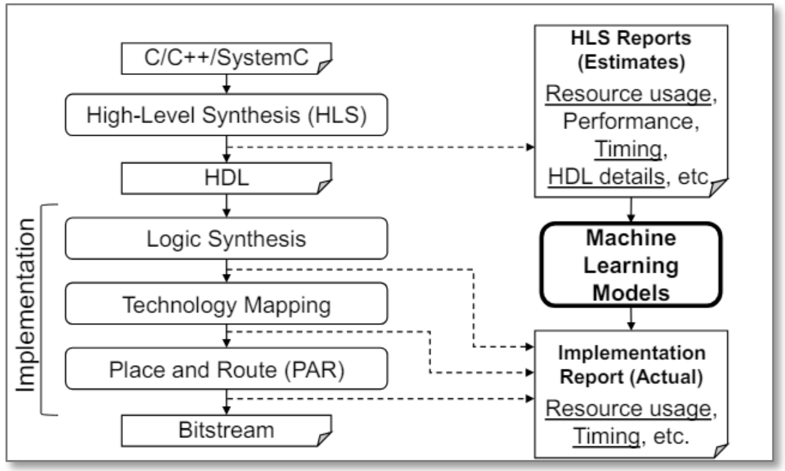
HLSFactory
A Framework Empowering High-level Synthesis Datasets For Machine Learning And Beyond

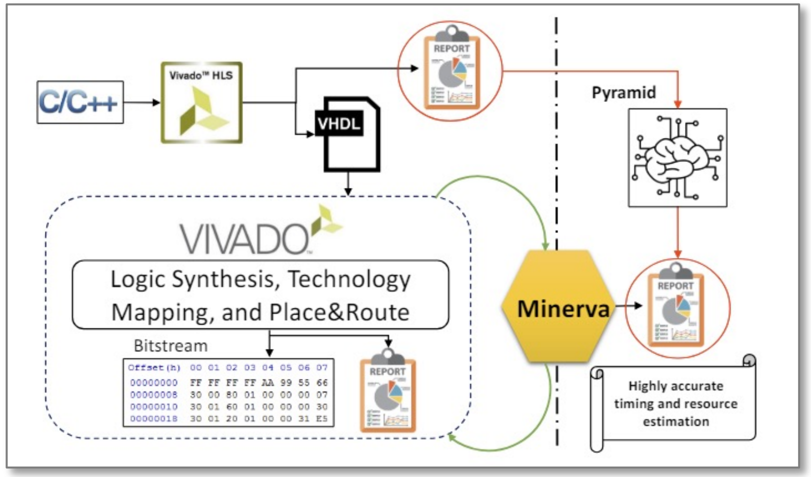
Stefan Abi-Karam[1,2], Rishov Sarkar[1], Allison Seigler[3], Sean Lowe[4], Zhigang Wei[3], Hanqiu Chen[1], Nanditha Rao[5], Lizy John[3], Aman Arora[4], Cong Hao[1]

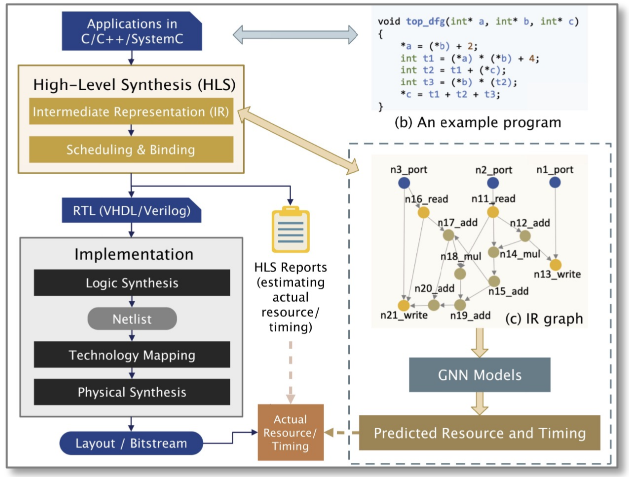
[1]Georgia Institute of Technology, [2]Georgia Tech Research Institute, [3]The University of Texas at Austin
[4]Arizona State University, [5]International Institute of Information Technology Bangalore

Background

ML has been widely used in HLS domain, BUT every study has its own dataset

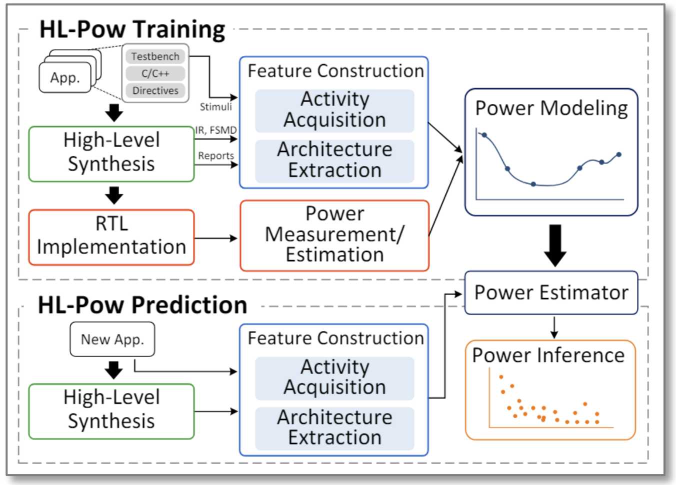


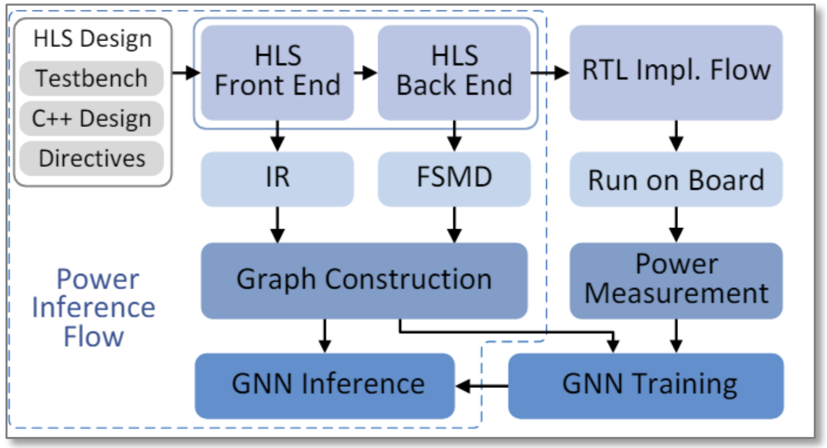

C/C++
Vivado™ HLS
VHDL
REPORT
VIVADO
Logic Synthesis, Technology Mapping, and Place&Route
Bitstream
Pyramid
Minerva
Highly accurate timing and resource estimation



```
void top_dfg(int* a, int* b, int* c)
{
    *a = (*b) + 2;
    int t1 = (*a) * (*b) + 4;
    int t2 = t1 + (*c);
    int t3 = (*b) * (t2);
    *c = t1 + t2 + t3;
}
```

Accurate Timing and Resource Estimation [FCCM’18, FPL’19, DAC’22]





Power Estimation [ASP-DAC’20, DATE’22]

Background

ML has been widely used in HLS domain:

1. XGB, ANN are used to predict post-implementation resource utilization [FCCM’19]
2. Pyramid used ANN, SVM to help find design with optimal timing and resource usage [FPL’19]
3. GNN is used to predict actual resource and timing [DAC’22]
4. HL-POW used CNN to predict on-board measured average power for each FPGA [ASP-DAC’20]
5. PowerGear used GNN further increase the accuracy of average power prediction [DATE’22]

However, every study has its own dataset

Background

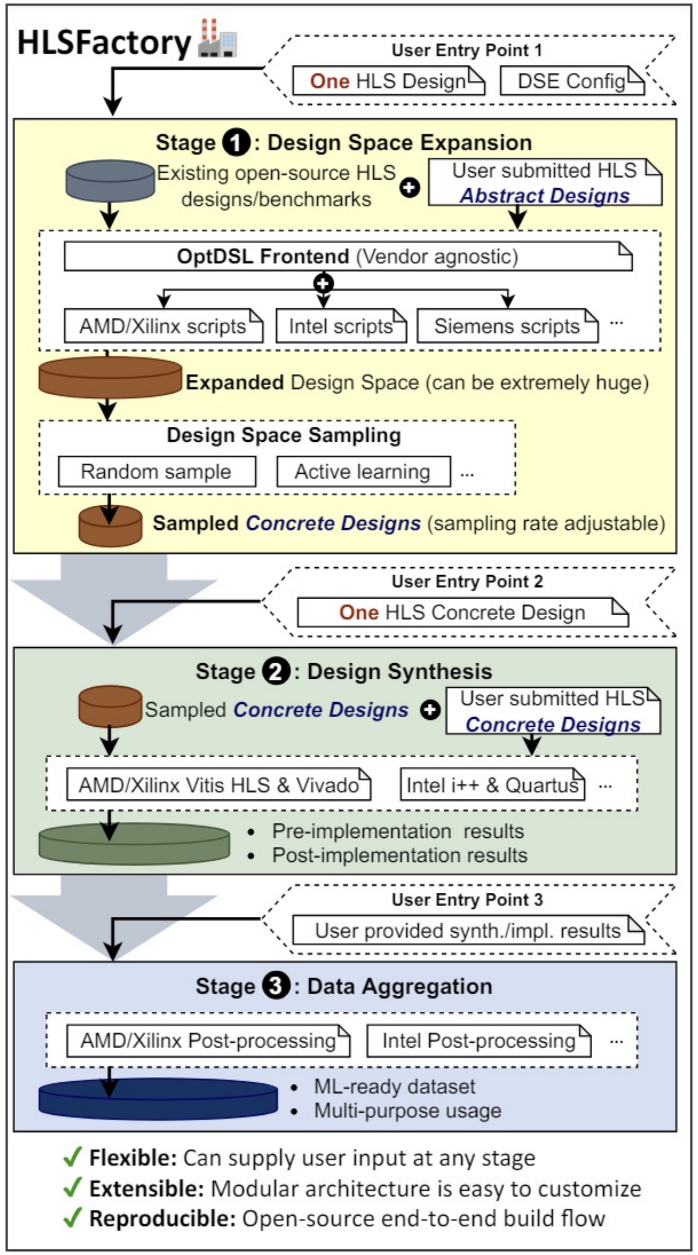
Existing dataset:

1. Small or homogeneous, contains only a subset of previously published HLS benchmark
2. The designs and intermediate/final tool outputs, which serve as important ML model features, are often reported organized in non-standard ad hoc ways
3. Challenging for external users to extend the dataset

Therefore, HLSFactory is proposed, and it boasts the following features:

1. Complete and easily extensible with user inputs at multiple stages
2. Diverse and comprehensive
3. Reproducible and user-friendly
4. ML-ready and multi-purpose
5. High performance and open-source

HLSFactory – Overview



HLSFactory
User Entry Point 1
One HLS Design
DSE Config
Stage 1: Design Space Expansion
Existing open-source HLS designs/benchmarks
User submitted HLS Abstract Designs
OptDSL Frontend (Vendor agnostic)
AMD/Xilinx scripts
Intel scripts
Siemens scripts
...
Expanded Design Space (can be extremely huge)
Design Space Sampling
Random sample
Active learning
...
Sampled Concrete Designs (sampling rate adjustable)
User Entry Point 2
One HLS Concrete Design
Stage 2: Design Synthesis
Sampled Concrete Designs
User submitted HLS Concrete Designs
AMD/Xilinx Vitis HLS & Vivado
Intel i++ & Quartus
...
Pre-implementation results
Post-implementation results
User Entry Point 3
User provided synth./impl. results
Stage 3: Data Aggregation
AMD/Xilinx Post-processing
Intel Post-processing
...
ML-ready dataset
Multi-purpose usage

✓ **Flexible:** Can supply user input at any stage
✓ **Extensible:** Modular architecture is easy to customize
✓ **Reproducible:** Open-source end-to-end build flow

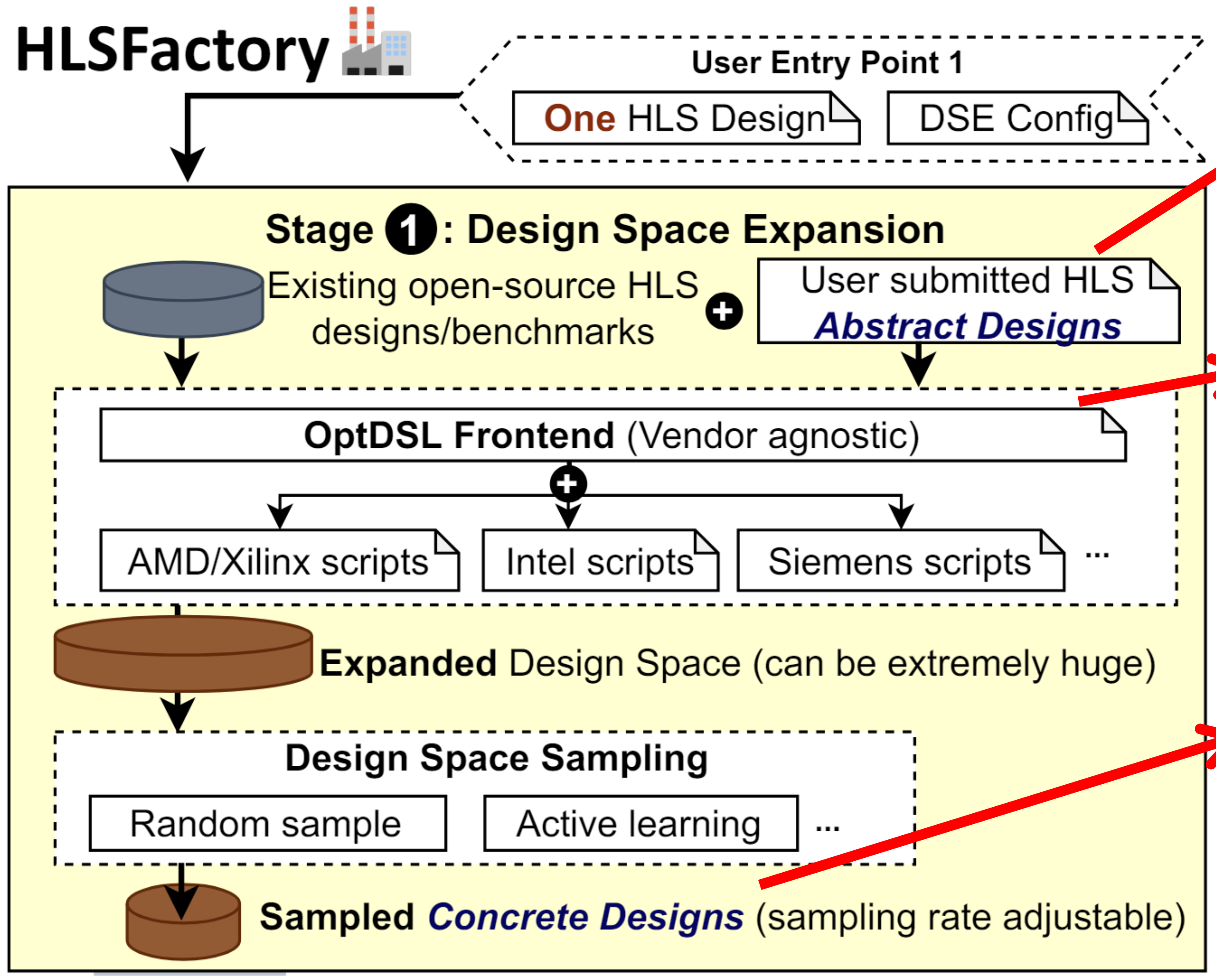
Stage 1: Design space expansion and sampling

Stage 2: Design Synthesis

Stage 3: Data extraction and Aggregation

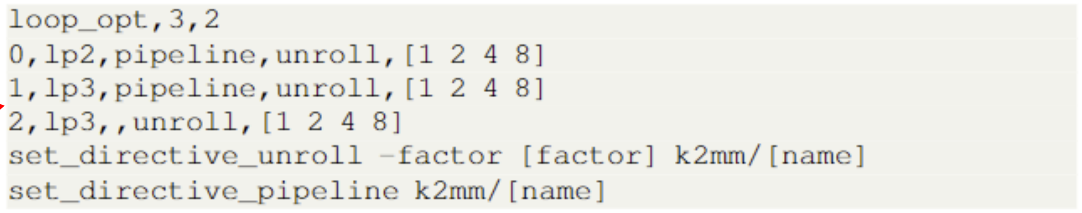
HLSFactory – Overview

Stage 1: Design Space Expansion And Sampling



HLSFactory
User Entry Point 1
One HLS Design
DSE Config
Stage 1: Design Space Expansion
Existing open-source HLS designs/benchmarks
User submitted HLS Abstract Designs
OptDSL Frontend (Vendor agnostic)
AMD/Xilinx scripts
Intel scripts
Siemens scripts
...
Expanded Design Space (can be extremely huge)
Design Space Sampling
Random sample
Active learning
...
Sampled Concrete Designs (sampling rate adjustable)

Vendor Agnostic: supports both AMD/Xilinx and Intel)



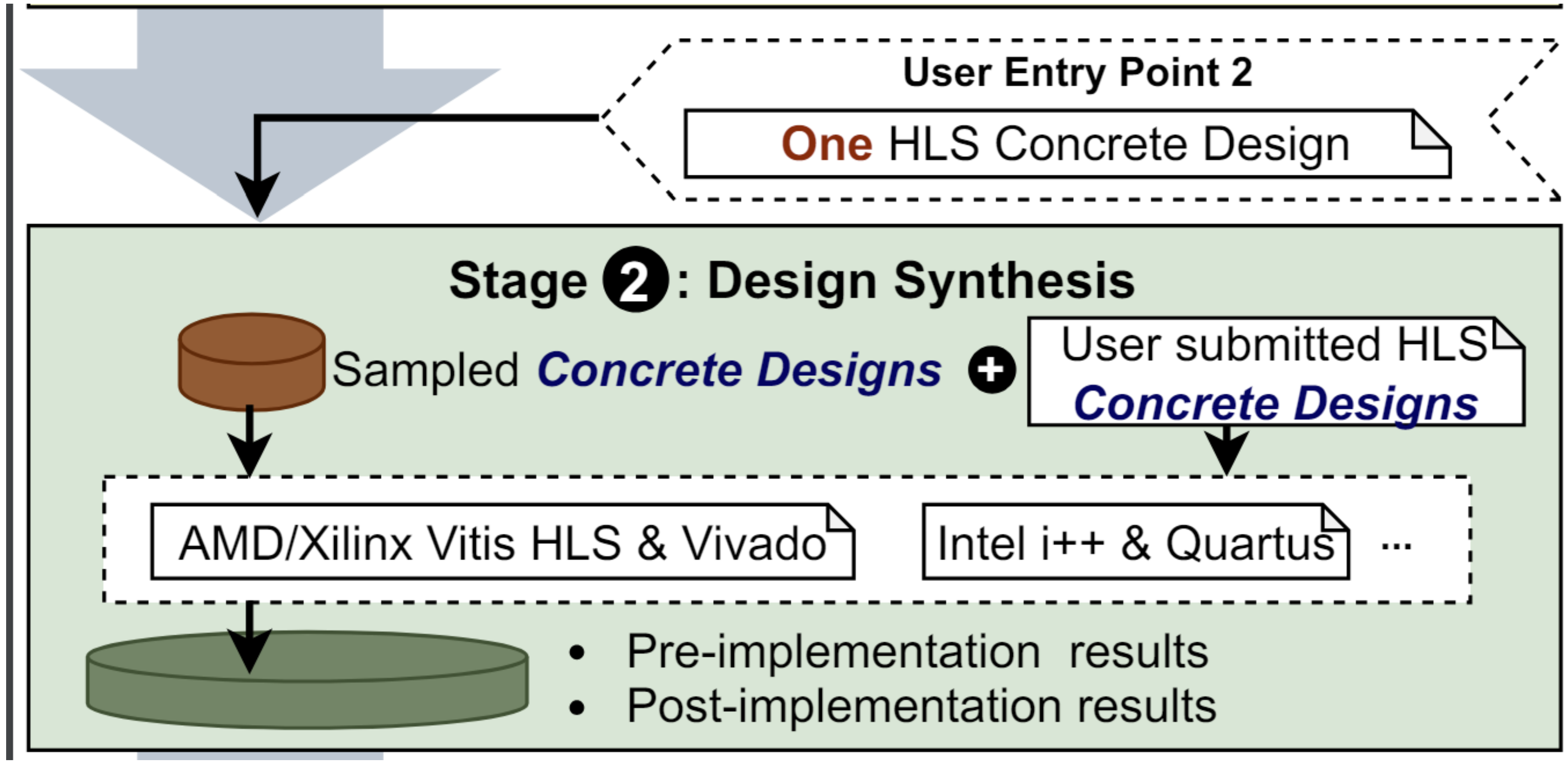
```
loop_opt,3,2
0,lp2,pipeline,unroll,[1 2 4 8]
1,lp3,pipeline,unroll,[1 2 4 8]
2,lp3,,unroll,[1 2 4 8]
set_directive_unroll -factor [factor] k2mm/[name]
set_directive_pipeline k2mm/[name]
```

OptDSL syntax helps to expand designs from benchmark

Vendor-specific concrete designs

HLSFactory – Overview

Stage 2: Design Synthesis



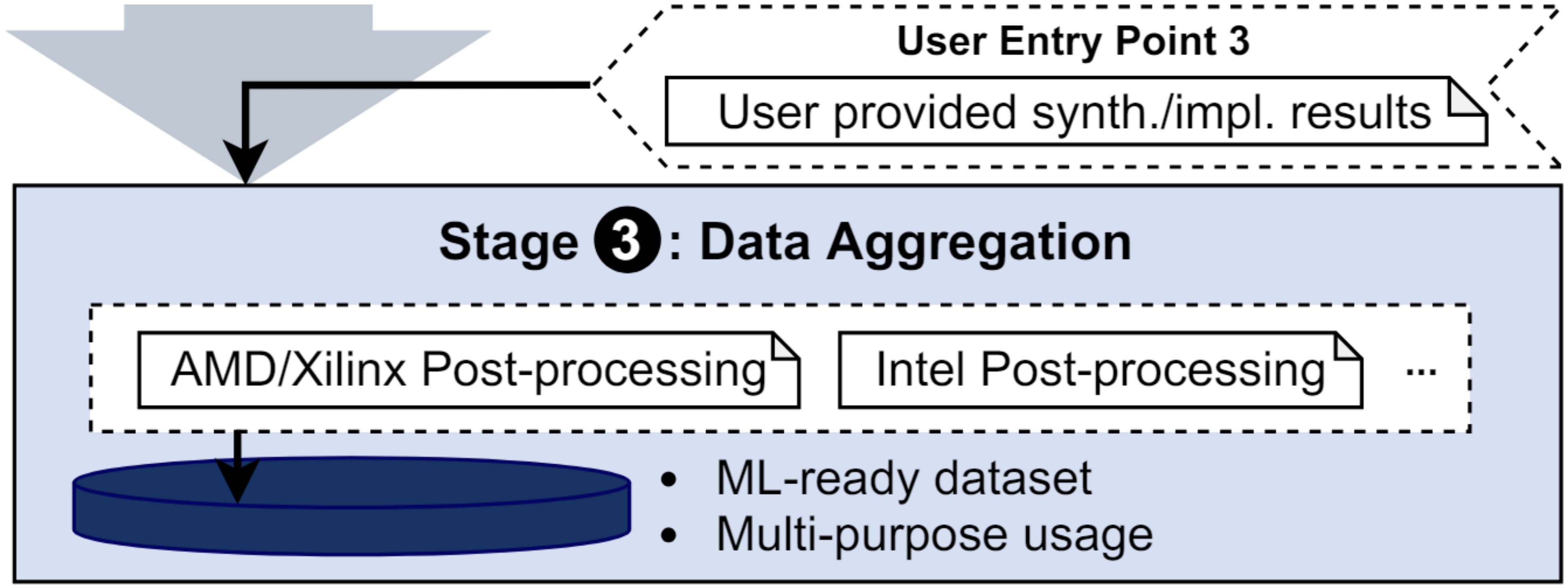
User Entry Point 2
One HLS Concrete Design
Stage 2: Design Synthesis
Sampled Concrete Designs
User submitted HLS Concrete Designs
AMD/Xilinx Vitis HLS & Vivado
Intel i++ & Quartus
...
• Pre-implementation results
• Post-implementation results

Two steps:

1. HLSSynth: synthesize HLS into RTL
2. HLSImpl: RTL code is implemented

HLSFactory – Overview

Stage 3: Data Extraction and Aggregation



User Entry Point 3
User provided synth./impl. results
Stage 3: Data Aggregation
AMD/Xilinx Post-processing
Intel Post-processing
...
ML-ready dataset
Multi-purpose usage

✓ **Flexible:** Can supply user input at any stage

✓ **Extensible:** Modular architecture is easy to customize

✓ **Reproducible:** Open-source end-to-end build flow

HLSFactory – Overview

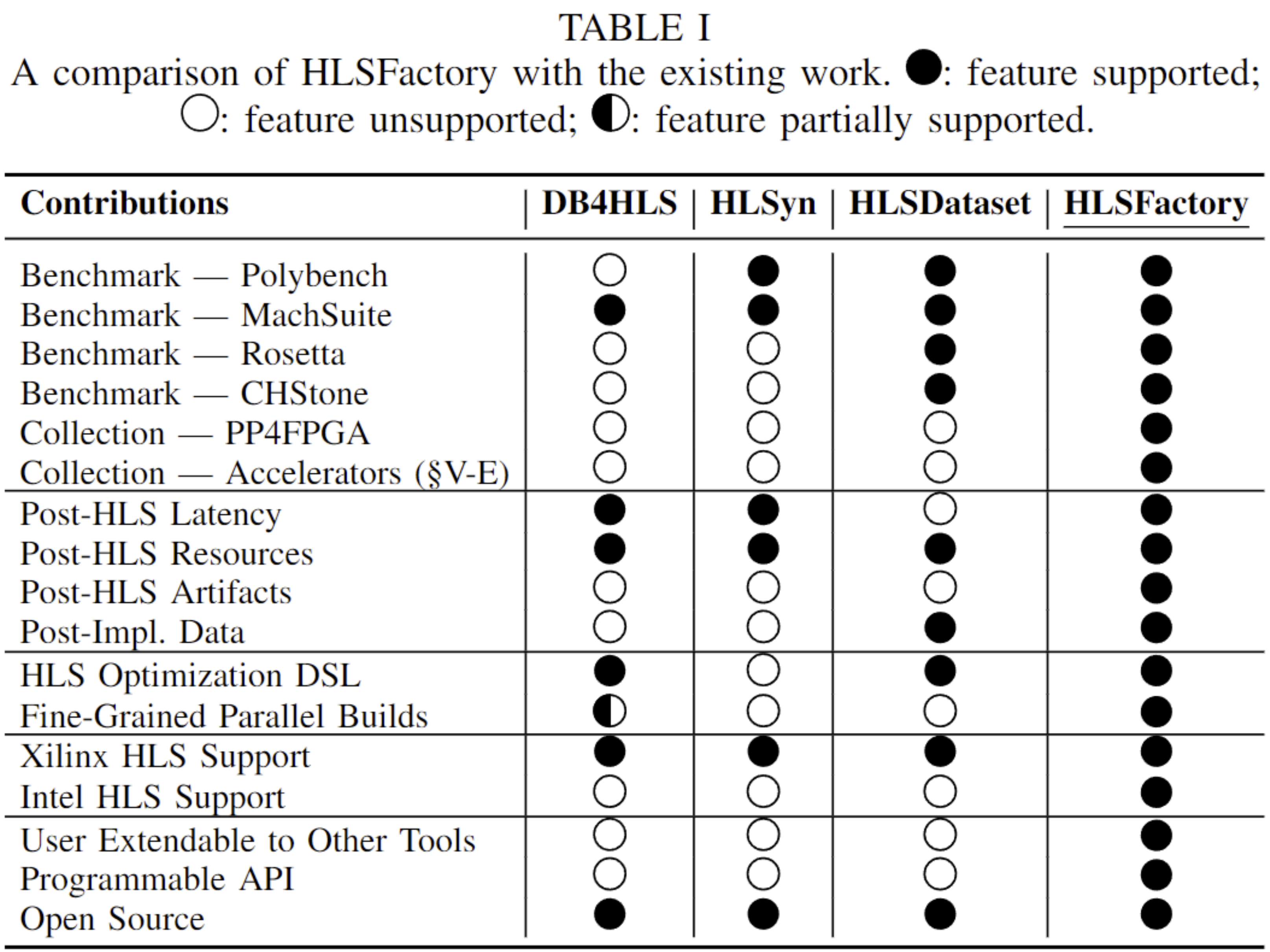


TABLE I
A comparison of HLSFactory with the existing work. ●: feature supported; ○: feature unsupported; ◐: feature partially supported.

Contributions	**DB4HLS**	**HLSyn**	**HLSDataset**	**<u>HLSFactory</u>**
Benchmark — Polybench	○	●	●	●
Benchmark — MachSuite	●	●	●	●
Benchmark — Rosetta	○	○	●	●
Benchmark — CHStone	○	○	●	●
Collection — PP4FPGA	○	○	○	●
Collection — Accelerators (§V-E)	○	○	○	●
Post-HLS Latency	●	●	○	●
Post-HLS Resources	●	●	●	●
Post-HLS Artifacts	○	○	○	●
Post-Impl. Data	○	○	●	●
HLS Optimization DSL	●	○	●	●
Fine-Grained Parallel Builds	◐	○	○	●
Xilinx HLS Support	●	●	●	●
Intel HLS Support	○	○	○	●
User Extendable to Other Tools	○	○	○	●
Programmable API	○	○	○	●
Open Source	●	●	●	●

HLSFactory – Implementation & Usage

Python API

API Functions	Description

`class Design`
`class Dataset`
	Single HLS design
	Multiple HLS designs

`class Flow(ABC)`
`Flow.execute(design)`
`Flow.execute_datasets_parallel(design)`
	Abstract class for arbitrary design flow
	Execute a flow on one design
	Execute a flow on many designs

`class Frontend(Flow)`
`class OptDSLFrontend(Frontend)`
	Abstract class for frontend design expansion
	Opt DSL frontend for Xilinx HLS designs

`class ToolFlow(Flow)`
`class VitisHLSSynthFlow(ToolFlow)`
`class VitisHLSImplFlow(ToolFlow)`
`class VitisHLSImplReportFlow(ToolFlow)`
	Abstract class for EDA tool
	Run Vitis HLS synthesis
	Run Vivado implementation (via Vitis HLS)
	Run Vivado reporting

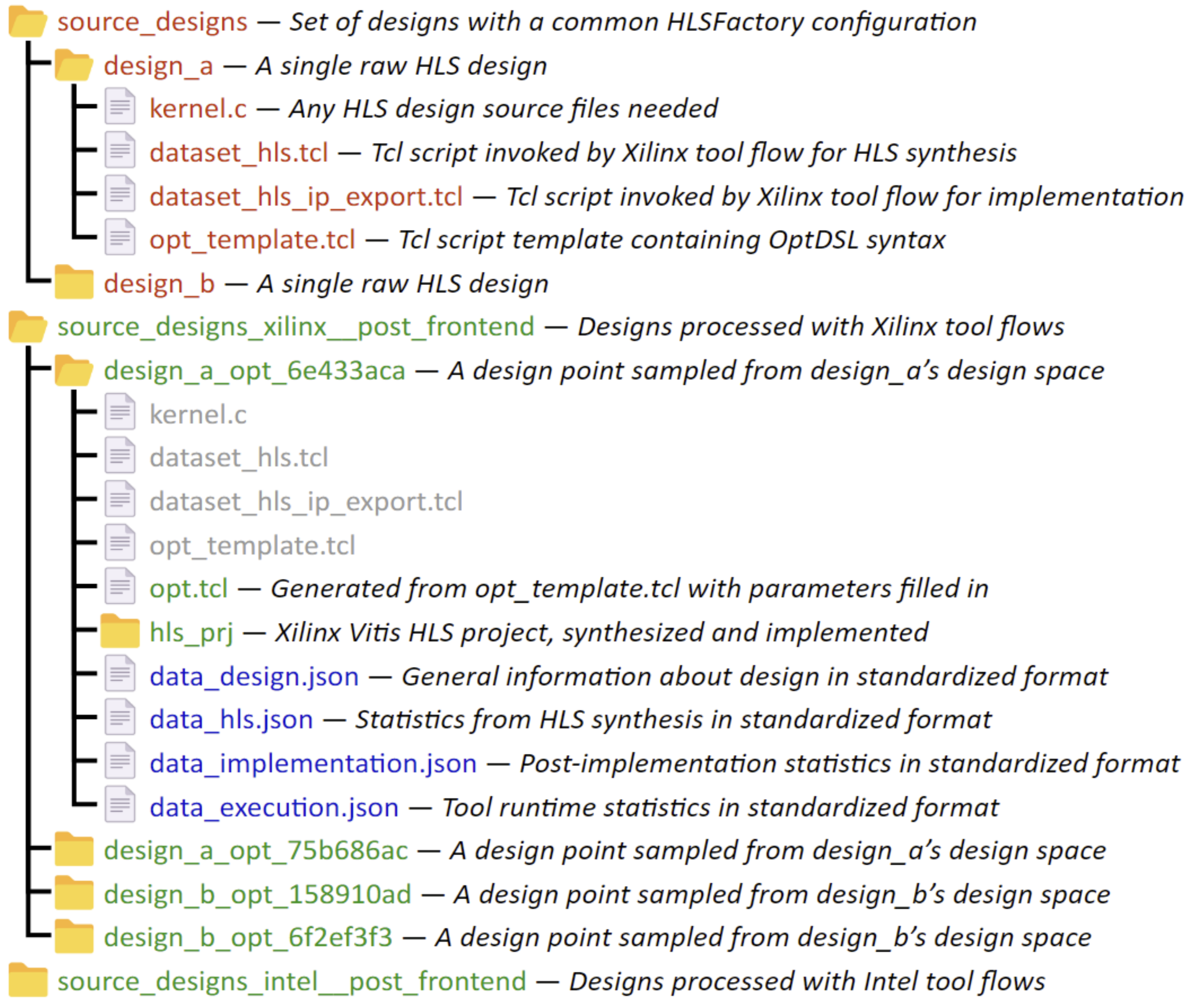
Example Use of the APIs

```
opt_dsl = OptDSLFrontend(WORK_DIR, random_sample=True,
                         random_sample_num=N_RANDOM_SAMPLES)
hls_synth = VitisHLSSynthFlow()
hls_impl = VitisHLSImplFlow()
hls_impl_report = VitisHLSImplReportFlow()

datasets_post_frontend = opt_dsl.execute_datasets_parallel(
   datasets, n_jobs=N_JOBS)
datasets_post_synth = hls_synth.execute_datasets_parallel(
   datasets_post_frontend, n_jobs=N_JOBS)
datasets_post_hls_impl = hls_impl.execute_datasets_parallel(
   datasets_post_synth, n_jobs=N_JOBS)
hls_impl_report.execute_datasets_parallel(
   datasets_post_hls_impl, n_jobs=N_JOBS)
```

Backend is running in parallel

HLSFactory – Implementation & Usage



```
source_designs — Set of designs with a common HLSFactory configuration
├─ design_a — A single raw HLS design
│  ├─ kernel.c — Any HLS design source files needed
│  ├─ dataset_hls.tcl — Tcl script invoked by Xilinx tool flow for HLS synthesis
│  ├─ dataset_hls_ip_export.tcl — Tcl script invoked by Xilinx tool flow for implementation
│  └─ opt_template.tcl — Tcl script template containing OptDSL syntax
└─ design_b — A single raw HLS design
source_designs_xilinx__post_frontend — Designs processed with Xilinx tool flows
├─ design_a_opt_6e433aca — A design point sampled from design_a's design space
│  ├─ kernel.c
│  ├─ dataset_hls.tcl
│  ├─ dataset_hls_ip_export.tcl
│  ├─ opt_template.tcl
│  ├─ opt.tcl — Generated from opt_template.tcl with parameters filled in
│  ├─ hls_prj — Xilinx Vitis HLS project, synthesized and implemented
│  ├─ data_design.json — General information about design in standardized format
│  ├─ data_hls.json — Statistics from HLS synthesis in standardized format
│  ├─ data_implementation.json — Post-implementation statistics in standardized format
│  └─ data_execution.json — Tool runtime statistics in standardized format
├─ design_a_opt_75b686ac — A design point sampled from design_a's design space
├─ design_b_opt_158910ad — A design point sampled from design_b's design space
└─ design_b_opt_6f2ef3f3 — A design point sampled from design_b's design space
source_designs_intel__post_frontend — Designs processed with Intel tool flows
```

Fig. 4. The directory structure that HLSFactory uses. Red are input files; green are the intermediate design points; blue are output files.

Design Directory Structure

Shows specific entry points scripts that users add to integrate into HLSFactory

Case Study 1: ML Prediction Of Post-implementation Quality-of-Results (QoR)

Generating more data points using HLSFactory can result in higher prediction accuracy

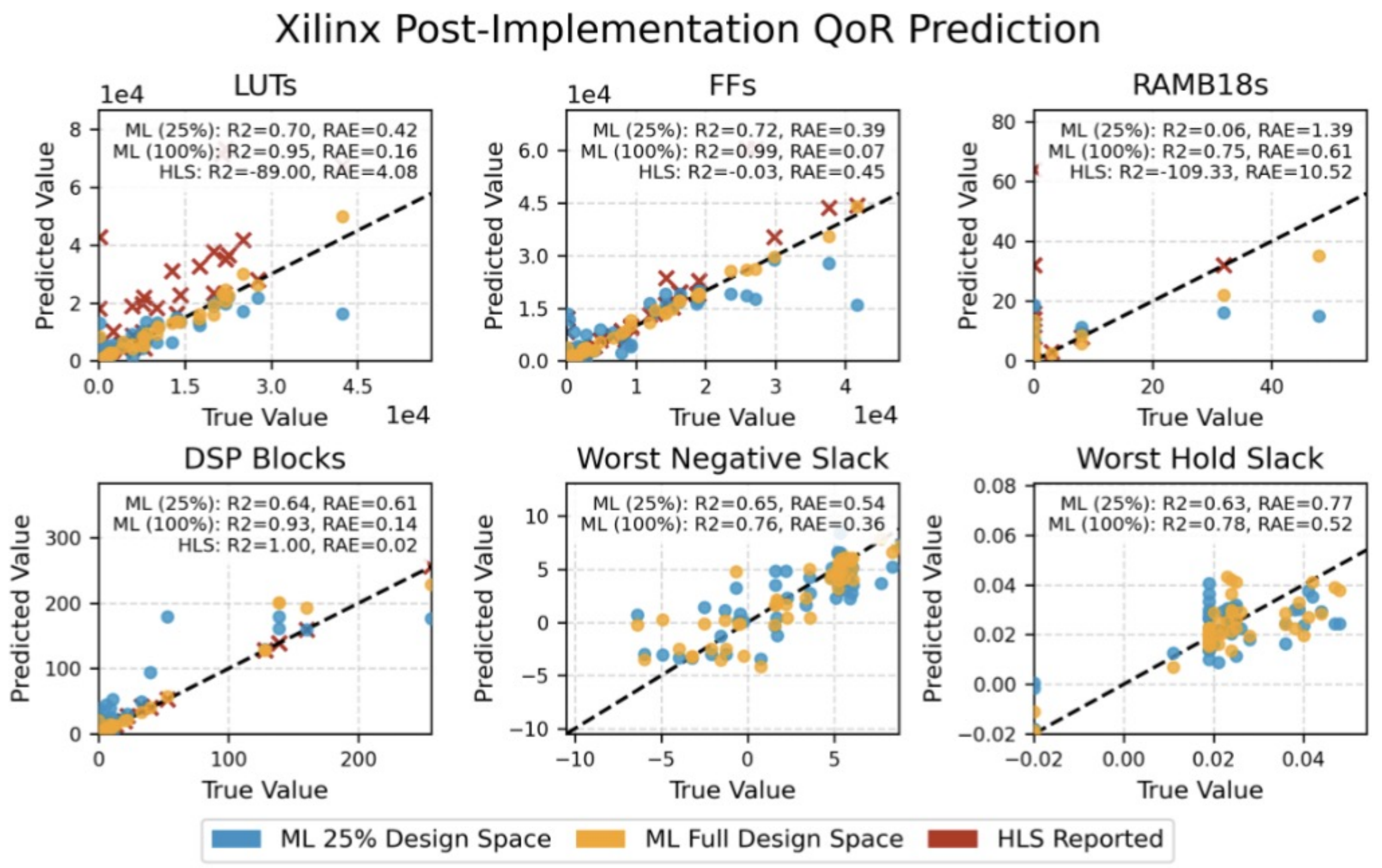
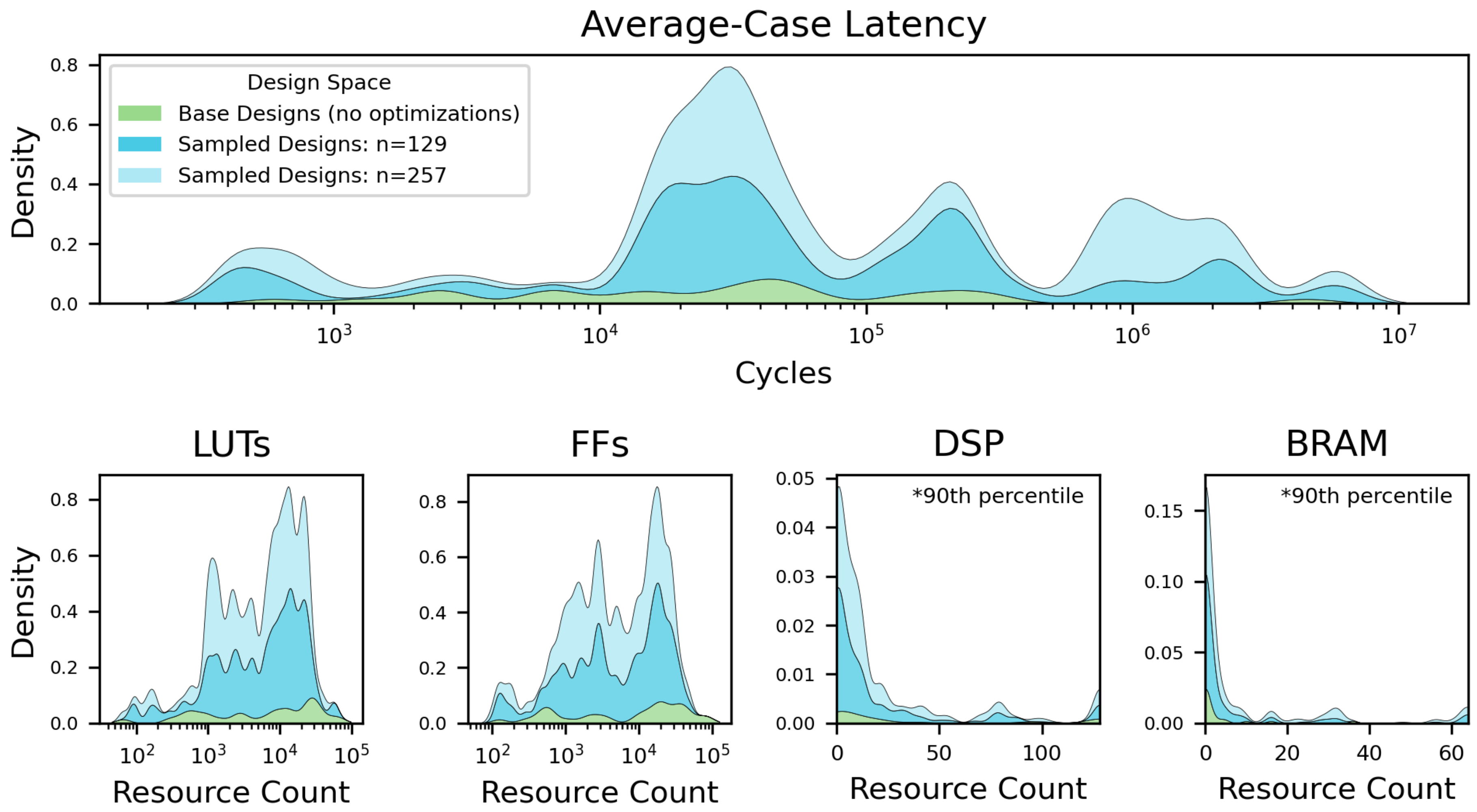


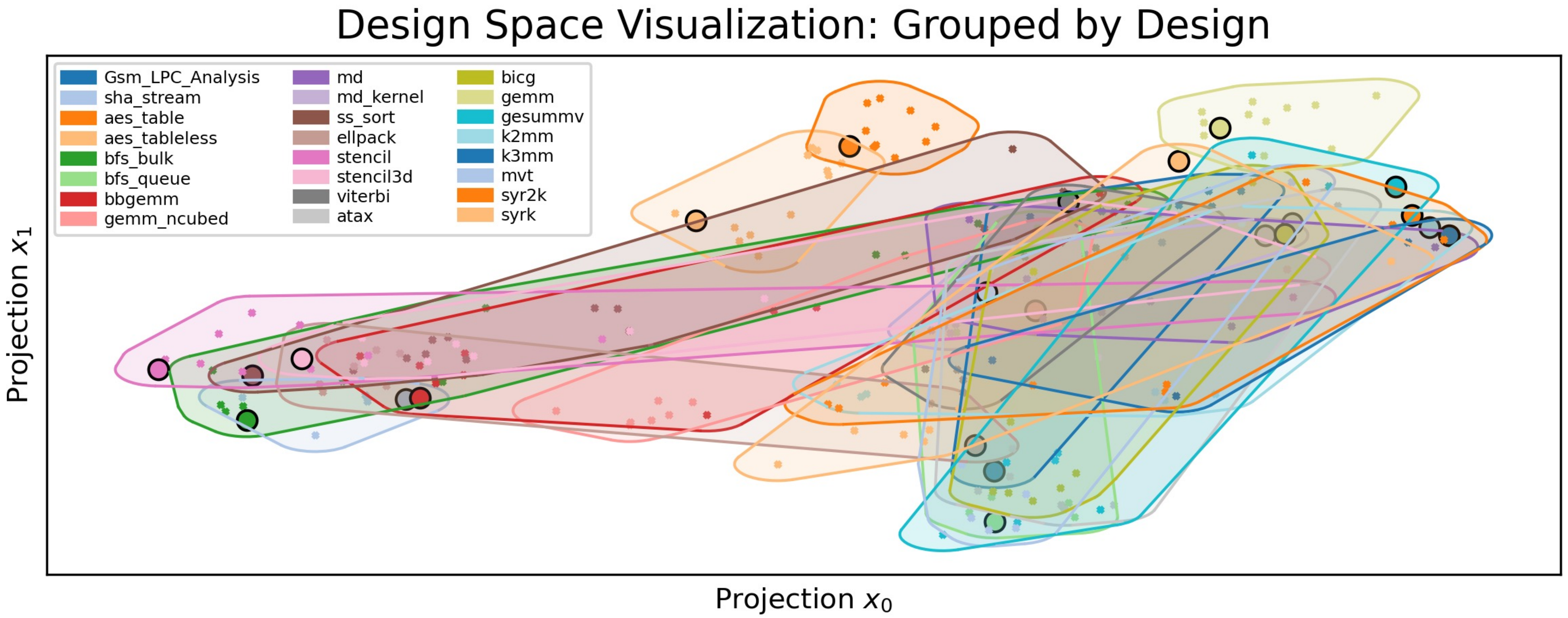
Fig. 5. True-vs-predicted plots for the HLS-based ML QoR model. Test values are shown for models trained on the complete and partial subset of the training design space. "RAE": Relative Absolute Error ($|\hat{y}-y|/|y-\bar{y}|$), "R2": Coefficient of Determination

Case Study 2: Design Space Coverage

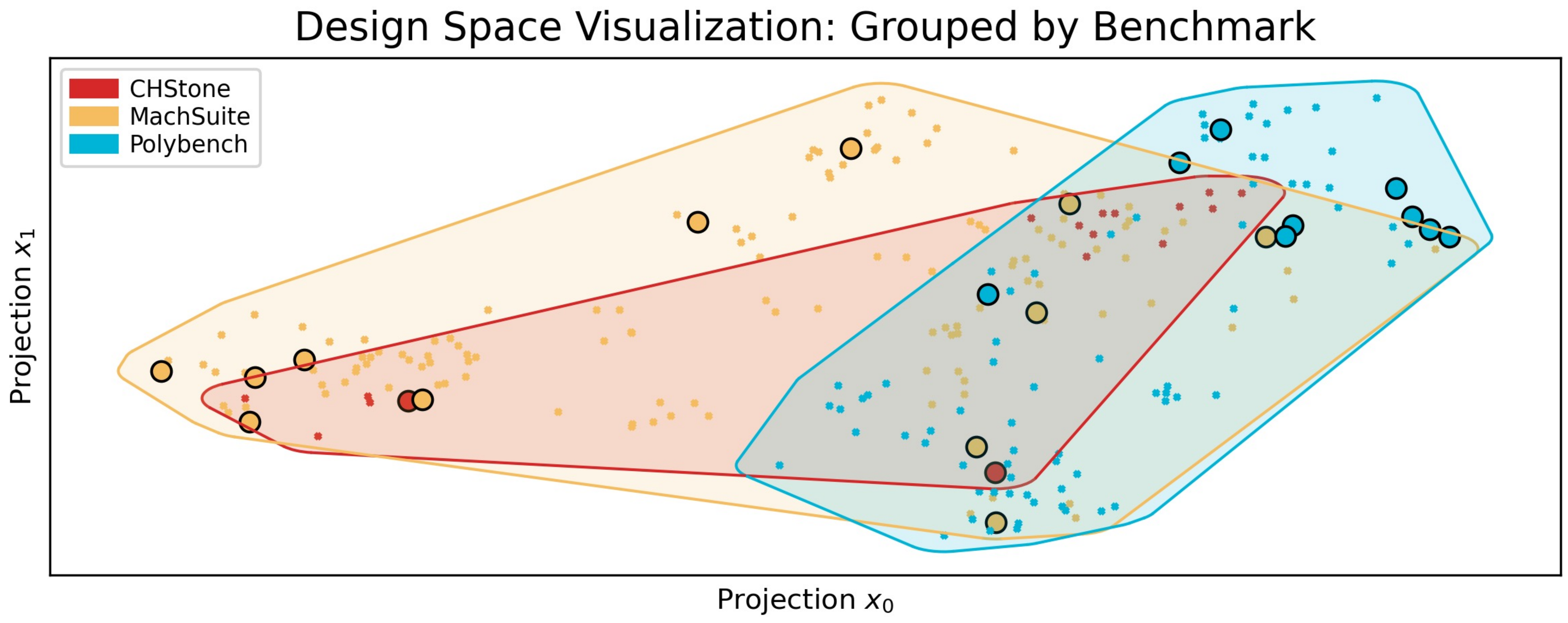


Effect of design sampling to cover more design space. Sampled designs cover a wider range of metrics than base designs with no optimizations. Latency is HLS estimated; resources are post-implementation. Note that these are stacked density plots to show the effect of cumulative design sampling.

Case Study 2: Design Space Coverage

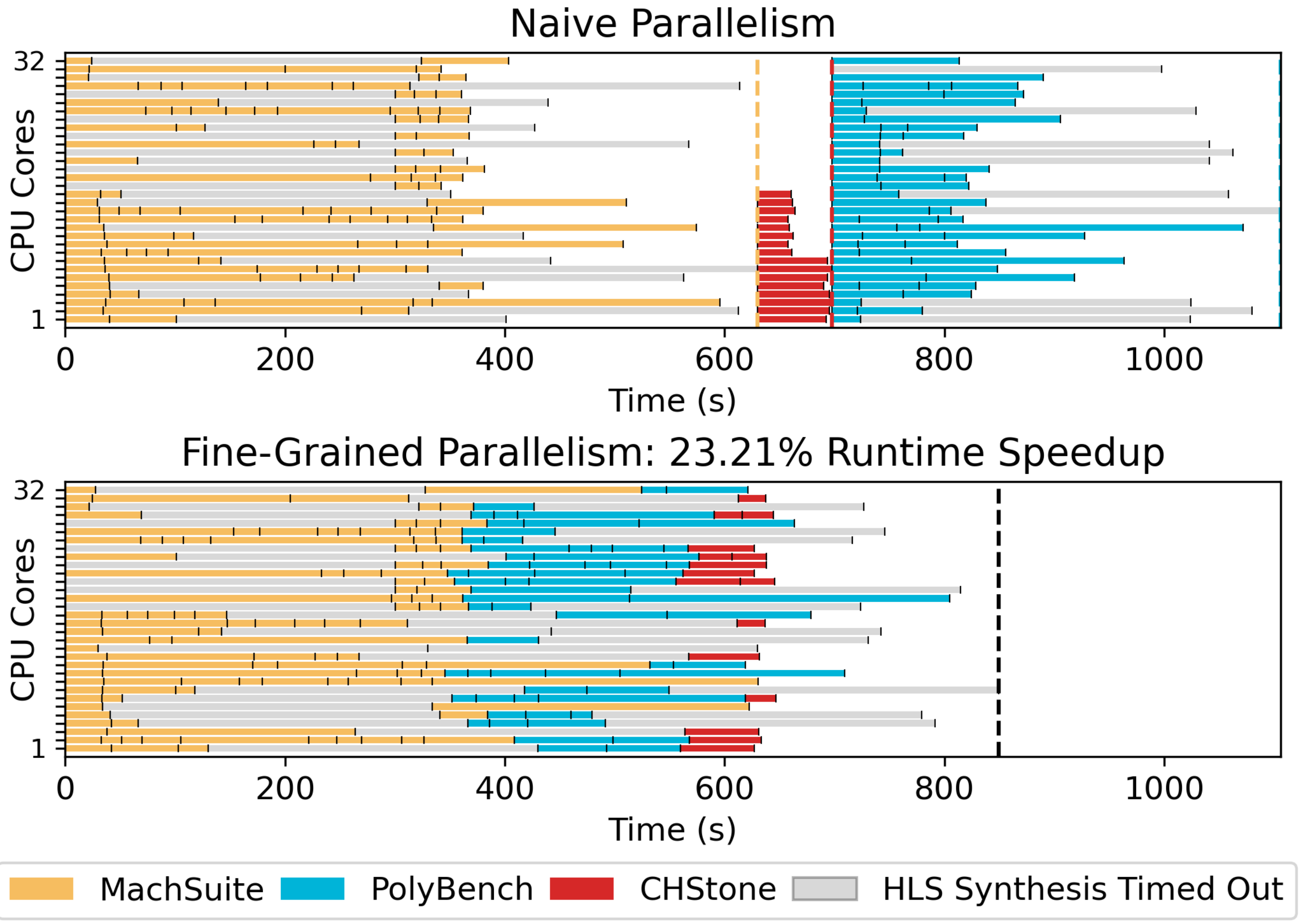


Case Study 2: Design Space Coverage

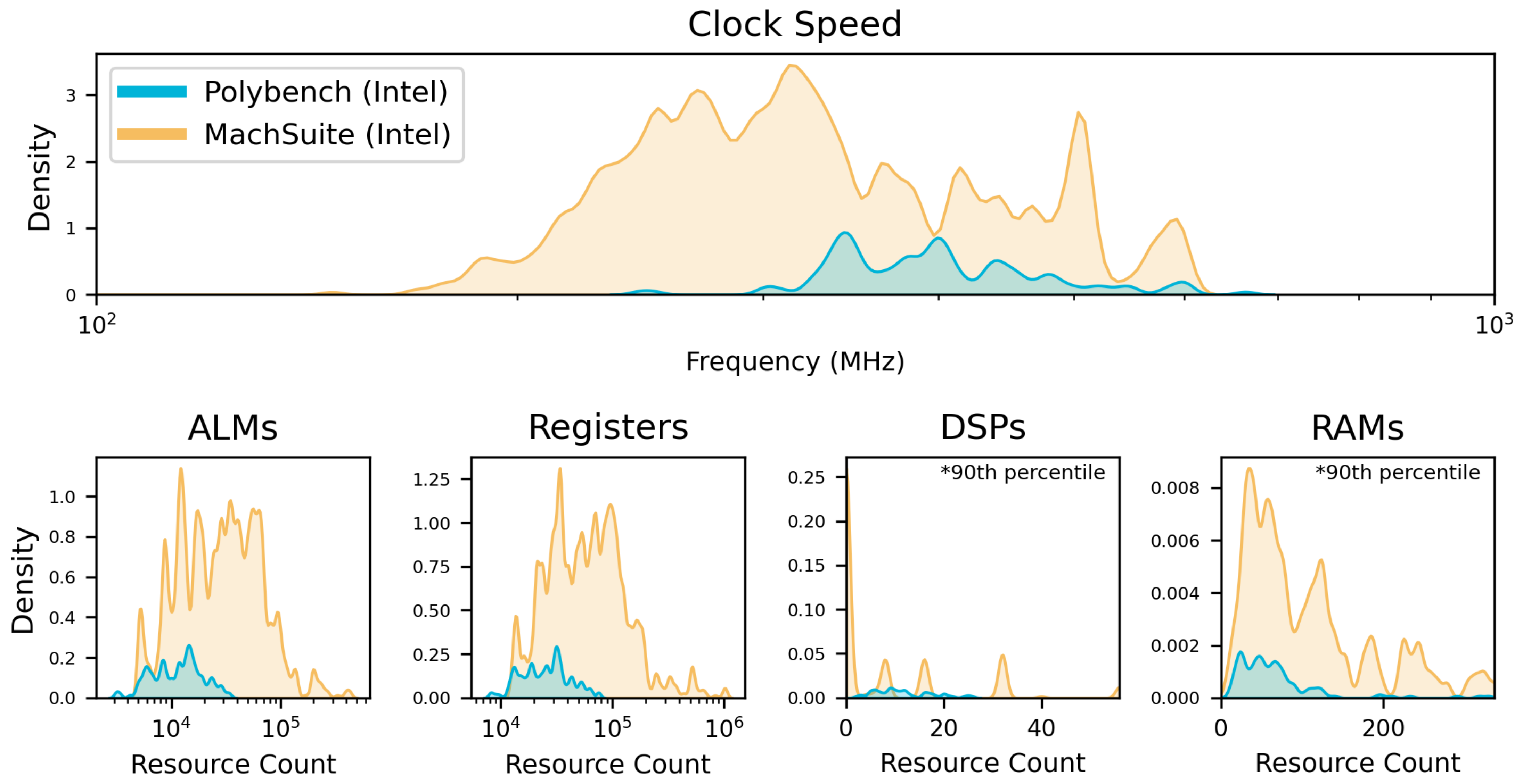


Case Study 3: Speedup of Fine-Grained Design Parallelism

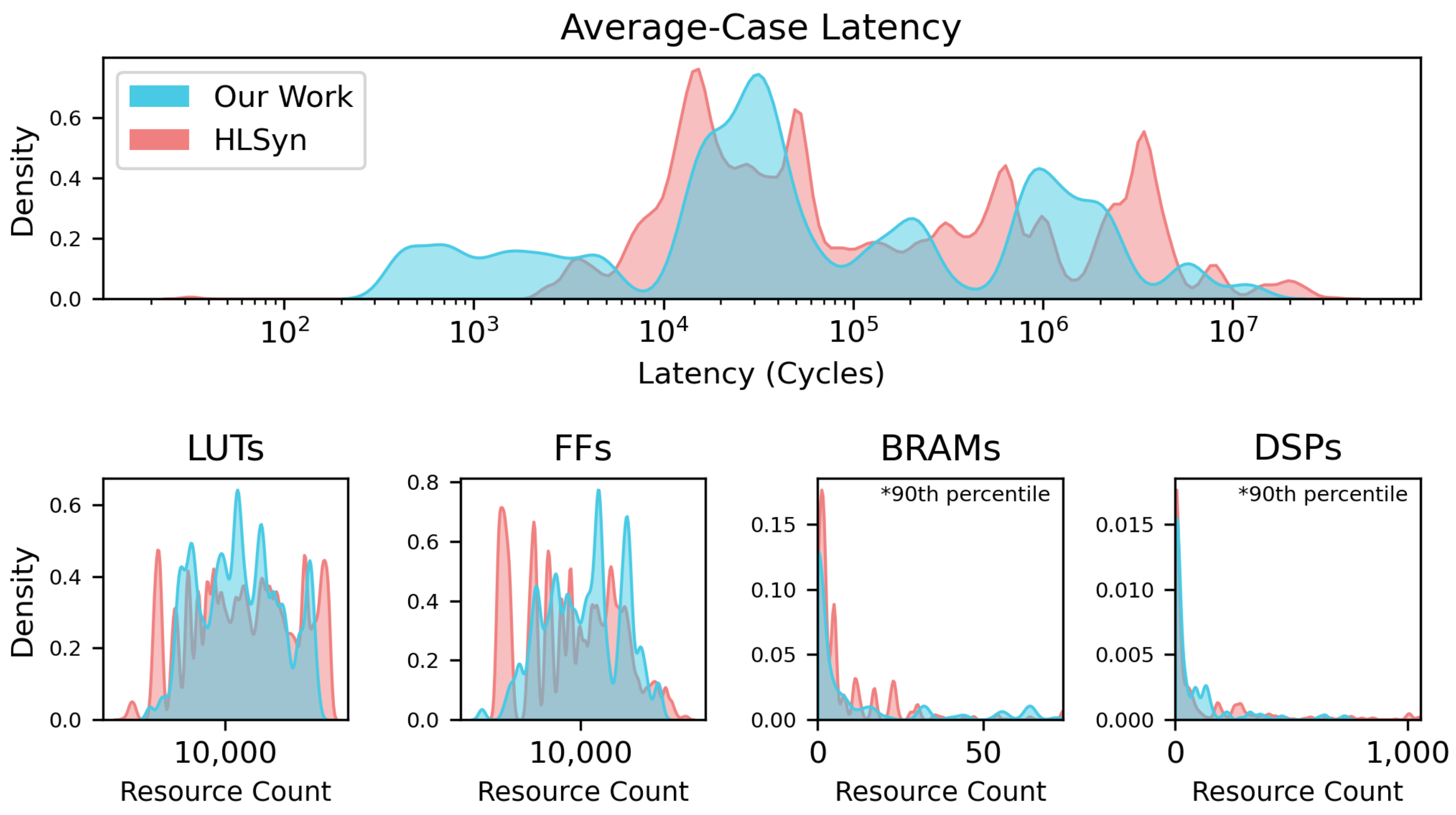
Parallel execution of Vitis HLS synthesis. Top panel shows core utilization over time with naive parallelism across datasets; bottom panel shows our fine-grained design parallelism across datasets.



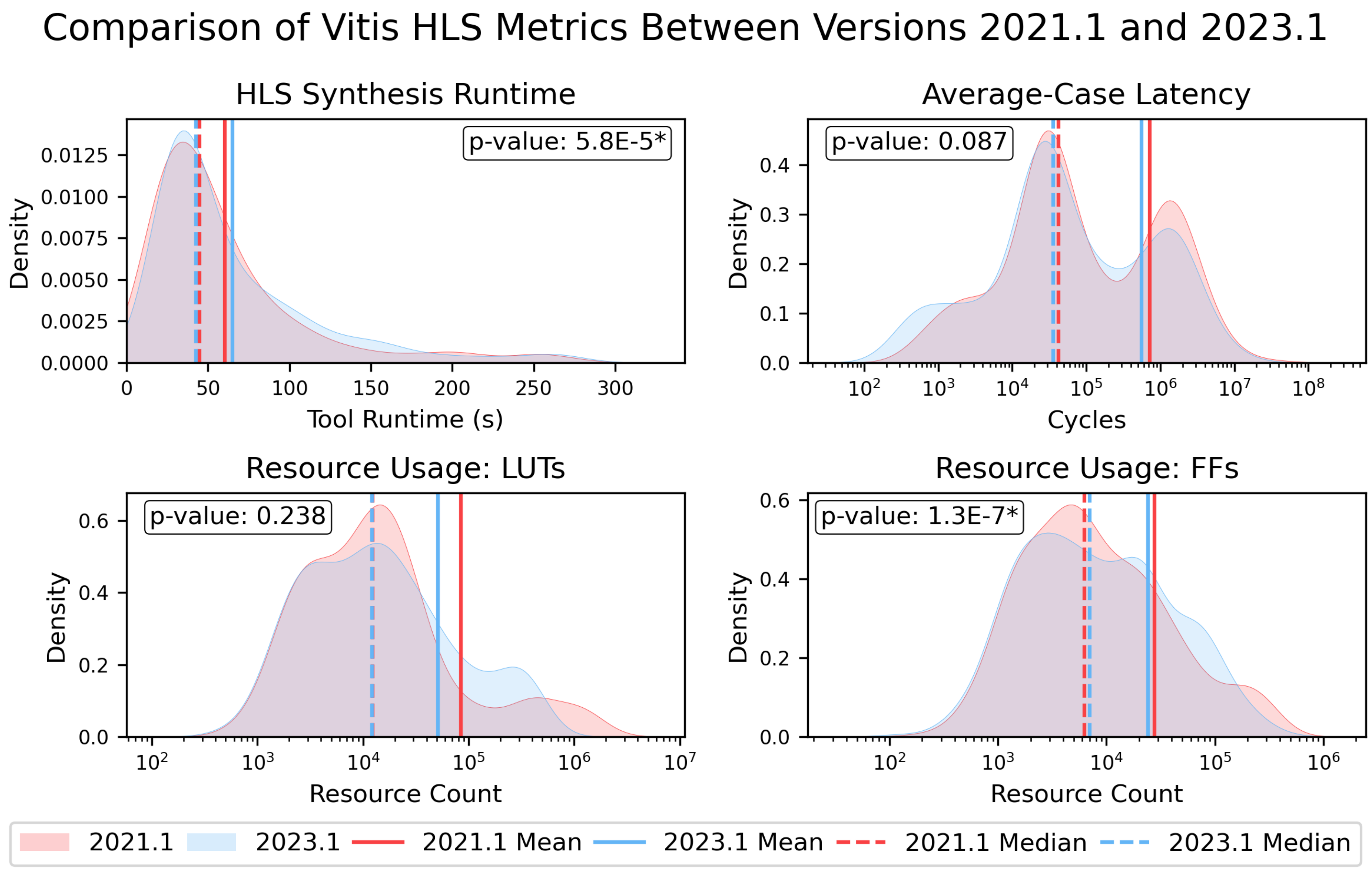
Case Study 4: Targeting Different Vendors (Intel)



Case Study 5:Integrating Released Data From Other Works



Case Study 6: Regression Benchmark of HLS Synthesis Tools



Distribution of HLS tool metrics from two versions of Vitis HLS

Conclusion

HLSFactory Key Points:

1. Complete and easily extensible with user inputs at multiple stages
2. Diverse and comprehensive
3. Reproducible and user-friendly
4. ML-ready and multi-purpose
5. High performance and open-source (available at https://github.com/sharc-lab/HLSFactory)

Future Direction:

- Simulation Flows, e.g. vendor supported co-simulation or with our own published tools like LightningSim (which is co-sim accurate and much faster)
- More designs to add from others in the academic community and ope- source
- Developing more frontends and vendor agnostic to abstractions to enumerate more designs from different design spaces
- ***Ex: An HLS4ML frontend to enumerate HLS designs from HLS4ML model specs or from ONNX models***

Georgia Tech

HLS Factory

Documentation + Tutorials

sharc-lab.github.io/HLSFactory/docs/

GitHub Repository

github.com/sharc-lab/hlsfactory

ArXiv Pre-Print Paper

arxiv.org/abs/2405.00820

Thanks!

Questions?

References

[1] H. Mohammadi Makrani, F. Farahmand, H. Sayadi, S. Bondi, S. M.Pudukotai Dinakarrao, H. Homayoun, and S. Rafatirad, “Pyramid: Machine learning framework to estimate the optimal timing and resource usage of a high-level synthesis design,” in 2019 29th International Conference on Field Programmable Logic and Applications (FPL), 2019.
[2] S. Dai, Y. Zhou, H. Zhang, E. Ustun, E. F. Young, and Z. Zhang, “Fast and accurate estimation of quality of results in high-level synthesis with machine learning,” in 2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). IEEE, 2018, pp. 129–132.
[3] D. Liu and B. C. Schafer, “Efficient and reliable high-level synthesis design space explorer for fpgas,” in 2016 26th International Conference on Field Programmable Logic and Applications (FPL), 2016.
[4] W. Haaswijk, E. Collins, B. Seguin, M. Soeken, F. Kaplan, S. S üsstrunk, and G. De Micheli, “Deep learning for logic optimization algorithms,” in 2018 IEEE International Symposium on Circuits and Systems (ISCAS), 2018.
[5] Y. Luo, C. Tan, N. B. Agostini, A. Li, A. Tumeo, N. Dave, and T. Geng, “Ml-cgra: An integrated compilation framework to enable efficient machine learning acceleration on cgras,” in 2023 60th ACM/IEEE Design Automation Conference (DAC), 2023.
[6] V. A. Chhabria, Y. Zhang, H. Ren, B. Keller, B. Khailany, and S. S. Sapatnekar, “Mavirec: Ml-aided vectored ir-drop estimation and classification,” in 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2021.
[7] R. G. Kim, J. R. Doppa, and P. P. Pande, “Machine learning for design space exploration and optimization of manycore systems,” in 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2018, pp. 1–6.
[8] Z. Lin, J. Zhao, S. Sinha, and W. Zhang, “HL-Pow: A Learning-Based Power Modeling Framework for High-Level Synthesis,” in 25th Asia and South Pacific Design Automation Conference (ASP-DAC), 2020.
[9] G. Singha, D. Diamantopoulosb, J. G ómez-Lunaa, S. Stuijkc, H. Corporaalc, and O. Mutlu, “LEAPER: Fast and Accurate FPGA-based System Performance Prediction via Transfer Learning,” in IEEE 40th International Conference on Computer Design (ICCD), 2022.
[10] Z. Lin, Z. Yuan, J. Zhao, W. Zhang, H. Wang, and Y. Tian, “Powergear: Early-stage power estimation in fpga hls via heterogeneous edge-centric gnns,” in Design, Automation & Test in Europe Conference & Exhibition (DATE), 2022.