Toward End-to-End Adaptive Application Decomposition and Execution Across Distributed Computing Infrastructures

Guilherme Mendes Vieira de Matos
guilherme.matos@estudante.ufscar.br
Federal University of Sao Carlos
Sorocaba, Sao Paulo, Brazil

Abstract

Dennard scaling slowdown and Moore's Law limits have broken CPU-data growth coupling, while disaggregated infrastructures combine CPUs, GPUs, FPGAs, and SmartNICs across the computing continuum from devices to cloud. Applications require fine-grained decomposition, cost-aware orchestration, and portable execution. Recent LLVM/MLIR-based decomposition improves CNN cache utilization to less than 50% but exhaustive profiling doesn't scale and leaves placement/execution unsolved. This work in progress proposes a pipeline that builds DFGs from LLVM/MLIR IR, identifies kernels via cosine similarity, predicts per-kernel time via XGBoost, and implement kernels using Wasm component model. CNN experiments validate kernel identification accuracy/speed and high-fidelity cost prediction, establishing foundation for scalable decomposition across heterogeneous continuum infrastructures.

ACM Reference Format:
Guilherme Mendes Vieira de Matos. 2026. Toward End-to-End Adaptive Application Decomposition and Execution Across Distributed Computing Infrastructures. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (SIGMETRICS '26)*. ACM, New York, NY, USA, 2 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Dennard scaling slowdown and the effective end of Moore's Law have broken the historical coupling between general-purpose CPU performance and data growth [5], while I/O and accelerator capabilities continue to increase rapidly in both cloud and edge environments. Recent infrastructures now combine CPUs, GPUs, FPGAs, SmartNICs, and programmable switches, and 5G/edge deployments add highly heterogeneous and resource-constrained nodes [1]. In this scenario, deciding what parts of an application to run, where to run them, and how to execute them efficiently and portably becomes a key performance challenge, directly impacting latency, throughput, energy, and resource utilization.

Within this emerging computing continuum (from devices through edge to the cloud), both general-purpose applications and AI-RAN workloads increasingly depend on fine-grained decomposition, orchestration aware of costs and constraints, and portable execution across heterogeneous platforms, making automated analysis of code structure and cost-intelligent placement essential to fully exploit these distributed resources [4]. Recent works show that applications can be split into smaller functions using static analysis over LLVM/MLIR intermediate representations (IR) and then profiled dynamically across different hardware configurations. For convolutional neural network (CNN) workloads, this approach exposes a rich design space of execution plans and can improve cache utilization by more than 50% while preserving end-to-end execution time, enabling better performance–energy trade-offs on disaggregated CPU systems [6].

Despite these advances, exhaustive profiling is not scalable in large, highly dynamic environments, and remains expensive when the number of possible hardware combinations grows. Moreover, existing approaches do not fully address where to run each part of the application in the computing continuum, nor how to execute it efficiently and portably across heterogeneous infrastructures, especially under strict latency and energy constraints typical of those environments.

This work targets the following end-to-end research question: given a real application, how can we (i) discover and characterize its internal computational structure, (ii) estimate time and energy for each computational kernel on multiple hardware targets, (iii) select the best execution plan(s) that satisfy a given SLA, and (iv) execute these components efficiently and portably across heterogeneous infrastructures. We address this by integrating three lines of work: static and dynamic functional decomposition of code, orchestration driven by metrics and costs across the computing continuum, and the use of the WebAssembly (Wasm) component model as a portable, low-overhead execution substrate.

2 Methodology

As can be seen in Figure 1, we propose a pipeline with three main stages: (1) code understanding and kernel identification (yellow blocks), (2) cost prediction and SLA-aware plan selection (blue and green blocks), and (3) metrics-driven orchestration with portable execution (salmon blocks).

Code understanding and kernel identification: Starting from source code, we leverage LLVM/MLIR IR to construct a Dataflow Graph (DFG) representing the application's structure. We identify and isolate candidate kernels from the DFG, mapping each subgraph into a vector representation. By applying cosine similarity against a library of pre-characterized kernel vectors, we identify the highest-probability match, effectively mapping the application's DFG to a sequence of typed operations (e.g., convolution, ReLU, and batch normalization).

SIGMETRICS '26, Ann Arbor, MI

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/XXXXXXX.XXXXXXX

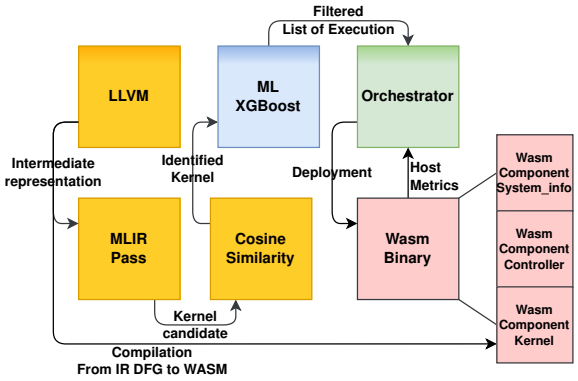

Figure 1: Pipeline workflow.

Cost prediction and SLA-aware plan selection: The identified kernels and their features are passed to a pre-trained machine learning model that predicts, for each kernel and each available hardware target, the expected execution time and energy consumption. For a full model or application composed of multiple kernels, we aggregate these per-kernel predictions to derive end-to-end time and energy profiles for different hardware assignment strategies. Given an SLA (e.g., a maximum latency bound or a minimum-energy objective), we enumerate feasible execution plans and apply a first filter that keeps only those plans that satisfy the SLA. The output of this stage is a set of candidate execution plans, each specifying a mapping from kernels to hardware types and the corresponding predicted time and energy.

Metrics-driven orchestration and portable execution: The orchestrator receives (i) the list of SLA-compliant execution plans and (ii) live health metrics from the infrastructure, where each node periodically reports its current CPU and memory utilization and network usage. Based on these metrics, the orchestrator refines the mapping from hardware types to specific nodes. The DFG accompanying each plan preserves dependencies and indicates the next kernel to execute, guiding the orchestrator through the full execution graph.

Each node runs the same WebAssembly binary comprising three components: Component (1), implemented in Rust, monitors local host metrics (CPU, memory, network) and periodically reports them to the orchestrator. Component (2), also in Rust, receives execution instructions and data (inputs, weights) from the orchestrator. Component (3) executes the selected kernel, consuming the data delivered by component (2). For known kernels identified by dataflow graph matching (e.g., conv2D, GEMM), the system directly instantiates the corresponding pre-optimized C/Wasm component from its repository for maximum performance; for unknown kernels, the extracted IR is compiled on-demand to Wasm using LLVM/MLIR tools, ensuring immediate execution regardless of origin. This hybrid approach delivers a unified wasm binary deployable via Wasmtime/WASI on CPUs, WebGPU on GPUs, or Wasm2RTL runtimes on FPGAs, providing automatic optimization when available, universal fallback for novel cases, and full hardware portability while maintaining reusable preprocessing

These three components are composed into a single WebAssembly binary via WebAssembly Composition (WAC) [2], which can be instantiated on CPUs, GPUs, or FPGAs using appropriate WebAssembly runtimes, providing a uniform and portable execution environment across heterogeneous hardware.

3 Experimental evaluation

We evaluate two core components already implemented: (1) kernel identification via cosine similarity on the DFG, and (2) the XGBoost-based cost prediction model. Experiments use CNN workloads (convolution, ReLU, batch normalization) analyzed from LLVM/MLIR IR across heterogeneous CPU targets.

Kernel identification: Starting from source code, we generate LLVM/MLIR IR and apply our MLIR pass to construct the DFG. Candidate kernels are extracted and mapped to vector representations. Cosine similarity against a library of pre-characterized kernel vectors identifies the type (e.g., convolution) with inference times under 1ms. Unlike ML graph-based alternatives [3], our approach requires no training and achieves equivalent accuracy.

Cost prediction: Identified kernels feed into a pre-trained XGBoost model taking software features (e.g., convolution input/output sizes, iteration counts) normalized by hardware characteristics to estimate memory/CPU utilization rates and predict execution time. Tuned model performance has a MAE of 0.000508, R^2 0.98, and MAPE 1.53%, demonstrating high accuracy for per-kernel time prediction across hardware targets.

SLA filtering, metrics-driven orchestration, and WebAssembly component model execution remain work-in-progress. Preliminary Wasm deployment tests confirm the component model supports the required modularity (monitoring + control + kernel) with negligible startup overhead. Prior decomposition work demonstrated 50%+ cache utilization improvements [6], which we expect to extend across heterogeneous targets with the full pipeline.

References

[1] [n. d.]. Intel Unveils Infrastructure Processing Unit — intel.com. https://www.intel.com/content/www/us/en/newsroom/news/infrastructure-processing-unit-data-center.html. [Accessed 31-07-2024].

[2] ByteCodeAlliance. 2026. The WebAssembly Component Model. https://component-model.bytecodealliance.org/ Accessed: 2025-03-27.

[3] Chris Cummins, Zacharias V. Fisches, Tal Ben-Nun, Torsten Hoefler, Michael F P O'Boyle, and Hugh Leather. 2021. ProGraML: A Graph-based Program Representation for Data Flow Analysis and Compiler Optimizations. In *Proceedings of the 38th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 2244–2253. https://proceedings.mlr.press/v139/cummins21a.html

[4] Lauri Loven, Reza Farahani, Ilir Murturi, Stephan Sigg, and Schahram Dustdar. 2026. Agentic Edge Intelligence: A Research Agenda. In *Proceedings of the 18th IEEE/ACM International Conference on Utility and Cloud Computing (UCC '25)*. Association for Computing Machinery, New York, NY, USA, Article 66, 5 pages. doi:10.1145/3773274.3777421

[5] Yizhou Shan, Will Lin, Zhiyuan Guo, and Yiying Zhang. 2022. Towards a fully disaggregated and programmable data center. In *Proceedings of the 13th ACM SIGOPS Asia-Pacific Workshop on Systems* (Virtual Event, Singapore) *(APSys '22)*. Association for Computing Machinery, New York, NY, USA, 18–28. doi:10.1145/3546591.3547527

[6] Guilherme Mendes Vieira de Matos, Washington Rodrigo Dias da Silva, Fábio Luciano Verdi, and Andrew Williams. 2025. DECOMPOSER: Functional decomposition and distributed execution of monolithic applications to heterogeneous resources. In *NOMS 2025-2025 IEEE Network Operations and Management Symposium*. 1–10. doi:10.1109/NOMS57970.2025.11073700