Towards AI-Augmented Co-Compilation for Smart IoT Systems Through Domain-Specific LLMs

Federico Ciccozzi*
*Mälardalen University, Sweden
Email: federico.ciccozzi@mdu.se

***Abstract*—The explosion of smart, interconnected IoT devices is driving demand for higher performance, real-time processing, and self-* capabilities (self-adaptation, self-reconfiguration, self-healing) across heterogeneous platforms. CPUs, GPUs, and FPGAs now provide exceptional compute power but greatly complicate software toolchains. We envision *AI-COMPILER*, an LLM-driven, domain-specific framework that automates co-compilation and optimization of heterogeneous applications in smart cities, smart transportation, and real-time analytics. Tackling multi-objective goals—power efficiency, reliability, and dynamic reconfiguration—AI-COMPILER aims to transform how IoT and edge infrastructures are developed, deployed, and maintained. We examine critical issues such as generalization to new hardware, human-AI collaborative tuning, and semantic coherence across multiple DSLs, arguing that AI-COMPILER can usher in more agile, resilient IoT ecosystems through flexible, context-aware code generation and adaptation.**

***Index Terms*—Smart IoT, compiler optimization, heterogeneous computing, domain-specific languages, self-adaptation, large language models, artificial intelligence, MLIR**

I. Introduction

Heterogeneous computing platforms, ranging from CPUs and GPUs to FPGAs and specialized accelerators, are increasingly critical for emerging smart IoT applications. As cities, transportation systems, and industrial processes become more interconnected and data-driven, the computational demands for real-time analytics and decision making continue to rise [1]. Beyond raw performance, modern IoT applications must also exhibit self-adaptation (adapting to changing environmental and operational conditions), self-reconfiguration (dynamically deploying and adjusting tasks across diverse hardware), and self-healing (automatically recovering from performance bottlenecks or faults).

Traditional compilation flows for CPUs, GPUs, and FPGAs often operate in isolation, requiring domain experts to apply optimizations manually or use single-target compiler heuristics. This siloed approach poses challenges in ensuring seamless, continuous operation within resource- and power-constrained devices [2].

We propose *AI-COMPILER*, a co-compilation framework driven by a domain-specific Large Language Model (LLM) that automatically optimizes code generation for heterogeneous hardware platforms used in smart Internet-of-Things (IoT) systems. Unlike general-purpose LLMs, this specialized model is trained on diverse sets of IoT-related codebases, performance metrics, and hardware specifications. **AI-COMPILER** aims to:

1) **Integrate self-* capabilities:** leverage automated adaptation, reconfiguration, and healing to improve the resilience and responsiveness of smart applications.
2) **Reduce human effort:** offload labor-intensive tuning tasks for each hardware type, accelerating development and deployment cycles.
3) **Enable multi-objective optimization:** balance performance, power consumption, latency, and reliability in IoT systems.
4) **Maintain semantic fidelity across DSLs:** ensure that cross-platform transformations preserve functional correctness and domain-specific constraints.

II. Background and Motivation

IoT applications in smart cities, intelligent transportation, and industrial automation often employ a range of heterogeneous devices. Sensors, edge nodes, and cloud servers must interact seamlessly while guaranteeing real-time data processing, minimal energy usage, and adaptive fault tolerance. Achieving these objectives at scale requires sophisticated compilation and scheduling strategies [3].

A. Heterogeneity in IoT and Need for Self-*

IoT nodes may incorporate specialized accelerators (e.g., GPUs, TPUs, or AI ASICs) or reconfigurable logic (FPGAs) for on-device intelligence. Meanwhile, advanced CPUs handle control logic or system orchestration. Managing these varied architectures typically calls for a mix of DSLs (e.g., TensorFlow, OpenCL, CUDA, or high-level synthesis for FPGAs) and diverse toolchains.

Self-adaptation involves monitoring system metrics and environmental conditions, then adjusting code execution strategies (e.g., changing the parallelization level or offloading computations to a GPU under peak load). Self-reconfiguration may entail partial reprogramming of an FPGA, whereas self-healing detects performance bottlenecks or anomalous conditions and autonomously triggers remedial compilation actions [4].

Manually integrating these self-* capabilities into heterogeneous systems is challenging. Autonomic management frameworks exist, but they often lack tight integration with compiler-level resource allocation and optimization. A domain-specific LLM could unify these concerns by learning from extensive real-world examples and continuously refining its compilation strategies.

B. Motivation for Domain-Specific LLM and Co-Compilation

While general-purpose LLMs have demonstrated remarkable progress in code generation and interpretation, they frequently lack deep awareness of complex IoT workloads, their real-time constraints, and the intricacies of heterogeneous hardware. In many cases, optimization for resource-constrained edge devices or adaptive real-time scheduling requires specialized domain knowledge that is absent from broad LLM training corpora. Hence, harnessing a *domain-specific LLM* becomes essential. By embedding hardware-specific insights and IoT-driven constraints into the language model, we can facilitate more effective co-compilation strategies—i.e., automatically partitioning workloads between CPUs, GPUs, and FPGAs, while respecting power budgets, timing deadlines, and fault-tolerance requirements. This domain-focused approach not only streamlines code generation but also ensures that higher-level goals (e.g., energy efficiency, reliability) are maintained from design time to runtime, promoting a cohesive end-to-end toolchain for next-generation IoT applications.

III. Related Work

Traditional approaches to optimizing IoT applications for heterogeneous platforms often rely on platform-specific toolchains and heuristics. For example, TVM [5] automates deep-learning workload optimization across CPUs and GPUs, while Halide [6] introduces a domain-specific language with scheduling constructs adaptable to embedded and heterogeneous targets. LLVM-based frameworks [7] have significantly advanced compiler analyses and transformations, and MLIR [8] extends this support to diverse hardware backends via specialized dialects. However, MLIR's FPGA and GPU extensions [9] often rely on static optimizations requiring expert-level configuration.

Machine learning has been leveraged to guide compiler optimizations (e.g., loop tiling, vectorization) [10], with recent approaches employing reinforcement learning and neural networks to automate pass selection [11]. These solutions typically target a single accelerator or domain and lack self-* capabilities critical in IoT contexts. LLMs like Codex [12] and AlphaCode [13] excel at code generation but are not inherently hardware-aware. Existing LLM-based optimizations rarely support adaptive runtime reconfiguration, a key requirement for edge devices. Meanwhile, IoT self-* frameworks commonly employ rule-based or policy-driven methods (e.g., MAPE-K loops [14]) that often rely on static heuristics.

Recent research in domain-specific modeling and compilation seeks to make heterogeneous hardware more accessible and systematic. Our prior work [15] bridges software engineering practices with platform constraints, and we have also addressed the precise compilation of domain-specific modeling languages [16]. Our proposed AI-COMPILER extends these ideas by integrating a specialized LLM that can flexibly adapt domain-specific models to heterogeneous IoT hardware and add runtime self-* capabilities absent in many existing solutions.

In sum, while MLIR-based compilation, ML-driven optimizations, and LLM-based code generation each address parts of the challenge, no existing approach fully unifies specialized LLM capabilities with runtime self-* adaptation for heterogeneous IoT hardware. AI-COMPILER aims to close this gap through a holistic framework that integrates these complementary techniques.

IV. AI-COMPILER Vision for IoT

AI-COMPILER envisions a layered architecture where a domain-specific LLM orchestrates the entire compilation pipeline and runtime adaptation of IoT applications across CPU, GPU, and FPGA platforms. Key components include:

1) **IoT-driven frontend:** interfaces with sensor data streams, control logic, and multiple DSLs used in edge analytics or control applications.
2) **Domain-specific LLM core:** ingests high-level specifications along with real-time feedback on performance, resource usage, and error metrics. This LLM proposes code transformations, partitioning strategies, and reconfiguration plans for the target hardware.
3) **Runtime adaptive layer:** continuously monitors system conditions (workload changes, partial device failures, etc.) and triggers LLM-driven self-* actions (e.g., re-synthesizing FPGA modules or migrating workloads to an underutilized GPU).

Self-Adaptation. Via real-time performance monitoring, AI-COMPILER can dynamically tune parameters (e.g., thread block sizes on GPUs), adjust parallelization on CPUs, or modify pipelining on FPGAs, ensuring optimal throughput under evolving conditions.

Self-Reconfiguration. As workloads fluctuate, AI-COMPILER could decide to offload certain tasks from a CPU to an FPGA if latency requirements tighten or power must be conserved. The LLM can swiftly generate the required hardware descriptions, possibly employing partial reconfiguration on FPGAs to minimize downtime.

Self-Healing. If performance anomalies emerge (e.g., a GPU kernel crashes or FPGA clock constraints are violated), the framework can isolate problematic code segments, fine-tune or regenerate them using the LLM, and redeploy without human intervention, significantly reducing service interruptions.

Smart IoT systems often operate under resource constraints and have safety-critical requirements. Therefore, AI-COMPILER's decisions should not only account for typical compiler optimizations but also integrate system-level and domain-level constraints (e.g., real-time deadlines, fail-safe states). Collaboration between the LLM, the IoT device's operating system, and higher-level orchestration (for cloud-edge coordination) is imperative.

V. Technical Architecture

A critical enabler of AI-COMPILER's capabilities is a modular, extensible architecture that integrates LLM-based

code generation with industry-standard compiler infrastructures such as MLIR (Multi-Level Intermediate Representation).

A. Frontend and Intermediate Representations

Multi-DSL Parsing. The *IoT-Driven Frontend* ingests code from various DSLs—TensorFlow, OpenCL, CUDA, or FPGA high-level synthesis (HLS)—and converts them into a common intermediate representation. MLIR provides a powerful infrastructure for modeling domain-specific dialects while maintaining a standardized set of compiler passes.

LLM-Oriented Embeddings. Once the code is parsed, it can be transformed into textual or token-based embeddings suitable for the domain-specific LLM. This embedding includes not only high-level constructs (e.g., loops, operators) but also metadata on constraints, such as target power budgets or timing requirements.

B. LLM-Driven Optimization Layer

Domain-Specific Transformation Proposals. The specialized LLM receives the embedded IR and system constraints (e.g., real-time deadlines, FPGA resource availability). Based on its training, it proposes code transformations, parallelization strategies, or offloading decisions. These proposals are output as either: **MLIR transform passes**, custom lowerings that can be directly injected into the MLIR pipeline, or **Refined IR modifications**, direct edits or rewrites of IR sections to optimize performance, memory usage, or energy efficiency.

Iterative Feedback Loop. After the MLIR pipeline applies the transformations, static analyses (e.g., resource usage estimates, scheduling feasibility) provide feedback to the LLM. This feedback loop enables the model to refine its proposals iteratively, adjusting design parameters (e.g., unrolling factors, scheduling heuristics) to meet multi-objective goals.

C. Back-End Compilation and Deployment

In the back-end, standard toolchains generate low-level code or bitstreams for each target platform.

CPU/GPU compilation. For GPUs, the framework can invoke CUDA or OpenCL back-ends to produce optimized kernels. For CPUs, it leverages traditional LLVM passes, enhanced by any LLM-derived transformations (e.g., vectorization, memory layout optimizations).

FPGA toolchain integration. MLIR's HLS or specialized FPGA dialect can be converted to a vendor-specific format (e.g., Xilinx HLS). The LLM can guide partial reconfiguration strategies by selecting which modules to swap out at runtime under specific conditions.

Finally, the compiled artifacts (CPU binaries, GPU kernels, FPGA bitstreams) are packaged with metadata that enables runtime self-* actions. For instance, the system can dynamically load a new FPGA bitstream if the LLM indicates that a reconfiguration could yield better power or latency profiles.

D. Runtime Monitoring and Adaptation

The *Runtime Adaptive Layer* continuously monitors performance counters, resource usage, and fault events. These metrics are fed back to the LLM, either directly or through a lightweight aggregator. In scenarios where an IoT device experiences partial failures (e.g., an overheating GPU), the LLM can initiate a re-compilation or reconfiguration sequence on the fly, guided by MLIR-level transformations that preserve semantics and real-time constraints.

VI. Envisioned Benefits of AI-COMPILER

While general-purpose LLMs demonstrate remarkable capabilities in understanding and generating code, their lack of specialized knowledge hampers performance-critical tasks for IoT devices. A domain-specific LLM offers several advantages:

To effectively co-compile for CPUs, GPUs, and FPGAs in IoT scenarios, the LLM is fine-tuned on:

- IoT-specific code patterns (e.g., low-power optimization, secure sensor reading).
- Real-time data processing paradigms (e.g., streaming analytics).
- Hardware design flows typical of FPGAs used in edge devices (including partial reconfiguration techniques).
- Power management profiles, concurrency models, and scheduling heuristics.

The model can learn best practices for dynamic workload distribution (e.g., scaling GPU usage during peak times, employing FPGA acceleration for cryptographic tasks) and automated parallelization strategies tuned for edge constraints (bandwidth, energy, thermal limits).

A specialized LLM-driven co-compilation framework offers transformative benefits for next-generation IoT systems:

Performance and Latency. By automating hardware-specific optimizations, AI-COMPILER can achieve near-expert-level performance across heterogeneous resources. This is critical in applications like smart transportation or industrial process control, where milliseconds matter.

Energy Efficiency and Sustainability. IoT devices are often energy-constrained. AI-COMPILER can automatically select low-power operating modes, balance workloads across underutilized accelerators, or throttle unnecessary computations to meet real-time constraints while conserving energy.

Resilience and Fault Tolerance. Smart systems are prone to environmental changes and sporadic faults. AI-COMPILER's self-healing mechanisms lower system downtime by proactively detecting and addressing performance regressions. FPGA partial reconfiguration, for instance, can be guided by the LLM to restore failing logic blocks.

Faster Development Cycles. Developers no longer need to hardware experts. By providing high-level directives (e.g., specifying target latency bounds or power budgets), they can rely on the AI-COMPILER to generate appropriately optimized code. This rapid iteration is valuable in prototyping new IoT services for smart cities or pilot-scale industrial systems.

High-Level Semantic Preservation. Translating between different DSLs (e.g., from TensorFlow to a low-level FPGA

pipeline description) can risk semantic mismatches that degrade correctness or performance. The LLM ensures semantic coherence by preserving essential domain semantics, including concurrency and real-time constraints.

VII. Discussion and Future Directions

Live Data Feedback Loops. Edge devices continuously gather real-world data. Integrating this data with AI-COMPILER could enable more fine-grained, context-aware adaptation. For instance, an LLM might adjust FPGA configurations based on traffic load forecasts in smart transport corridors or environmental sensors in industrial plants.

Security and Privacy. AI-driven code generation must consider secure programming practices, especially in safety- and mission-critical IoT applications. Future research could explore fine-tuning on security-sensitive datasets to proactively identify and mitigate vulnerabilities before deployment.

Scalability to Cloud-Edge Collaboration. Although this paper focuses on device-level compilation, a natural extension involves distributing workloads across edge and cloud resources. AI-COMPILER could orchestrate multi-tier deployments, ensuring that tasks run at optimal locations based on latency, cost, and energy constraints. In large-scale environments like smart cities, balancing real-time data processing at the edge with heavier analytics in the cloud could further improve system responsiveness and resource efficiency.

Integration with Domain-Specific Modelling Languages. Building on prior work on accessible software engineering for heterogeneous hardware [15] and systematic compilation of domain-specific modeling languages [16], deeper integration with high-level modeling tools remains an open challenge. Future research should investigate how AI-COMPILER can interpret advanced domain models (e.g., safety-critical UML/MARTE profiles or advanced SysML diagrams), automatically translating them into optimized low-level representations with minimal user intervention.

Beyond Single Accelerator Types. While CPUs, GPUs, and FPGAs are primary targets, emerging accelerators (e.g., AI ASICs, neuromorphic chips) may be part of future IoT ecosystems. The domain-specific LLM architecture can be extended to these new platforms by incorporating specialized training data and optimization heuristics that address architecture-specific parallelism or novel computation paradigms.

Latency. Offline compilation latency increases with LLM-guided exploration of the transformation space. Mitigation tactics include (i) edge–cloud partitioning of heavy searches, (ii) caching and reuse of earlier transformations, and (iii) speculative compilation using lightweight surrogate models during tight control cycles.

Human-in-the-Loop Customization. Although the goal is to reduce the burden on human experts, certain design decisions or safety constraints might require domain-specific knowledge that an LLM cannot fully automate. Future iterations of AI-COMPILER should provide intuitive interfaces for developers to guide or override the system, ensuring that critical domain constraints are respected without undermining the benefits from automated optimization.

VIII. Conclusion

Heterogeneous computing architectures hold great potential for advancing the performance, adaptability, and resilience of smart IoT systems. However, fully exploiting these platforms' capabilities demands sophisticated, automated approaches to compilation and runtime management. We introduced AI-COMPILER, a domain-specific LLM-driven vision that aims to integrate self-adaptation, self-reconfiguration, and self-healing into the co-compilation process. By leveraging real-time system feedback and specialized hardware knowledge, AI-COMPILER can revolutionize how we build and maintain smart IoT applications, from smart cities and autonomous vehicles to industrial automation.

The success of this approach will rely on continued research into domain-specific LLM architectures, multi-objective optimization, robust feedback loops, and seamless integration with modeling tools. By converging compiler technology, machine learning, and IoT system design, we can realize a new era of agile, autonomous, and resource-efficient computing infrastructures for the dynamic needs of future smart environments.

Acknowledgment. This work was partially funded by the VR ORPHEUS project (rn. 2022-03408).

References

[1] W. Shi *et al.*, "Edge Computing: Vision and Challenges," *IEEE Internet of Things Journal*, 2016.
[2] A. Romero *et al.*, "Efficient Deployment of Deep Learning Models in IoT Devices: A Survey," *IEEE Internet of Things Journal*, 2020.
[3] R. Mayer *et al.*, "Smart Cities: State-of-the-Art and Future Challenges," *IEEE Access*, 2019.
[4] R. Bi *et al.*, "IoT Traffic Anomaly Detection via Reinforcement Learning," *IEEE Transactions on Network and Service Management*, 2019.
[5] T. Chen *et al.*, "TVM: An Automated End-to-End Optimizing Compiler for Deep Learning," *OSDI*, 2018.
[6] J. Ragan-Kelley *et al.*, "Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines," *ACM SIGPLAN Notices*, vol. 48, no. 6, 2013.
[7] C. Lattner and V. Adve, "LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation," in *Proceedings of the International Symposium on Code Generation and Optimization*, 2004.
[8] C. Lattner *et al.*, "MLIR: A Compiler Infrastructure for the End of Moore's Law," *IEEE Micro*, vol. 41, no. 4, 2021.
[9] N. Vasilache *et al.*, "Towards Fast and Efficient FPGA Programming with MLIR," *arXiv preprint arXiv:2008.08229*, 2020.
[10] A. H. Ashouri *et al.*, "A Survey on Compiler Autotuning Using Machine Learning," *ACM Computing Surveys*, vol. 51, no. 5, 2018.
[11] C. Cummins *et al.*, "End-to-end Deep Learning of Optimization Heuristics," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2017.
[12] M. Chen *et al.*, "Evaluating Large Language Models Trained on Code," *arXiv preprint arXiv:2107.03374*, 2021.
[13] Y. Li *et al.*, "Competition-Level Code Generation with AlphaCode," *Science*, vol. 378, no. 6624, 2022.
[14] M. Salehie and L. Tahvildari, "Self-Adaptive Software: Landscape and Research Challenges," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 4, no. 2, 2009.
[15] F. Ciccozzi, "Towards accessible software engineering for heterogeneous hardware," *Procs of ACDSA 2024*. Institute of Electrical and Electronics Engineers Inc., 2024.
[16] F. Ciccozzi, "Towards Systematic and Precise Compilation of Domain-Specific Modelling Languages," *Procs of ITNG*. Cham: Springer Nature Switzerland, 2024.