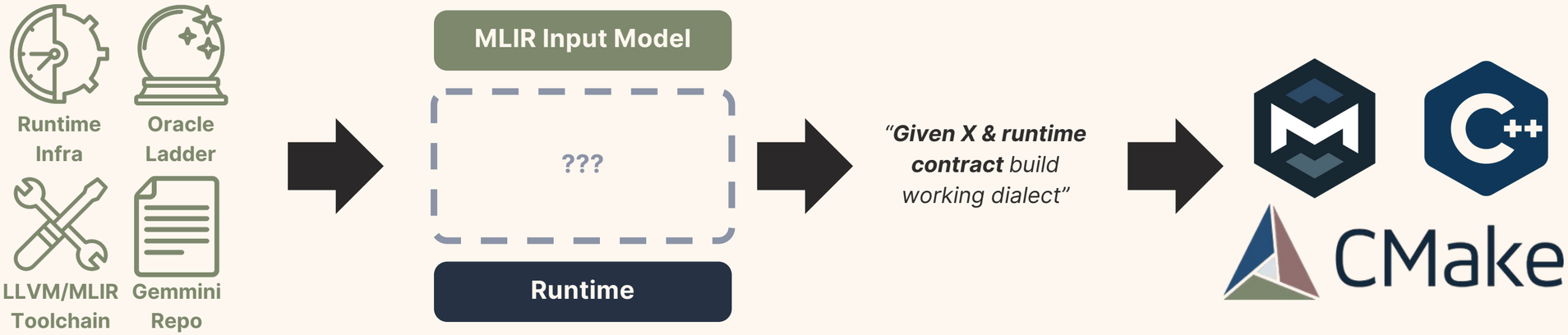


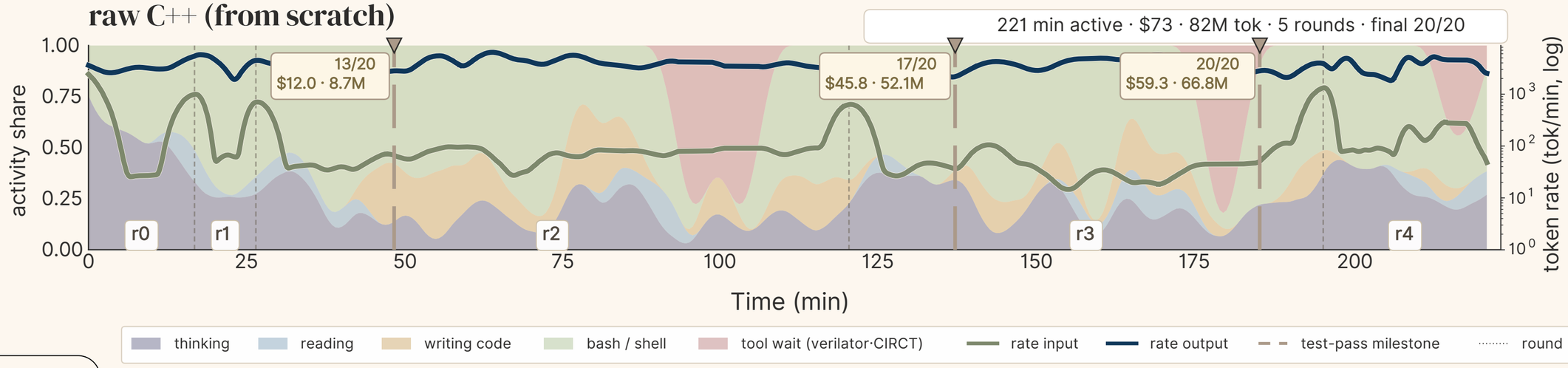
Experiment results:
C++ Scaffold



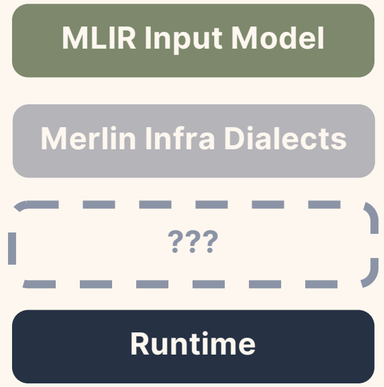


Experiment results:
C++ Scaffold

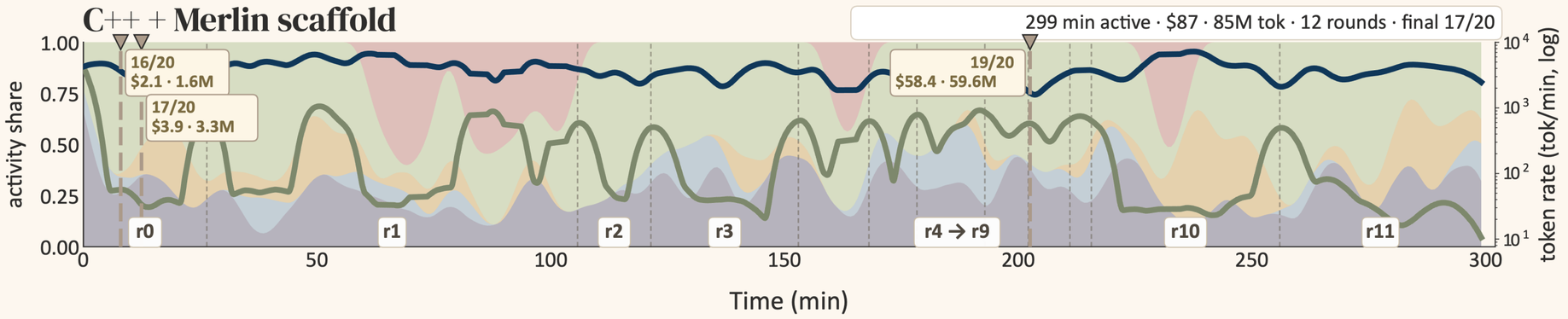




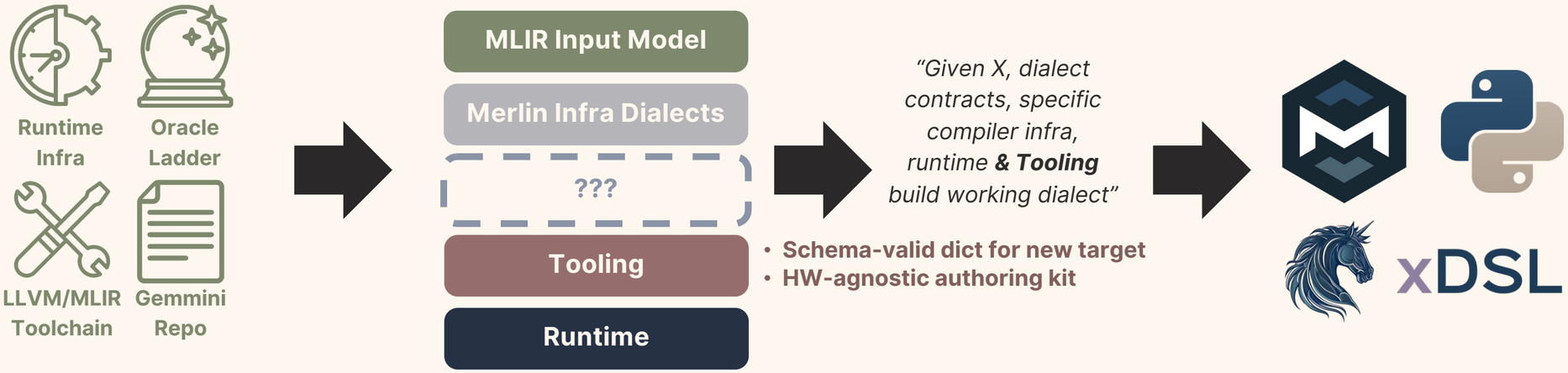
Experiment results:
C++ & Merlin infra

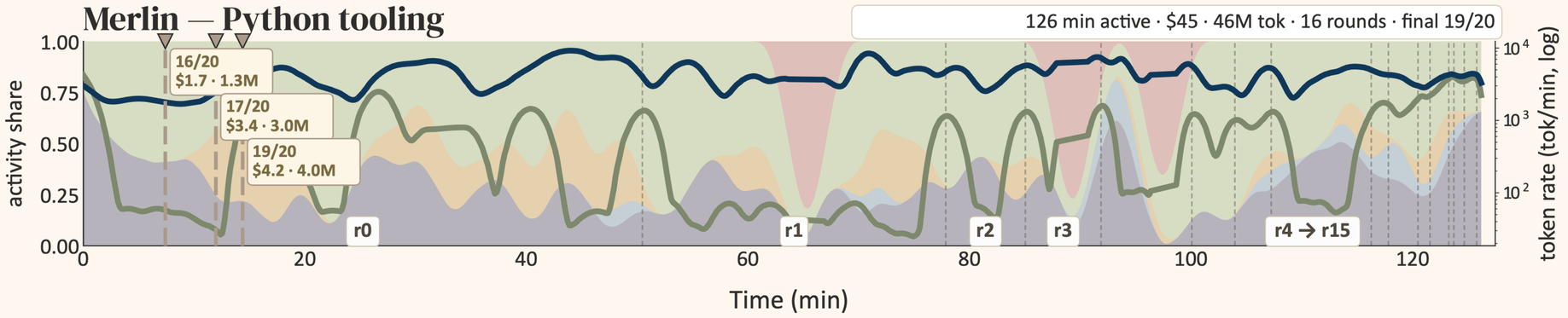


"Given X, ***dialect contracts, specific compiler infra*** *& runtime build working dialect"*



Experiment results:
Python & Merlin Tooling





Experiment results: Previous + CIRCT flow

MLIR Input Model

Merlin Infra Dialects

CIRCT Helper

???

Tooling

Runtime

*"Given X, dialect contracts, specific compiler infra, runtime, tooling and **compiled HW hints and checks** build working dialect"*

RTL → CIRCT Compilation to hw dialect → DAG

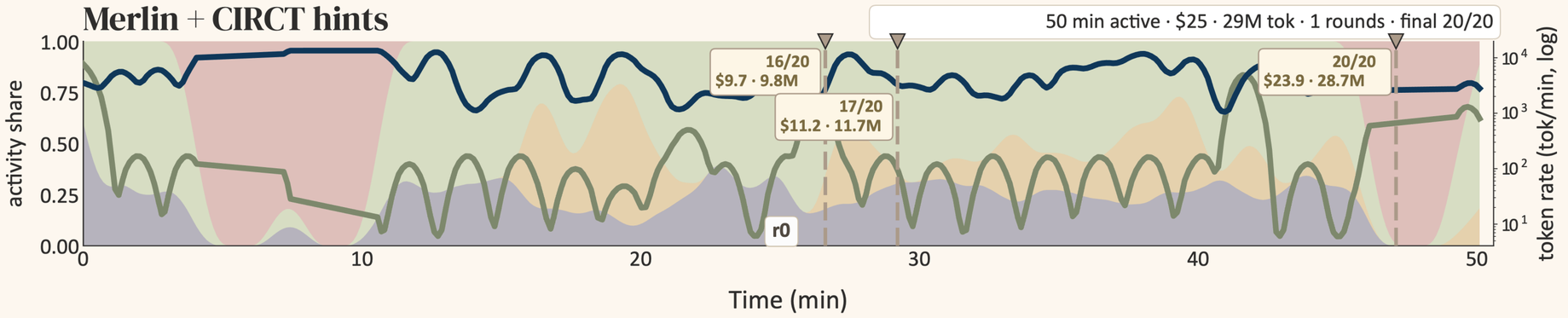
Using Passes

Extract

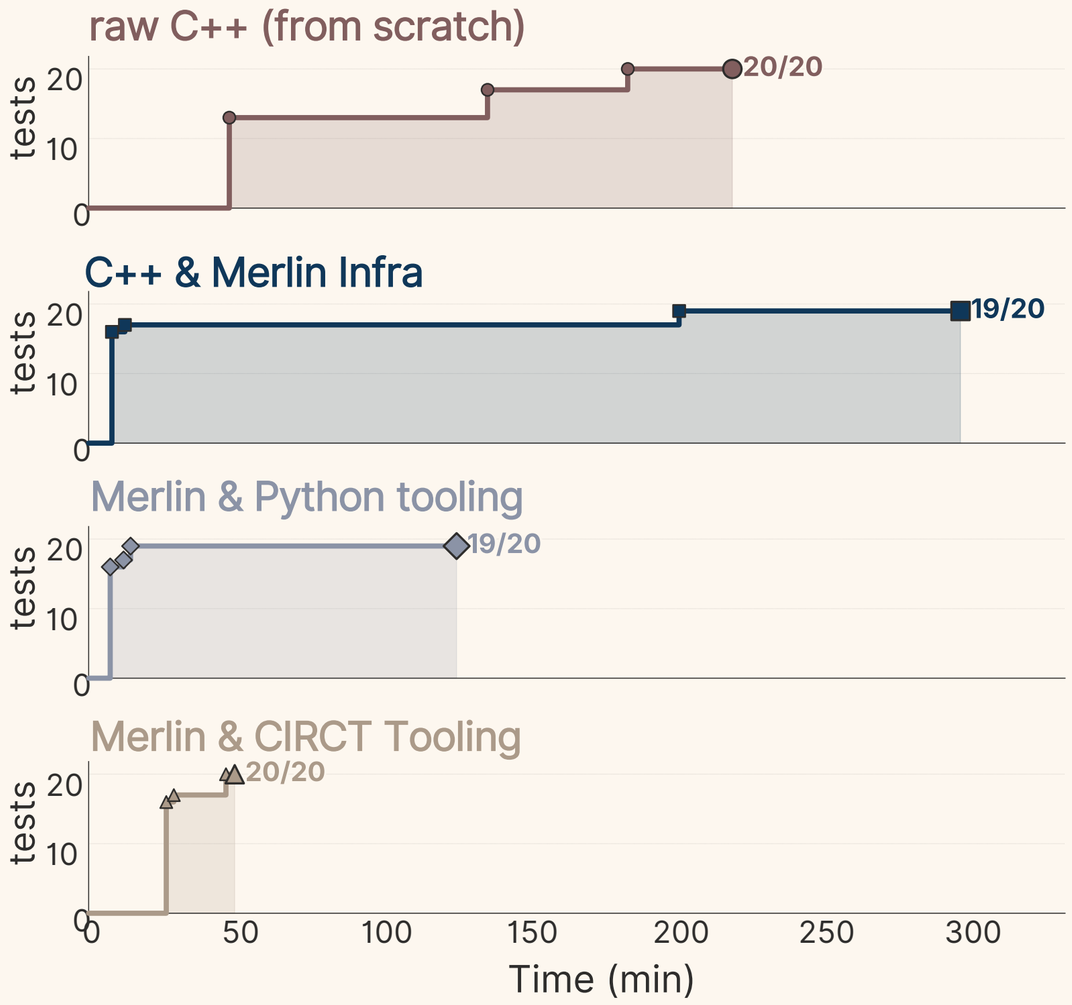
```
array: 16 x 16
scratchpad_bytes: 262144
legal_funct: 3, 4, 5, 6, 7, 8
custom_opcode: 0x7b
spad_addr: < 0x40000
```

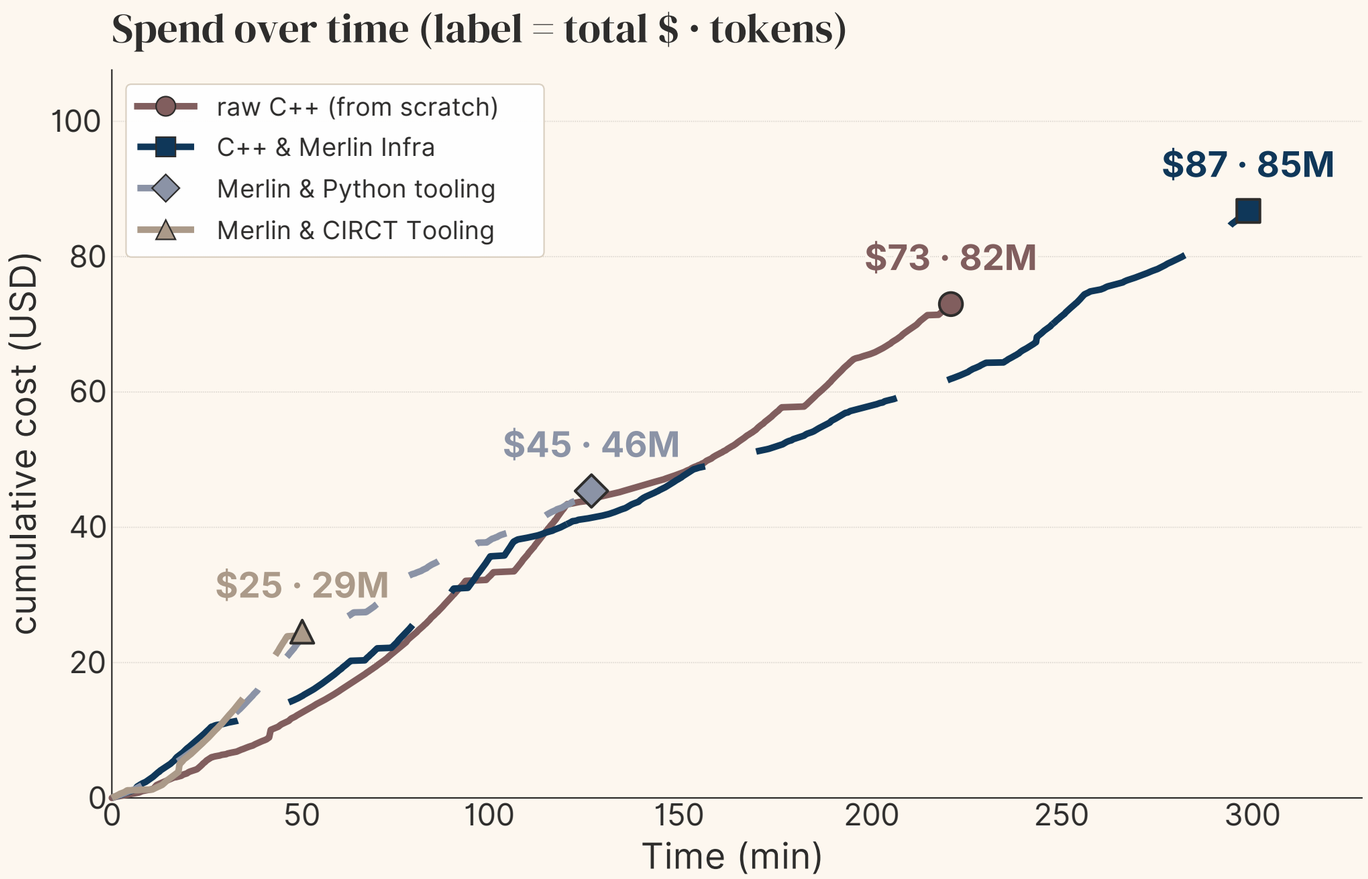
Compiled FileCheck tests

```
// CHECK:           funct in legal_funct
// CHECK:           spad_addr < 0x40000
// CHECK-COUNT-16:  mvout // 16x16 → tiles
// CHECK-NOT:       UNKNOWN opcode
```

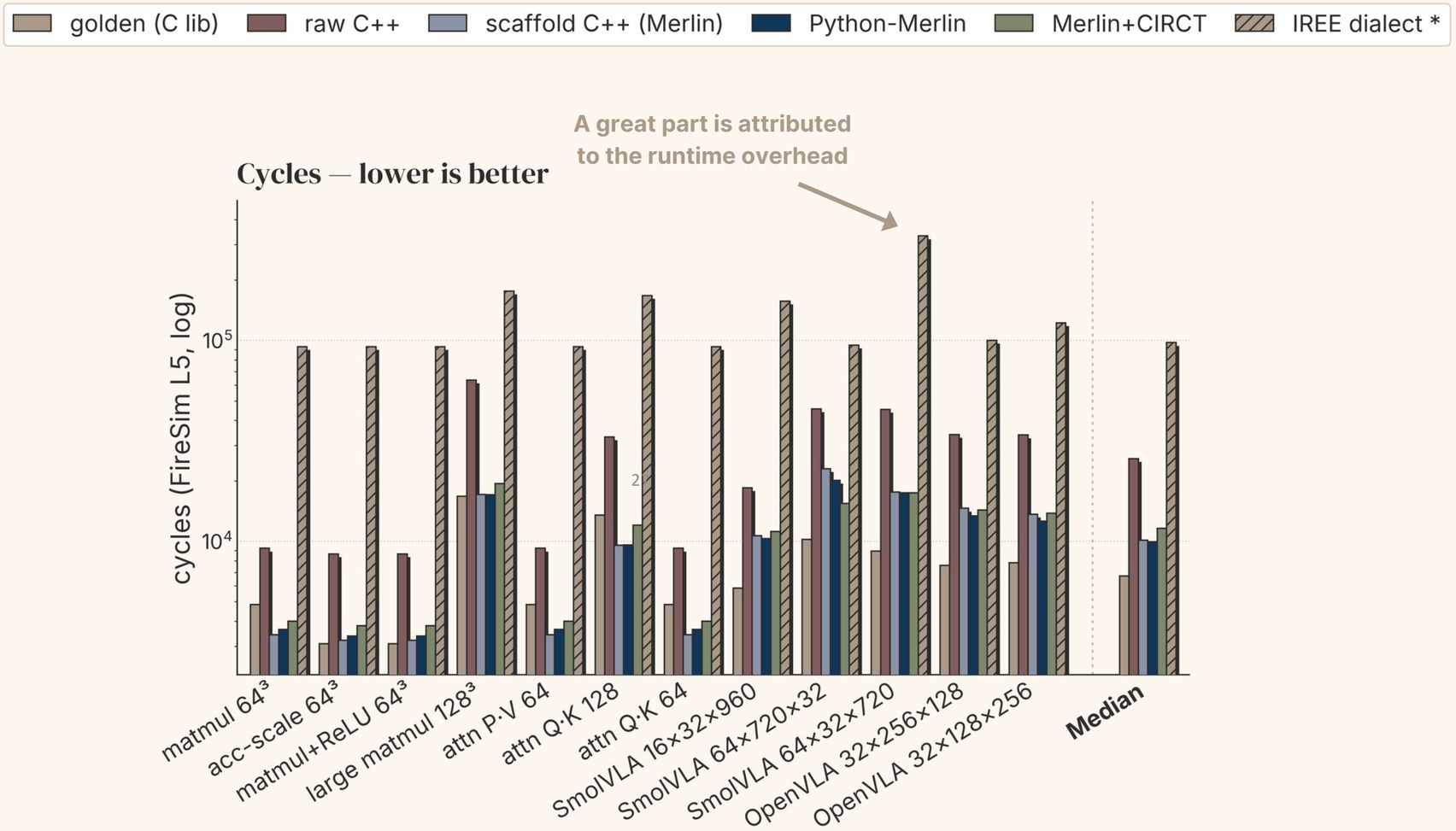


Experiment results overview



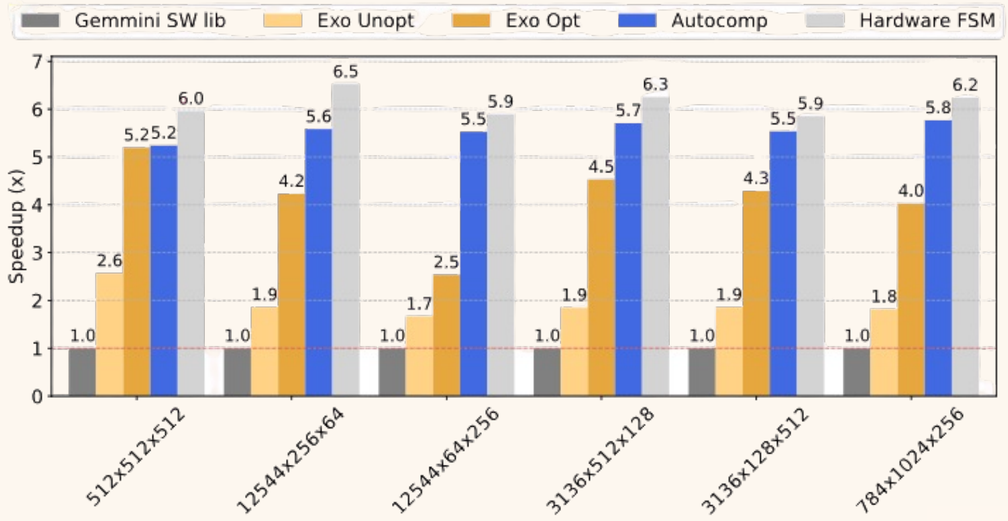


Experiment results: Gen vs. IREE & Golden Lib



Optimizing your Compiler with Kernels

Generated kernels advance fast but overfit

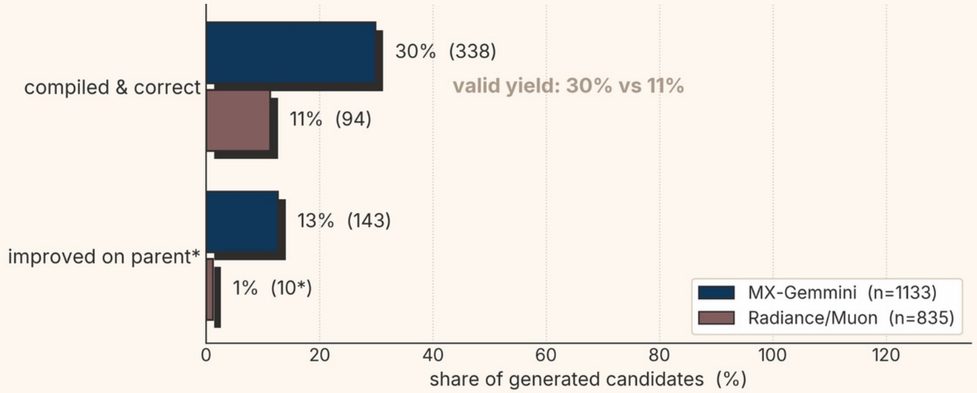


Great results in a short period of time!

Good iteration speed!

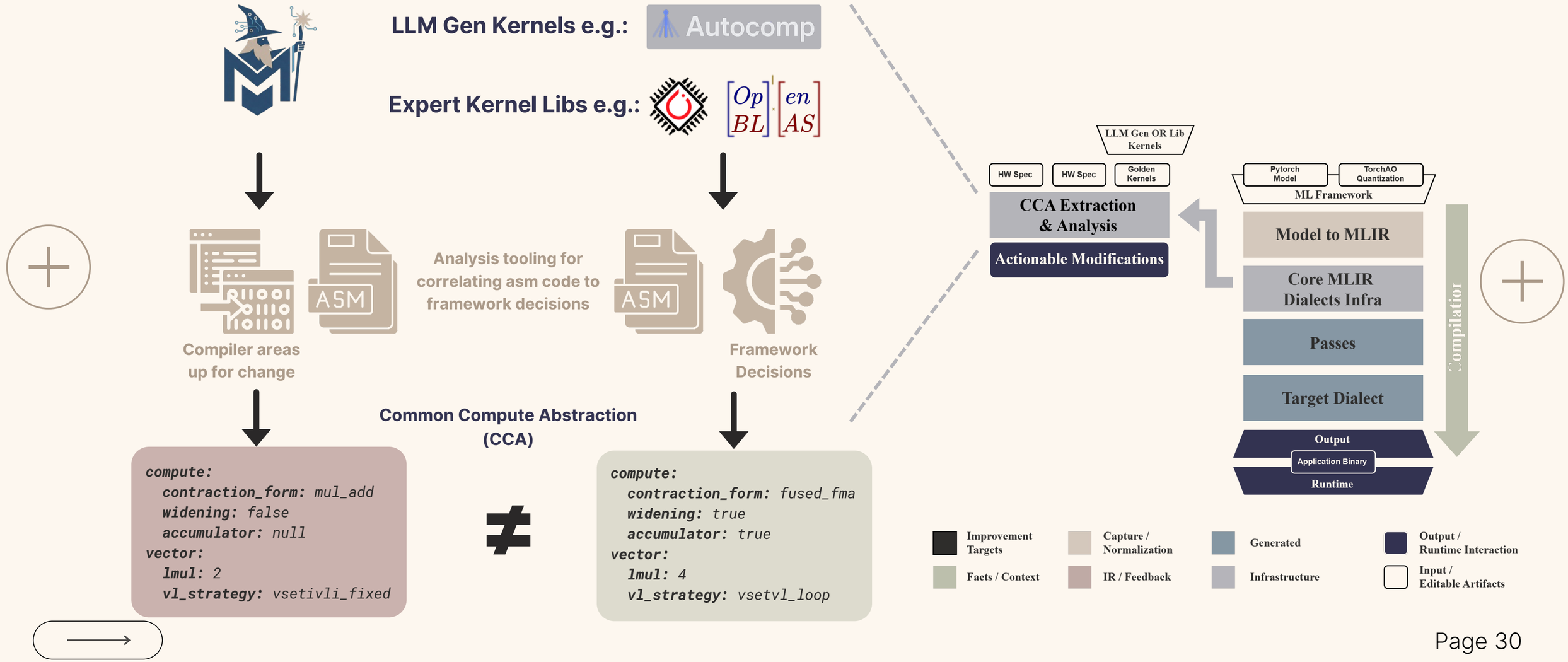
Case-by-case optimizations

Iteration in simulators or firesim costly



Not Kernels OR Compiler → BUT BEST OF BOTH

Extracting information from kernels



Integrating kernel learnings

Our CCA ≠ Expert CCA

What to improve?

Tiling / Dataflow | Fusion / Layout | Register Residency | Instruction Selection | Runtime / Sync

Actions to take:

Pass | Heuristic | Flag | Knob

Beam Search

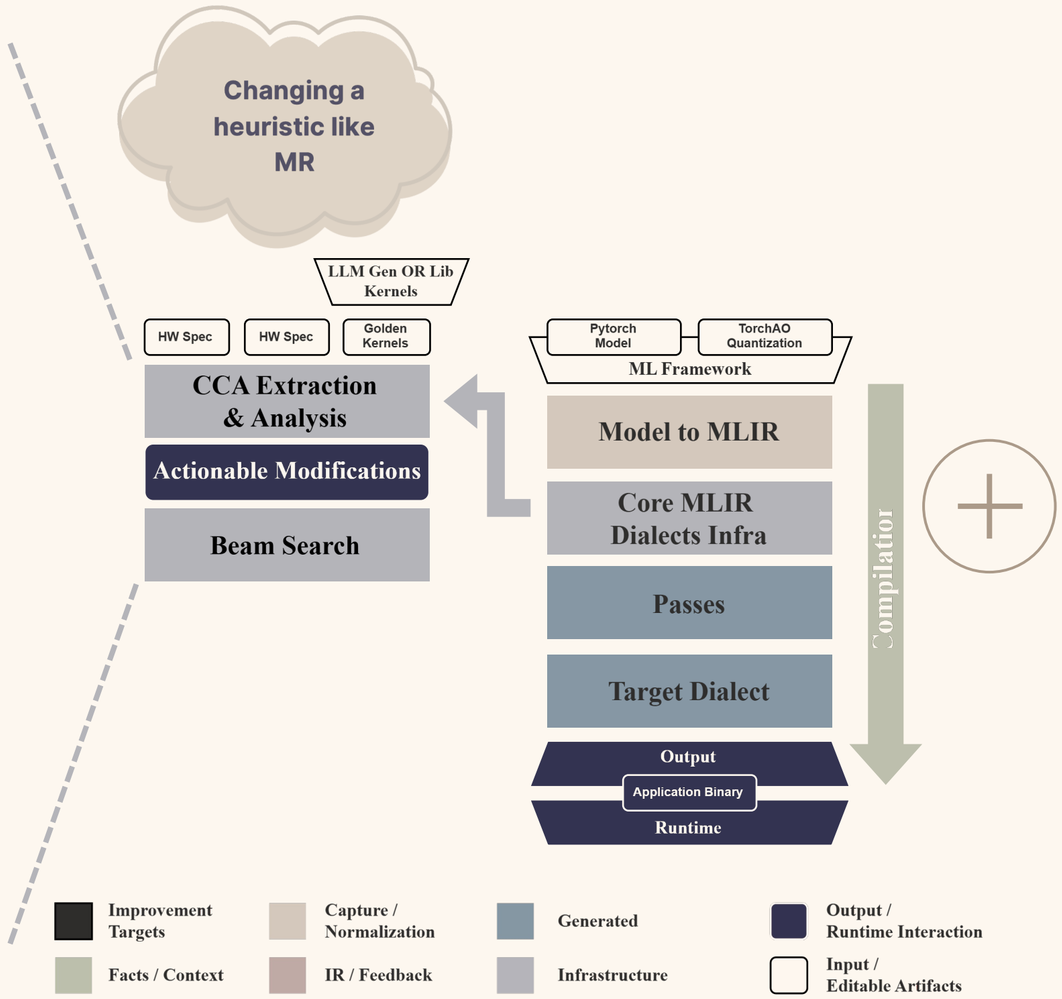
Fork Validity & Build — Did the CCA divergance close?

Functional Correctness — Does the IR behave as intended?

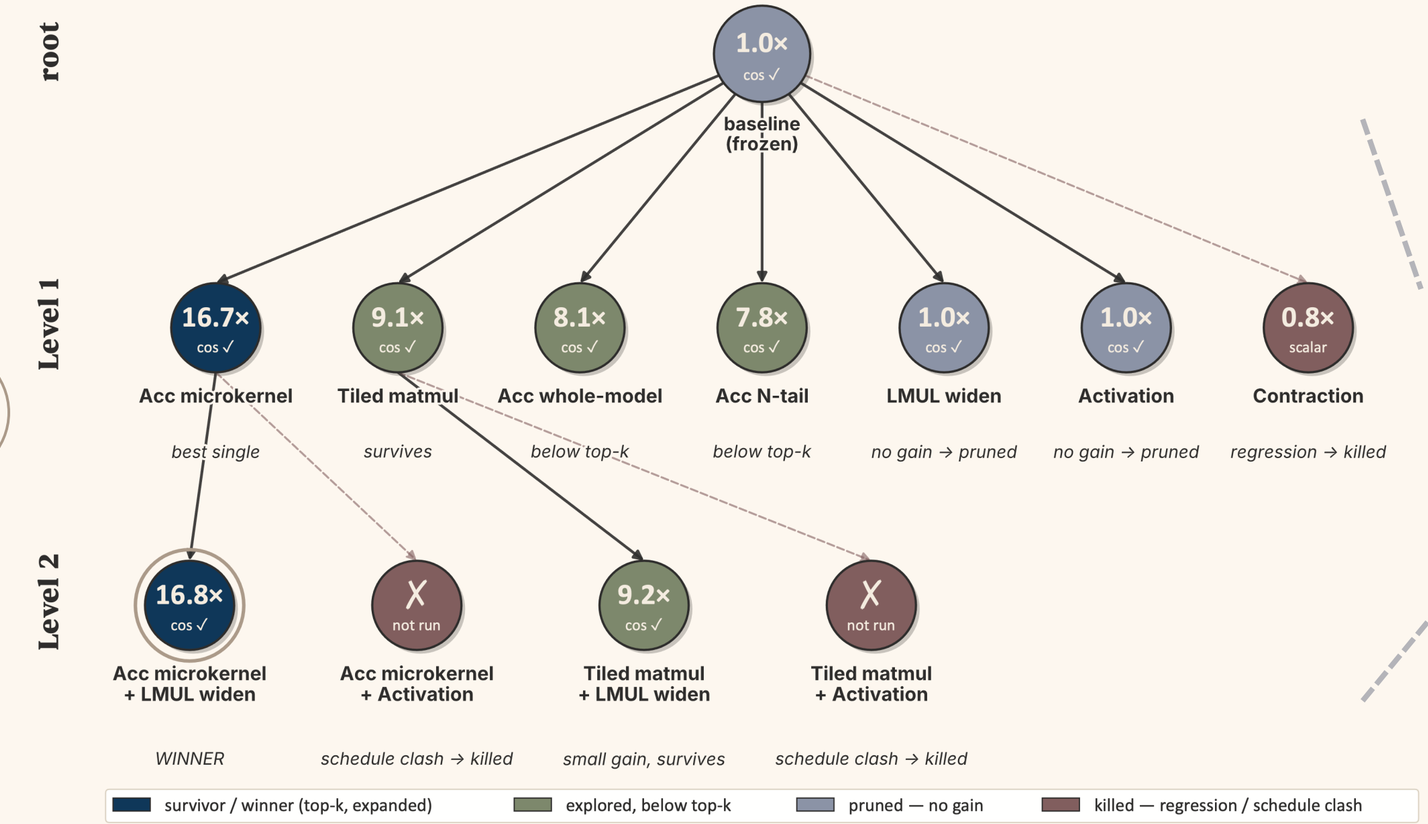
Utilized HW Capabilities — Real or fake speedup?

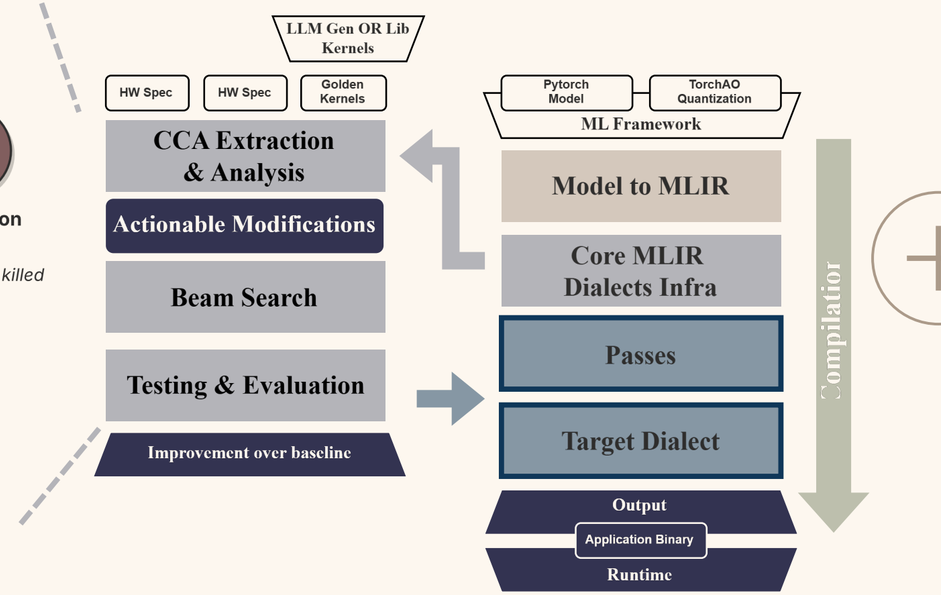
HW-in-the-loop True Cost — What is the performance?

If fail at gate:
Candidate pruned

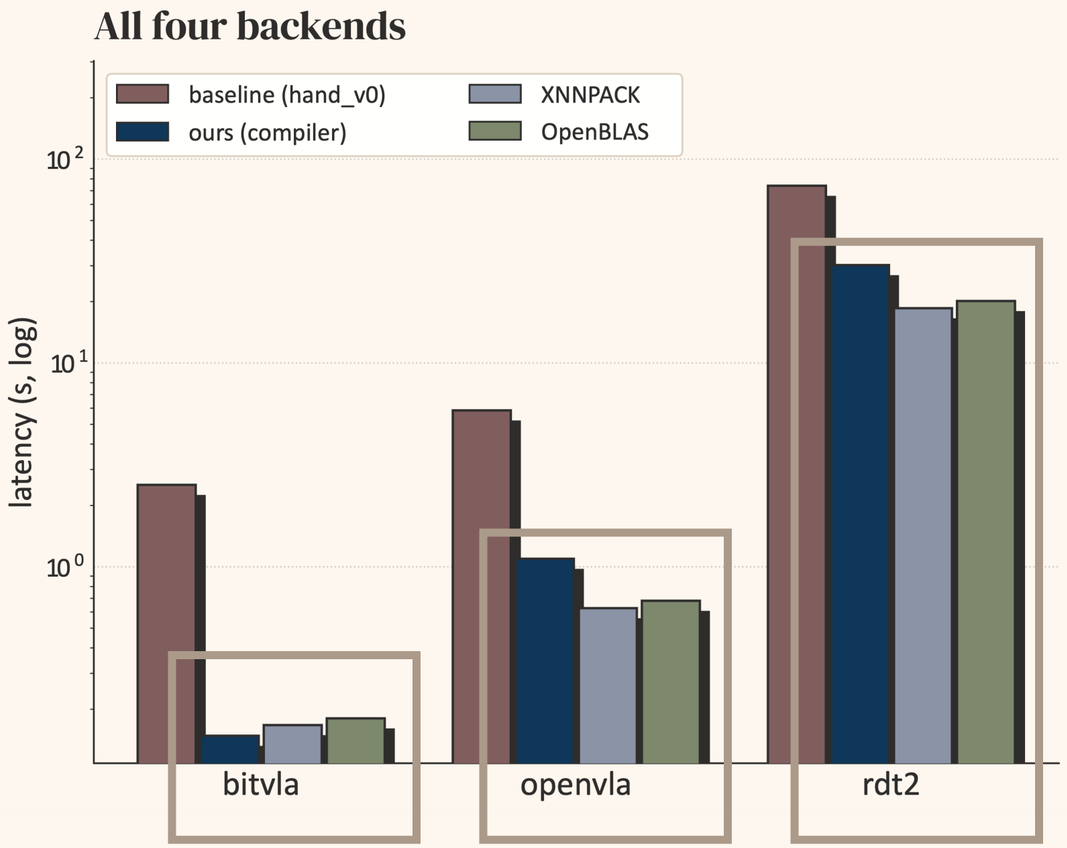


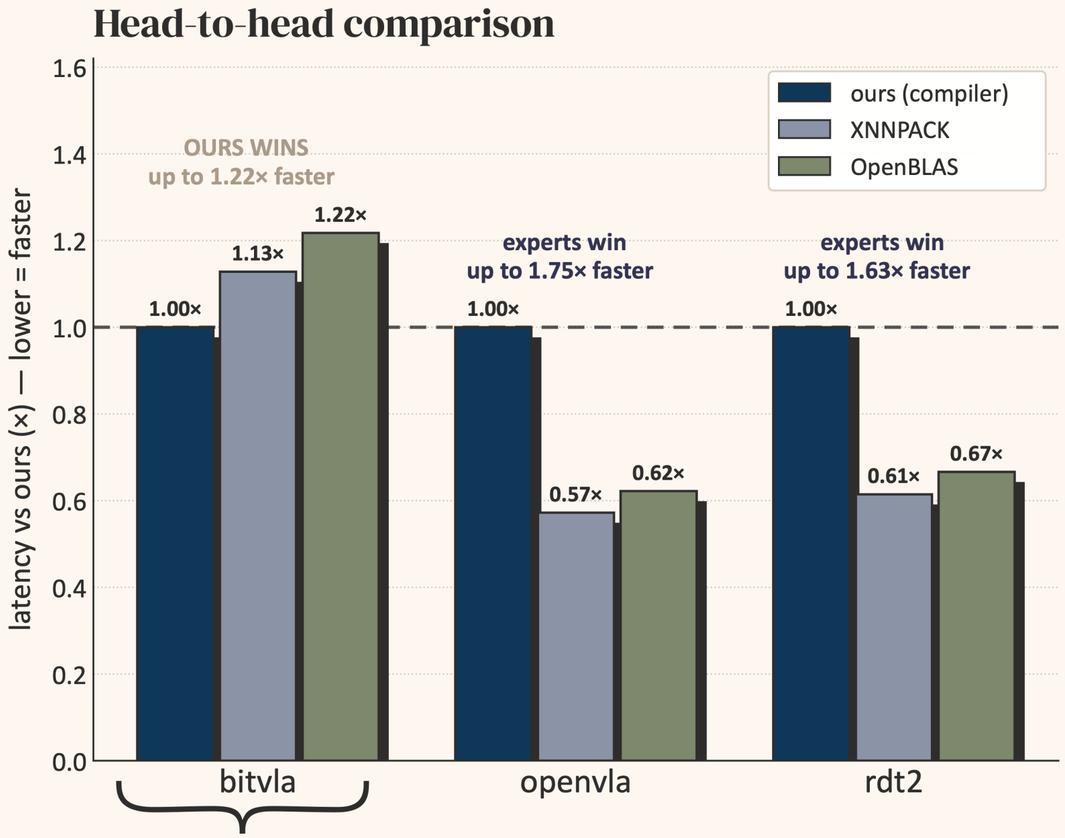
Beam search through each modification





Experiment results on Saturn RVV [4]





Partitioning of model beyond XNNPACK lib boundaries

We find ways to optimize compute & requant for 1.58-bit weights

Optimized packing & removal of dispatch overhead

Extending our study cases

1st BASELINE

IREE

Currently all targets mentioned have working examples in our OOT IREE extension

All targets will be migrated through target gen experiments!

Spatial Array	NPU	Mix precision Spatial Array	SIMT Core
e.g.: Gemmini	*e.g.: Atlas NPU*	*e.g.: Gemmini-MX*	*e.g.: Radiance*
Working!	TODO	TODO	In progress

[5] RADIANCE

Muon Talk @OSCAR: **1:30 p.m. Today!**

Points left to address

Agentic MLIR Gen

- + data points
- + HW targets

- **Compare to TVM & EXO**

Improvements from CCA

Workload Analysis Tool

- **Test with DSE tool**

SLICE

THANK YOU FOR YOUR TIME!

References

[1] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar,R. Riddle, T. Shpeisman, N. Vasilache, and O. Zinenko, “MLIR: A compiler infrastructure for the end of Moore’s law,” arXiv preprintarXiv:2002.11054, 2020.

[2] The IREE Authors, “IREE,” 2019. [Online]. Available: https://iree.dev/.Repository: https://github.com/iree-org/iree

[3] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao,D. Grubb, H. Liew, H. Mao, A. Ou, C. Schmidt, S. Steffl, J. Wright,I. Stoica, J. Ragan-Kelley, K. Asanovic, B. Nikolic, and Y. S. Shao,“Gemmini: Enabling systematic deep-learning architecture evaluationvia full-stack integration,” in Proceedings of the 58th Annual DesignAutomation Conference (DAC), 2021

[4] J. Zhao, D. Grubb, M. Rusch, T. Wei, K. Anderson, B. Nikolic, andK. Asanovi´c, “The Saturn Microarchitecture Manual,” EECS Depart-ment, University of California, Berkeley, Tech. Rep. UCB/EECS-2024-215, Dec. 2024. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2024/EECS-2024-215.html

[5] H. Kim, R. R. Yan, S. Anand, J. You, M. Shi, C. W. Fletcher, and Y. S.Shao, “Radiance,” GitHub repository, 2026. [Online]. Available: https://github.com/ucb-bar/radiance

[6] Fehr, M., Weber, M., Ulmann, C., Lopoukhine, A., Lücke, M.P., Degioanni, T., Vasiladiotis, C., Steuwer, M. and Grosser, T., 2025, March. xdsl: Sidekick compilation for ssa-based compilers. In *Proceedings of the 23rd ACM/IEEE International Symposium on Code Generation and Optimization* (pp. 179-192).