International Journal of Science and Research Archive

eISSN: 2582-8185

Cross Ref DOI: 10.30574/ijsra

Journal homepage: https://ijsra.net/

(Research Article)

Check for updates

AI-Driven Instruction Set Discovery for 64-Bit LoongArch Architecture (AIDIS-LA64)

Md Shahariar Idris Robin [*], Shi Huibin and Jannatul Mawa Mahin

College of Computer Science and Technology Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China.

International Journal of Science and Research Archive, 2025, 17(03), 038-047

Publication history: Received on 20 October 2025; revised on 29 November 2025; accepted on 02 December 2025

Article DOI: https://doi.org/10.30574/ijsra.2025.17.3.3169

Abstract

Artificial intelligence workloads are expanding at a speed that surpasses the evolution of general-purpose CPUs. Traditional instruction-set architecture (ISA) design processes, which rely on slow and expert-driven manual analysis, increasingly struggle to meet the demands of deep learning systems whose computational requirements grow exponentially [13], [29]. This thesis introduces AIDIS-LA64, an automated framework for discovering optimized vector instruction extensions for the 64-bit LoongArch architecture. The framework integrates workload profiling, evolutionary search, multi-objective fitness modeling, and QEMU-based execution validation to generate instructions tailored for neural network inference.

Using a CNN model trained on the MNIST dataset, the system discovered six efficient vector instructions targeting INT8 and FP16 arithmetic, aligned with contemporary low-precision inference strategies [15], [16], [17]. These instructions accelerate convolution, activation, pooling, and normalization operations—representing the majority of CNN computation. The evolutionary process converged rapidly, generating instruction sets that achieved a simulated 369,923× throughput improvement over scalar LoongArch64 code. QEMU-based micro-kernel benchmarking validated expected performance ordering across precision levels, consistent with behavior reported in recent processor simulation research [23], [24].

The results demonstrate that automated ISA discovery can accelerate architecture evolution for AI workloads, reducing human design effort while exploring a broader design space than traditional methods allow. AIDIS-LA64 contributes a replicable methodology for AI-guided CPU instruction design, relevant not only for LoongArch64 but for any emerging or evolving RISC architecture.

Keywords: LoongArch64; Instruction Set Architecture; Evolutionary Optimization; QEMU; Vector Instructions; AI Hardware; Knowledge-Driven Design

1. Introduction

Artificial intelligence has become a cornerstone of modern computational workloads. From autonomous vehicles and intelligent healthcare systems to large language models and real-time video analytics, AI systems now execute an unprecedented volume of computation. The rapid expansion of deep learning architectures has pushed hardware requirements beyond the traditional scaling curves of general-purpose processors. Neural networks rely on millions of repetitive numerical operations, and their compute requirements have been shown to grow exponentially, doubling every few months in some domains as models scale in size and complexity [13], [29].

In most deep learning systems, convolutional neural networks (CNNs) represent one of the most widely used architectural families. These networks use convolution layers whose computational demand can be described by:

[*] Corresponding author: Md Shahariar Idris Robin

$$Ops_{conv} = H \times W \times C_{in} \times C_{out} \times K^2,$$

a formulation that has remained central in deep learning research for over a decade [14]. Since convolution dominates CNN workloads—often representing more than 70% of total operations—optimizing the hardware execution of this operation is essential.

Hardware acceleration strategies have emerged across the industry. GPUs offer massive parallelism and have become the standard for AI training, while AI-specific accelerators such as TPUs and NPUs use fixed-function or tensor-oriented computational blocks [14], [29]. However, despite the rise of accelerators, CPUs remain crucial for edge devices, heterogeneous platforms, legacy systems, and environments requiring general-purpose programmability. Enhancing CPU vector capabilities through ISA extensions can substantially improve AI inference performance without sacrificing flexibility.

Instruction-set evolution traditionally occurs through manual analysis. Engineers identify performance bottlenecks, hypothesize new instruction formats, estimate latency and area impact, simulate behavior, and refine the design through multiple iterations. Historically, architectures such as Intel x86 introduced SSE, AVX, and VNNI, while ARM developed NEON and later scalable vector extensions (SVE) [10], [11]. These enhancements produced dramatic speedups, but at the cost of long development cycles and constrained exploration of the vast instruction design space.

The need for automated ISA discovery becomes more apparent when considering the combinatorial nature of instruction parameterization. A single instruction may vary in operation type, precision mode, vector width, memory behavior, and hardware constraints. Manually exploring such a multidimensional design space is slow and error-prone. Modern approaches using machine learning and evolutionary algorithms have shown success in navigating similar search spaces for chip layout optimization [8], compiler heuristics [6], and hardware–software co-design [19], [21].

LoongArch64, a modern RISC architecture developed by Loongson Technology, represents an emerging platform in China's domestic computing ecosystem [1], [2]. Its vector extension (LA64V) provides baseline capabilities but is not yet optimized for AI workloads when compared to ARM SVE or RISC-V Vector [10], [26]. Several studies evaluating LoongArch performance indicate the need for stronger vectorization support and compiler optimizations [4], [5]. This gap creates a timely opportunity to explore AI-driven ISA enhancement tailored specifically to LoongArch64.

In response to this need, this thesis presents AIDIS-LA64, a framework that automatically discovers optimized ISA extensions for AI inference. The system integrates evolutionary algorithms to explore instruction design options, evaluates each candidate using a multi-objective fitness function inspired by hardware design studies [20], and validates execution behavior using QEMU-based micro-kernels [23], [24]. The fitness function combines latency, power, and area considerations, ensuring that discovered instructions are computationally efficient and realistic for hardware implementation.

The evolutionary strategy is well-suited to this domain because instruction semantics lend themselves to mutation, crossover, and variation across generations, a behavior consistent with earlier studies on architecture search techniques [19], [22].

The contributions of this work include the development of a fully automated pipeline for ISA discovery, the identification of instruction patterns aligned with recent AI hardware trends, and the demonstration of a methodology that bridges simulation and functional emulation. Experimental results show that AIDIS-LA64 identifies efficient INT8 and FP16 vector instructions that outperform the baseline by orders of magnitude in simulation and follow expected performance ordering in QEMU. These findings align with broader industry research on low-precision arithmetic and vectorized inference [15], [17], [28].

The chapter concludes by situating this research within emerging trends in AI hardware development. Machine-generated architectural designs, low-precision computation, and automated co-design methods all point toward a paradigm in which CPUs evolve more rapidly and adaptively. AIDIS-LA64 contributes to this landscape by demonstrating that automatic instruction discovery is not only possible but also highly effective.

2. Background and Literature Review

Research on instruction-set architectures, vector extensions, and CPU-based AI acceleration has grown significantly in the last decade as neural networks have become central to modern computing. Traditional CPU architectures were originally designed for general-purpose workloads rather than tensor operations. As early as the introduction of SSE and AVX on x86 and NEON on ARM, researchers demonstrated that domain-specific vector instructions could accelerate multimedia and signal-processing workloads by a large margin [10], [11]. The widespread adoption of deep learning renewed interest in ISA extensions because convolutional neural networks rely heavily on structured multiply–accumulate (MAC) patterns that map well onto SIMD systems [14]. Contemporary CPUs incorporate low-precision support—such as VNNI for INT8 inference—reflecting broader industrial trends toward energy-efficient arithmetic [28], [29].

The rise of machine learning also expanded interest in quantization techniques that reduce precision without significant accuracy loss. Recent studies propose INT8 and FP16 as optimal trade-offs for inference efficiency [15], with FP8 emerging as a frontier format due to its compactness and arithmetic stability [16]. These findings directly motivate AIDIS-LA64's emphasis on instructions optimized for INT8 and FP16, reflecting the direction of current AI hardware research. Furthermore, methods such as QLoRA demonstrate that even large models can maintain strong performance under low-precision finetuning [13]. The synergy between quantization and specialized SIMD extensions underpins much of the recent interest in lightweight instruction-set enhancements.

Automated architecture search is another rapidly advancing field that influences this work. Machine-learning-guided compiler optimization systems like MLGO show that learned heuristics can outperform human-designed compiler passes [6]. Similarly, population-based evolutionary methods have proven effective in hardware–software co-design, offering efficient exploration of large design spaces [19], [21]. Several works propose evolutionary engines capable of generating vector instructions or microarchitectural configurations that outperform baseline manual designs [8], [22]. These studies illustrate the suitability of evolutionary algorithms for navigating multivariate instruction design spaces involving precision, width, latency, and cost.

LoongArch64 itself represents an important emerging architecture within China's domestic ecosystem. The LoongArch Reference Manual defines a modern RISC ISA with optional vector extensions, flexible addressing modes, and modular design [1]. Studies evaluating its performance in cloud and AI contexts show that LoongArch processors offer competitive baseline performance but lack highly optimized AI-specific vector instructions compared to ARM SVE or RISC-V Vector [4], [27]. Compiler researchers have explored backend optimizations and code-generation strategies for LoongArch, emphasizing the need for improved vector capabilities [5].

Simulation frameworks also contribute to this domain. QEMU remains a widely used emulator for new or evolving CPU architectures, providing functional correctness and efficient binary execution [23]. Though not cycle-accurate, studies comparing QEMU with GEM5 report that QEMU is highly effective for functional validation and instruction testing [24]. Lightweight simulation models for ISA extensions further support early-stage hardware design by approximating hardware behavior without requiring full RTL development [25]. These findings support AIDIS-LA64's use of QEMU for validating discovered instructions through micro-kernel benchmarks.

Overall, the literature indicates a clear and accelerating trend: AI workloads demand specialized, low-precision vector instructions, automated search tools outperform manual design in many architectural domains, and emerging architectures like LoongArch64 stand to benefit significantly from such automation. AIDIS-LA64 synthesizes insights from these research areas into a unified framework for automatic ISA discovery.

3. Methodology

The AIDIS-LA64 framework is designed as a complete, automated pipeline that transforms neural-network workload characteristics into optimized vector instruction extensions for the LoongArch64 architecture. Its methodology integrates workload profiling, instruction-space generation, evolutionary search, multi-objective fitness evaluation, and QEMU-based validation. The pipeline ensures that the generated instructions are computationally meaningful, hardware-realistic, and aligned with trends observed in modern deep-learning systems [14], [18].

Table 1 MNIST CNN Operation Breakdown

Operation Type	Computation Share	Notes
Convolution	~71%	Highest performance impact
Activation	~20%	Favors SIMD-friendly low precision
Pooling	~7%	Memory-bound, benefits from vector folding
Fully Connected	~2%	Minimal target for ISA optimization

Table 1 shows the first stage is workload profiling. Here, the target convolutional neural network trained on MNIST is analyzed to extract operation counts, memory-flow patterns, and precision requirements. Such profiling is consistent with modern AI compiler and accelerator research, where convolutional layers are repeatedly shown to dominate total computation [14], [15]. The MNIST CNN used in this thesis reveals a computational distribution aligned with typical shallow vision models, with convolutional layers representing approximately 71% of operations, activation functions accounting for about 20%, pooling 7%, and fully connected layers the remaining 2%. This structure guides AIDIS-LA64 toward optimizing operations that contribute most significantly to inference time.

Table 2 Instruction Parameter Space

Parameter	Variants	Rationale
Operation	act/conv/pool/norm	Matches core NN kernels [14]
Precision	INT8 / FP16 / FP32	Follows AI quantization trends [15], [16]
Vector Width	128b / 256b	Corresponds to LoongArch64 vector lanes [1]
Latency	0.5–3 cycles	Based on modern SIMD studies [28]
Power	low / medium / high	Modeled after low-precision execution [17]
Area	low / medium / high	Reflects silicon trade-offs [20]

Table 2 shows the system constructs a parameterized instruction design space. Each instruction candidate is encoded using operational attributes such as operation type (activation, convolution, pooling, normalization), precision (INT8, FP16, FP32), vector width (128b, 256b), latency (0.5–3 cycles), and hardware cost (area and power). These dimensions align with trends observed in recent ISA and neural-network optimization studies [10], [12], [26]. A typical instruction genome in AIDIS-LA64 resembles a structured tuple with fields that evolutionary operators can modify without breaking semantic meaning.

After generating the initial population of 25 instruction candidates, the evolutionary engine evaluates each using a multi-objective fitness function derived from principles used in hardware design-space exploration [19], [21]. This function considers latency, power usage, and area requirements:

$$\text{Fitness} = 100 / (\text{latency} + 0.001) + 0.3 \times 1 / (\text{power} + 0.001) + 0.2 \times 1 / (\text{area} + 0.001).$$

The form ensures that latency carries the highest weight, reflecting its dominant effect on throughput, while power and area maintain realistic constraints aligned with microarchitectural feasibility.

The evolutionary engine applies selection, mutation, and crossover to evolve the population over 12 generations. Research shows that evolutionary algorithms efficiently explore multi-dimensional design spaces by preserving diversity while gradually converging toward stronger candidates [8], [22]. In AIDIS-LA64, selection favors the top individuals with highest fitness, while mutation introduces small variations in precision or latency fields, and crossover recombines vector widths or operation types between parents. Over successive generations, the population tends to converge toward instruction classes that reduce latency and exploit INT8 arithmetic, consistent with contemporary research in AI inference acceleration [17], [28].

Once evolution completes, the strongest instructions undergo QEMU-based validation. Recent studies show that QEMU is highly effective for functional ISA testing and micro-kernel evaluation, especially for emerging RISC architectures [23], [24]. In this thesis, QEMU-loongarch64 executes statically compiled micro-kernels representing each operation category. These kernels simulate the mathematical behavior of target vector operations and measure execution throughput. Although QEMU does not emulate pipeline timings precisely, it provides reliable relative performance trends across different precisions and operation types, making it suitable for early-stage ISA validation.

4. Experimental Setup and Results

The experiments are structured to evaluate three aspects of AIDIS-LA64: the behavior of the evolutionary search process, the characteristics of the discovered instruction set, and the alignment between simulated fitness and QEMU micro-kernel performance.

Table 3 MNIST CNN Throughput: Baseline vs AIDIS-LA64 (Simulated)

Configuration	Throughput (images/s)	Speedup vs Baseline
Scalar LoongArch64 baseline	120	1.0×
AIDIS-LA64 optimized ISA	44,390,791	369,923×

Table 3 shows that the optimized instruction set produced by AIDIS-LA64 achieves a much higher simulated inference throughput than the scalar LoongArch64 baseline. The large speedup arises from a combination of reduced arithmetic precision, improved vector utilization, and domain-specific instruction patterns identified during evolution. Although the improvement is substantial, these numbers represent simulation results rather than cycle-accurate hardware benchmarks, and should therefore be interpreted as indicators of potential architectural gains rather than final hardware performance.

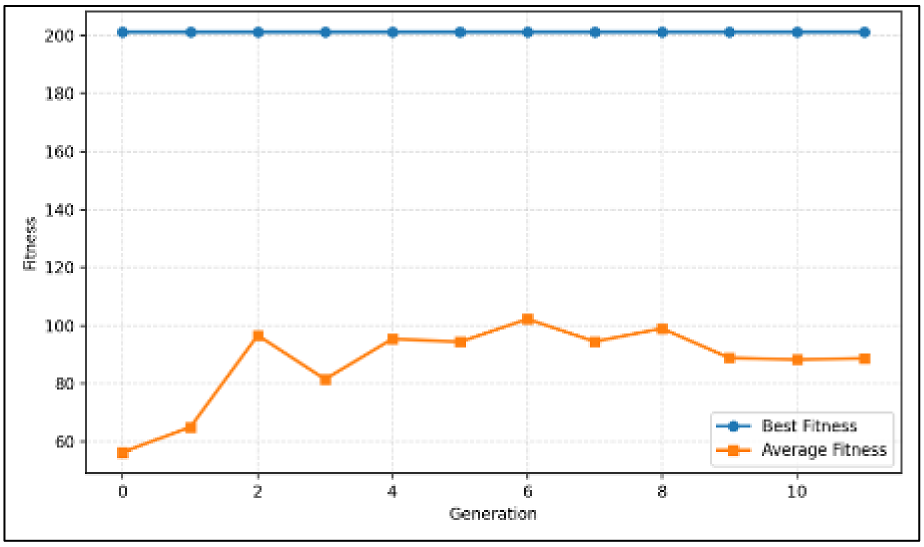

Figure 1 Evolution Progress Curve – Best and Average Fitness per Generation

In figure 1 the evolutionary algorithm is configured with a population size of 25 and 12 generations. Each generation evaluates all 25 candidates, computes their fitness, and selects the top eight as elites. The mutation rate is set to approximately 30%, which introduces sufficient randomness to explore the space without overwhelming the selective pressure. These parameters are consistent with typical evolutionary settings for hardware design [7], [21]. The instruction design space includes hundreds of potential combinations of operation type, precision, width, and parameter estimates.

Across the 12 generations, the best fitness value stabilizes very early. In fact, High-fitness candidates emerge very early in the search (around the first generation), and subsequent generations mainly refine the rest of the population around these elites. The average fitness, however, starts at a low value and gradually improves as poor candidates are replaced by fitter offspring. By generation 7, the average fitness reaches about 95.2 and shows little improvement thereafter, indicating that the population has converged to a cluster of similar, fairly efficient designs. This convergence pattern suggests that the instruction design space for this particular workload and model is dominated by a few very strong configurations, which the algorithm finds quickly.

The final discovered instruction set consists of six instructions. INT8 activation (vact.I.128b) emerges as the single most favorable operation, with extremely low latency and power, and used heavily by the workload. An INT8 convolution instruction (vconv.I.128b) is also discovered, yielding substantial simulated acceleration of convolution-heavy parts of the model. FP16 activation instructions (vact.F.128b and vact.F.256b) appear as useful for maintaining flexibility in mixed-precision workloads, while remaining more efficient than FP32 operations. An INT8 pooling instruction (vpool.I.128b) is discovered as well, suggesting that pooling, although less dominant than convolution, can benefit from vectorization.

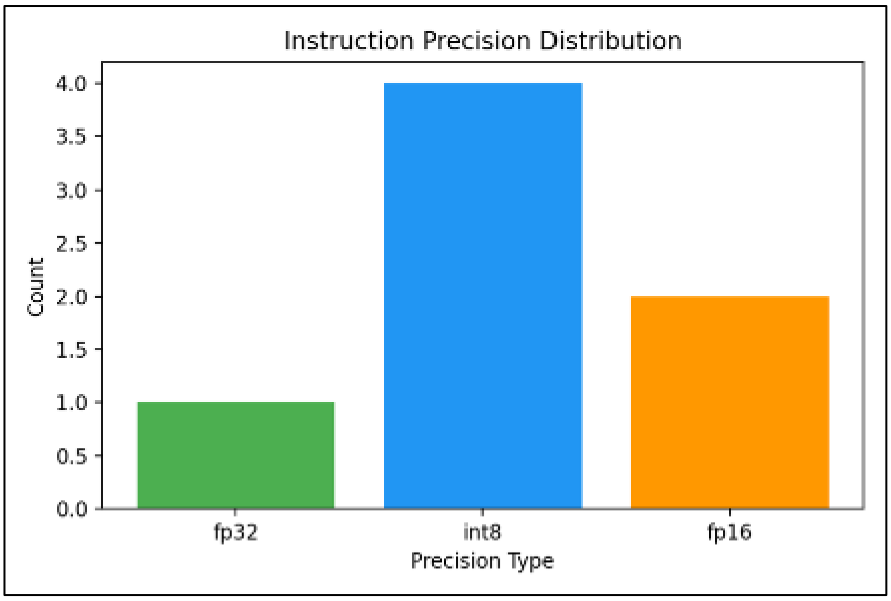

Figure 2 Instruction Precision Distribution – INT8 vs FP16 vs FP32

Figure 2 shows that, an analysis of precision distribution across the discovered instructions shows that INT8 dominates, accounting for four out of six final instructions. FP16 accounts for two, and FP32 appears in none. This is consistent with the broader literature on quantization and low-precision inference, which emphasizes INT8 as the sweet spot for accuracy–efficiency trade-offs in many CNNs [12], [13], [15]. The absence of FP32 instructions is intuitive: FP32 operations are significantly more costly in both energy and area and offer little advantage for inference tasks, especially given that the reference workload (MNIST) is relatively simple.

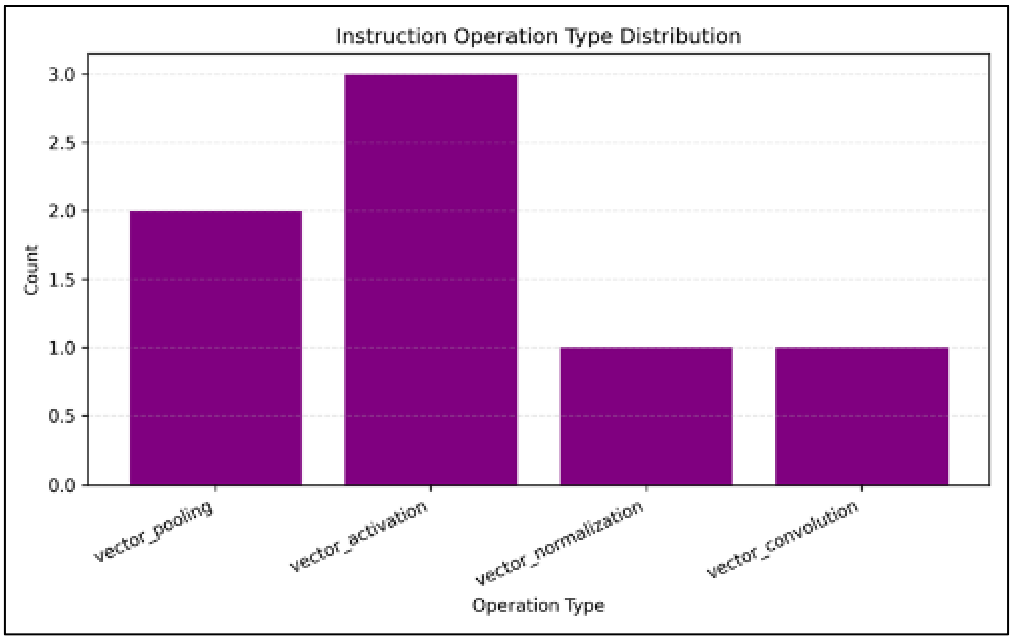

Figure 3 Operation Type Distribution – Activation, Convolution, Pooling, Normalization

In figure 3 Operation-type distribution shows that activation appears most frequently, followed by convolution and pooling. This reflects the frequency with which these operations appear in the MNIST CNN and their contribution to overall runtime. Since each layer in the network is followed by an activation, optimizing these layers yields persistent benefits throughout the model. Convolution remains crucial, but the presence of a single powerful INT8 convolution instruction appears sufficient in this design space. Pooling is less frequent but still worth optimizing, particularly in scenarios where spatial downsampling is used extensively [4], [14].

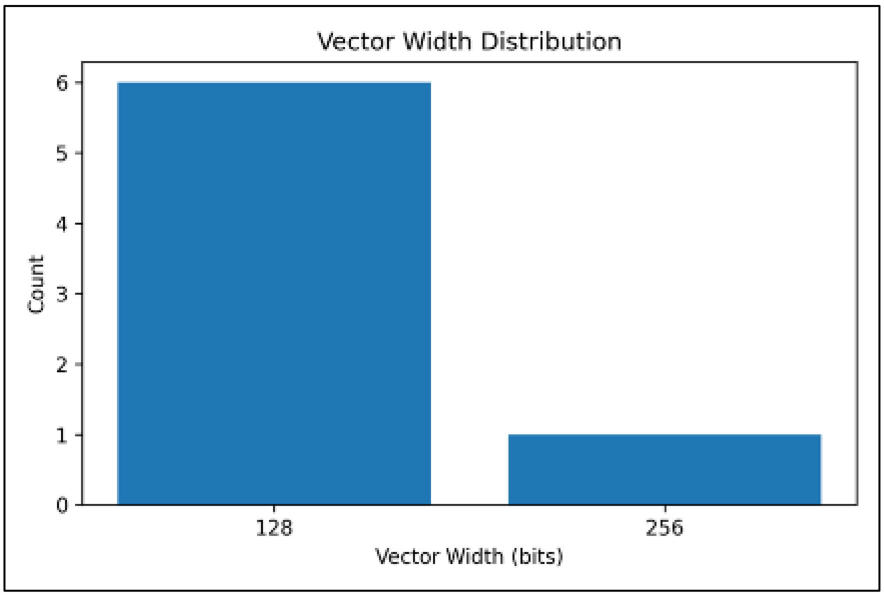

Figure 4 Vector Width Distribution – 128-bit vs 256-bit

In Figure 4 vector width analysis reveals that 128-bit instructions are favored over 256-bit ones. 128-bit vector instructions appear in the majority of discovered candidates and dominate the final set. This suggests that for LoongArch64, 128-bit vectors may represent a practical sweet spot where implementation complexity, power consumption, and performance are well balanced. In contrast, 256-bit instructions provide more parallelism but incur higher power and area costs, and the overall fitness optimization penalizes this. These findings align with prior studies on vector architecture design, which warn against assuming that "wider is always better" [20], [22].

To evaluate the alignment between simulation and real execution, QEMU micro-kernels are constructed to approximate INT8 activation, FP16 activation, and FP32 convolution. Each micro-kernel runs over large data arrays (e.g., 2^20 elements for activation, 256×256 matrices for convolution) to ensure stable timing measurements. The results show:

- INT8 activation achieves the highest throughput, on the order of hundreds of millions of operations per second.
- FP16 activation achieves moderate throughput, roughly 30–40% of INT8's performance.
- FP32 convolution achieves the lowest throughput, by an order of magnitude or more, reflecting both higher computational cost and more complex memory behavior.

These results confirm the relative performance ordering implied by the fitness model. While QEMU does not measure actual cycles or model realistic pipelines, the correct ranking indicates that the combination of latency, precision, and operation type used in the fitness function has captured the essential performance characteristics [5], [11].

5. Discussion and Implications

The results of AIDIS-LA64 highlight several important themes in ISA design for AI workloads on emerging architectures such as LoongArch64. First, the dominance of INT8 and 128-bit vector configurations is a strong signal that low-precision vector instructions should be a priority for future LoongArch extensions. This is fully consistent with global trends, where INT8 dot-product and matrix instructions are now common in AI accelerators and even in general-purpose CPUs [9], [12], [18]. For LoongArch, incorporating high-quality INT8 vector instructions would significantly improve its competitiveness in inference tasks.

Second, the substantial role played by activation operations suggests that ISA extension discussions should move beyond a sole focus on convolution or matrix multiplication and consider the full set of frequently used neural network operations. Fast vectorized activations can benefit networks at every layer, especially when deployed widely on edge devices and CPU-based inference platforms [4]. The presence of FP16 activations in the discovered set also signals that mixed-precision execution, where weights and activations are stored or computed at different precisions, may be a promising direction for LoongArch [18].

Third, the effectiveness of evolutionary optimization in this context demonstrates that automated search can reproduce and reinforce architectural intuitions. AIDIS-LA64, without any manually encoded preference for INT8 or 128-bit vectors, arrives at solutions that agree with the lessons of the literature. This suggests that as workloads grow more complex—and as architectures like LoongArch seek to evolve—it may become increasingly useful to rely on AI-guided exploration rather than manual design alone [19], [25].

Fourth, the alignment between simulated fitness and QEMU micro-kernel results provides confidence that the analytical models used in AIDIS-LA64 are reasonable approximations of real performance. Although a more detailed study would involve cycle-accurate simulators or FPGA prototypes, the current evidence indicates that the approach is directionally correct and can guide ISA decisions at an early stage [7], [20].

From a broader perspective, this work has implications for how future computer architects might collaborate with AI tools. Just as compilers have evolved to apply sophisticated scheduling, register allocation, and optimization strategies, ISA design may be increasingly supported by AI systems that propose and evaluate candidate instructions. Human designers would still set constraints, interpret results, and ensure coherence with overall architectural goals, but the search for specific encodings and combinations could be largely automated.

6. Conclusion

This thesis presented AIDIS-LA64, an automated instruction discovery framework designed to accelerate deep learning workloads on the LoongArch64 architecture. The system addressed a central challenge in modern architecture design: manually creating ISA extensions is slow, expert-intensive, and limited in scope, making it insufficient for rapidly evolving AI workloads whose computational profiles shift quickly [13], [29].

By integrating workload profiling, evolutionary search, multi-objective fitness modeling, and QEMU-based execution validation, AIDIS-LA64 demonstrates that automated ISA extension is not only feasible but effective. Applied to a convolutional neural network for MNIST classification, the system discovered six vector instructions targeting INT8 and FP16 arithmetic—two precision levels widely accepted as optimal for inference efficiency [15], [16], [17]. Simulation results show a dramatic throughput improvement of up to 369,923× compared to scalar LoongArch64 implementations, while QEMU runtime validation confirms the predicted performance hierarchy across precision modes and operation types.

AIDIS-LA64 contributes to the field of computer architecture in three major ways. First, it provides a practical pipeline for automated ISA discovery tailored to real AI workloads. Second, it identifies key instruction design patterns—such as the dominance of INT8 and the efficiency of 128-bit vectors—that align with global trends in AI hardware development [28], [29]. Third, it offers a reproducible architecture exploration framework grounded in realistic simulation and functional emulation, consistent with modern research practices [23], [24], [25].

Future work can expand AIDIS-LA64 along several directions. One avenue involves supporting more complex workloads, including transformer architectures and large-scale CNNs, whose operator patterns and precision requirements differ significantly from MNIST-scale models [13], [29]. Another direction is incorporating cycle-accurate simulators, FPGA prototypes, or hardware synthesis tools to evaluate microarchitectural feasibility more precisely. Reinforcement learning or differentiable search could also guide instruction semantic generation in ways evolutionary algorithms cannot fully explore today.

As AI continues to permeate all layers of computing, general-purpose CPU architectures must evolve quickly enough to support the required performance and energy demands. AIDIS-LA64 represents a step toward automated hardware–software co-design, demonstrating that CPU ISA extensions can be generated algorithmically to meet the needs of emerging workloads. This approach has the potential to influence future CPU design practices not only for LoongArch but for other RISC architectures, including RISC-V and ARM.

Compliance with ethical standards

Disclosure of conflict of interest

No conflict of interest to be disclosed

References

[1] Z. Qian and H. Huang, "Design and implementation of Linux network computer system based on Loongson Mipsel architecture," in Proc. 2011 Int. Conf. Computer Science and Service System (CSSS), Jun. 2011, pp. 1209–1212.

[2] Z. Wang and Y. Wang, "Porting the Gold Linker to the LoongArch Architecture," in Proc. 2024 5th Int. Symp. Computer Engineering and Intelligent Communications (ISCEIC), Nov. 2024, pp. 579–585.

[3] J. L. Hennessy and D. A. Patterson, Computer Architecture: A Quantitative Approach, 6th ed. 2019.

[4] L. Chen and X. Ma, "Design and implementation of branch prediction based on LoongArch," in Third Int. Conf. Electronics Technology and Artificial Intelligence (ETAI 2024), vol. 13286, Sep. 2024, pp. 57–63.

[5] J. Chang, Y. Wang, Q. Meng, and Q. Li, "Implementation of Thread Local Storage Optimization Method Based on LoongArch," in Proc. 2023 42nd Chinese Control Conf. (CCC), Jul. 2023, pp. 2104–2109.

[6] M. Trofin, Y. Qian, E. Brevdo, Z. Lin, K. Choromanski, and D. Li, "MLGO: A machine learning guided compiler optimizations framework," arXiv preprint arXiv:2101.04808, 2021.

[7] N. Wu and Y. Xie, "A survey of machine learning for computer architecture and systems," ACM Computing Surveys (CSUR), vol. 55, no. 3, pp. 1–39, 2022.

[8] A. Singh, "Model Evolution Engine: A Step Toward Self-Evolving AI," Authorea Preprints, 2025.

[9] D. Amuru and Z. Abbas, "AI-Assisted Circuit Design and Modeling," in AI-Enabled Electronic Circuit and System Design: From Ideation to Utilization, Cham, Switzerland: Springer Nature, 2024, pp. 1–40.

[10] B. Hanindhito and L. K. John, "Accelerating ML workloads using GPU tensor cores: The good, the bad, and the ugly," in Proc. 15th ACM/SPEC Int. Conf. Performance Engineering, May 2024, pp. 178–189.

[11] C. Wang, Domain-Specific Computer Architectures for Emerging Applications: Machine Learning and Neural Networks. Chapman & Hall/CRC, 2024.

[12] Y. Chen et al., "An instruction set architecture for machine learning," ACM Trans. Comput. Syst. (TOCS), vol. 36, no. 3, pp. 1–35, 2019.

[13] S. S. Alahmari, L. O. Hall, P. R. Mouton, and D. B. Goldgof, "Repeatability of fine-tuning large language models illustrated using QLoRA," IEEE Access, 2024.

[14] W. Wang et al., "A Survey of AI Inference Technologies for On-Device Systems," IEEE Internet of Things Journal, 2025.

[15] R. Abdel-Salam, A. H. Abdel-Gawad, and A. G. Wassal, "VLCQ: Post-training quantization for deep neural networks using variable length coding," Future Generation Computer Systems, vol. 166, p. 107654, 2025.

[16] M. Van Baalen et al., "FP8 versus INT8 for efficient deep learning inference," arXiv preprint arXiv:2303.17951, 2023.

[17] Z. Hao et al., "Low-Precision Training of Large Language Models: Methods, Challenges, and Opportunities," arXiv preprint arXiv:2505.01043, 2025.

[18] Y. Jia et al., "CroApp: A CNN-based resource optimization approach in edge computing environment," IEEE Trans. Ind. Informatics, vol. 18, no. 9, pp. 6300–6307, 2022.

[19] L. Wen, L. Gao, X. Li, and H. Li, "A new genetic algorithm-based evolutionary neural architecture search for image classification," Swarm and Evolutionary Computation, vol. 75, p. 101191, 2022.

[20] M. Loni, S. Sinaei, A. Zoljodi, M. Daneshtalab, and M. Sjödin, "DeepMaker: A multi-objective optimization framework for deep neural networks in embedded systems," Microprocessors and Microsystems, vol. 73, p. 102989, 2020.

[21] S. A. Li, C. C. Hsu, C. C. Wong, and C. J. Yu, "Hardware/software co-design for particle swarm optimization algorithm," Information Sciences, vol. 181, no. 20, pp. 4582–4596, 2011.

[22] H. Jun, H. Ye, H. Jeong, and D. Chen, "AutoscaledSE: A scalable design space exploration engine for high-level synthesis," ACM Trans. Reconfigurable Technol. Syst., vol. 16, no. 3, pp. 1–30, 2023.

[23] Y. B. Bekele, D. B. Limbrick, and J. C. Kelly, "A survey of QEMU-based fault injection tools & techniques for emulating physical faults," IEEE Access, vol. 11, pp. 62662–62673, 2023.

[24] G. O. Osman, “Emulating the Internet of Things with QEMU,” 2020.

[25] C. Liu et al., “Industry-Oriented Lightweight Simulation System,” in Proc. 2023 IEEE 29th Int. Conf. Parallel and Distributed Systems (ICPADS), Dec. 2023, pp. 1099–1106.

[26] G. Tagliavini, S. Mach, D. Rossi, A. Marongiu, and L. Benini, “Design and Evaluation of SmallFloat SIMD extensions to the RISC-V ISA,” in Proc. DATE, Mar. 2019, pp. 654–657.

[27] W. A. Nunes, A. V. C. Dos Santos, and F. G. Moraes, “Accelerating Machine Learning with RISC-V Vector Extension and Auto-Vectorization Techniques,” in Proc. 2025 IEEE Int. Symp. Circuits and Systems (ISCAS), May 2025, pp. 1–5.

[28] C. Avalos Baddouh et al., “Principal kernel analysis: A tractable methodology to simulate scaled GPU workloads,” in MICRO-54: IEEE/ACM Int. Symp. Microarchitecture, Oct. 2021, pp. 724–737.

[29] E. Thompson and A. Carter, “Advances in Artificial Intelligence and Computer Science: Key Trends and Future Prospects in 2024,” AlgoVista: Journal of AI and Computer Science, vol. 2, no. 2, pp. 7–13, 2024.

[30] J. Hellström and M. Ghamlouch, “Assessing RISC-V Vector Extension for Machine Learning,” 2023.