PROVISIONAL PATENT APPLICATION

Title: Computer-Implemented Systems and Methods for Closed-Loop Optimization of Hybrid Compute Architectures Comprising Under-Specified Elements Using Language Model-Driven Search Space Shaping and Hardware-in-the-Loop Evaluation

Filing Type: Provisional Patent Application
Applicant: Kernel Keys LLC
Inventor(s): David H. Silver

ABSTRACT

A computer-implemented system and method for optimizing hybrid compute architectures comprising deterministic digital cores and under-specified compute elements (including analog, neuromorphic, approximate, and stochastic circuits). The system employs a language model to generate candidate artifacts specifying code and hardware configuration, deploys candidates to physical hardware, collects telemetry during execution, and iteratively refines generation based on measured performance. A key innovation is search-space shaping, wherein the language model modifies the intermediate representation, grammar, and constraints of the optimization space based on observed hardware behavior, enabling discovery of hardware-specific optimizations not expressible in the original formulation. The system produces deterministic deployable artifacts meeting specified resource, timing, and safety constraints. In robotic embodiments, optimized artifacts are deployed to distributed artificial neuron nodes integrated into robotic structures, enabling low-latency reflex control through embodied optimization on physical hardware.

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims priority to and incorporates by reference U.S. Provisional Patent Application No. 63/927,859, filed November 30, 2025, titled "Systems and Methods for Grammar-Constrained Code Generation and Execution on Resource-Limited Embedded Devices," the entire disclosure of which is incorporated herein by reference.

FIELD OF THE INVENTION

This invention relates to optimization of hybrid compute architectures comprising both deterministic digital elements and under-specified compute elements (including analog, mixed-signal, neuromorphic, approximate, and stochastic circuits), hardware-in-the-loop calibration, language model-driven code and configuration generation, and distributed computing systems for robotics and embedded applications.

BACKGROUND OF THE INVENTION

The Emergence of Under-Specified Compute

Modern computing increasingly incorporates elements whose functional behavior is not fully specified by a deterministic instruction set architecture (ISA). Such "under-specified" compute elements include:

- **Analog and mixed-signal circuits** — where output depends on component tolerances (typically ±5-10%), temperature coefficients (100+ ppm/°C), and supply voltage variations
- **Neuromorphic and spiking circuits** — including memristor-based crossbar arrays (Intel Loihi, IBM TrueNorth) with device-to-device variability in synaptic weights spanning orders of magnitude
- **Approximate arithmetic units** — designed to trade exactness for energy efficiency, with error characteristics that vary by input distribution
- **Stochastic and probabilistic logic** — where outputs are intentionally randomized for computation (e.g., Gaines' stochastic computing)
- **Near-threshold and sub-threshold circuits** — with timing that varies exponentially with supply voltage
- **Photonic and optical processors** — where matrix operations depend on interferometric alignment, temperature-dependent refractive indices, and manufacturing variation in waveguide dimensions

These architectures offer 10-1000× improvements in energy efficiency, compactness, and throughput for certain workloads (see Mead, "Analog VLSI and Neural Systems," 1989; Shen et al., "Deep learning with coherent nanophotonic circuits," Nature Photonics 2017). However,

they present a fundamental barrier: they are difficult to program, model, and verify using conventional compilation workflows that assume deterministic mapping between program and output.

The Limitations of Traditional Compilation

Traditional compilers (see Aho et al., "Compilers: Principles, Techniques, and Tools," 2006) assume a fully specified ISA with deterministic semantics. The LLVM framework (Lattner & Adve, 2004) and similar toolchains implement transformations that preserve semantic equivalence—a property that is undefined when hardware behavior is under-specified.

When the hardware mapping is variable or dependent on per-device calibration, the separation between "hardware design" and "software compilation" breaks down. Existing approaches include:

1. **Manual tuning by hardware experts** — Requires specialized expertise; does not scale to thousands of devices or complex behaviors. A single analog neural network may require weeks of expert tuning.
2. **Lookup tables (LUTs)** — Pre-characterize input-output mappings. For n-bit inputs, requires 2^n entries per function. At n=32, this is 4.3×10^9 entries—intractable. LUTs cannot generalize to unseen inputs.
3. **Monte Carlo simulation** — Simulates device variability statistically. Requires millions of samples for confidence; simulation models omit second-order effects (crosstalk, substrate coupling) that affect real hardware. Accuracy gap between simulation and silicon can exceed 20%.
4. **Exhaustive characterization** — Measures all input-output pairs on physical hardware. At 1ms per measurement, characterizing a 32-bit function requires 50 days of continuous measurement per device.
5. **Design-time optimization** — AutoML and neural architecture search (NAS) optimize before deployment (Hutter et al., "Automated Machine Learning," 2019). These approaches assume deterministic execution and do not handle per-device calibration or closed-loop adaptation.

None of these approaches enables rapid, automated deployment of complex behavior onto under-specified substrates with per-device adaptation.

The Promise of Language Models for Code Generation

Large language models (LLMs) have demonstrated remarkable ability to generate code (Chen et al., "Evaluating Large Language Models Trained on Code," 2021; Li et al., "Competition-level code generation with AlphaCode," Science 2022). Grammar-constrained decoding ensures syntactic validity (Scholak et al., "PICARD," 2021; Poesia et al., "Synchromesh," 2022).

However, prior work has focused exclusively on generating code for deterministic architectures with fully specified semantics. The potential for LLMs to:

1. Generate candidate artifacts spanning both code and hardware configuration
2. Adapt generation strategy based on physical telemetry from the target hardware
3. Modify the search space itself—grammars, intermediate representations, operators—based on observed hardware behavior

...remains unexplored in the prior art.

Distributed Sensing in Robotics

Robotic systems increasingly require distributed sensing and local computation. Biological nervous systems achieve this through networks of neurons performing local preprocessing, event detection, temporal integration, and reflexive action (Dahiya et al., "Tactile sensing—from humans to humanoids," IEEE Transactions on Robotics, 2010).

Engineering such systems requires:
- Programming many distributed compute nodes (tens to hundreds per robot)
- Calibrating each node to its local sensors and mounting conditions
- Enabling low-latency reflex behavior (<10ms) without central controller round-trips
- Ensuring safety constraints and bounded behavior
- Adapting to manufacturing variability and sensor drift over time

Existing robotic sensing systems use centralized processing (high latency) or hand-coded distributed logic (inflexible, not adaptive). No existing system addresses these requirements through closed-loop optimization on physical hardware.

RELATED WORK

Prior work on automatic code generation and compiler optimization assumes a target architecture whose semantics are fully specified and accurately modeled in software. For example, model-based code generation and hardware description language (HDL) synthesis frameworks (see WO2006034352, "Automatic Generation of Code for Component Interfaces in Models"; WO2008033344, "Hardware Definition Language Generation for Frame-Based Processing") generate executable code or HDL from high-level models and perform optimizations based on deterministic timing and resource models of the target hardware. These approaches do not address device-dependent, time-varying behavior that cannot be captured by a static simulation model.

Similarly, language-model optimization techniques (see US20150325235, US9972311, "Language Model Optimization for In-Domain Application"; EP2026327, "Language Model Learning System") adapt language models or grammar models based on observed usage statistics to improve recognition or prediction accuracy in textual domains. In these systems, a grammar or language model is updated offline or on resource-constrained devices based on linguistic data, but the target of optimization remains symbolic text processing rather than physical hardware behavior. The optimization does not involve telemetry from under-specified compute elements or modification of an intermediate representation for hardware code and configuration.

Dynamically updatable grammar models for resource-constrained devices (see US9922138, US20160350320, WO2016191313, EP3385946, "Dynamically Updatable Offline Grammar Model for Resource-Constrained Offline Device"; WO2017146803, "Facilitation of Offline Semantic Processing in Resource-Constrained Device") teach updating speech recognition grammars on mobile or embedded devices based on usage patterns. However, these systems adapt grammars for speech or text recognition rather than for generating executable artifacts for under-specified hardware. The grammar modifications are driven by linguistic usage data, not physical hardware telemetry.

Existing AutoML and neural architecture search (NAS) methods (see WO2024156237, EP4339843) search over fixed or parametrically defined model spaces using simulation or deterministic training objectives. While these methods may optimize architectures or hyperparameters for performance and resource usage, they assume that the underlying compute substrate is deterministic or at least accurately simulable. They do not address per-

device calibration of under-specified elements whose effective functionality must be learned in a hardware-in-the-loop manner.

Hardware-in-the-loop schemes have also been proposed in domains such as power systems and control (see WO2020102450, US10971931), where candidate configurations are tested on physical hardware or detailed simulators. However, these schemes typically evaluate a fixed configuration space or controller parameterization and do not employ language models to generate code and configuration artifacts, nor do they dynamically reshape the search space based on telemetry.

None of these approaches teaches or suggests using a language model to (i) generate candidate artifacts comprising both code and device-specific calibration parameters for under-specified compute elements, and (ii) modify an optimization search space—including intermediate representation operators, grammar rules, and constraints—based on telemetry collected from physical under-specified hardware.

ADVANTAGES OVER PRIOR ART

In contrast to prior work, the present invention directly targets hybrid compute architectures that include under-specified elements whose functional behavior is device-dependent, time-varying, and not accurately captured by static simulation models. Rather than relying exclusively on design-time models, the disclosed systems and methods perform hardware-in-the-loop optimization using telemetry collected from the physical device.

A first advantage is the use of a language model not only as a code generator but as a **search-space shaper** that proposes modifications to intermediate representation operators, grammar production rules, and constraint sets based on patterns observed in physical telemetry. This goes beyond prior language-model optimization and grammar-update techniques (US9972311, EP3385946, WO2016191313), which operate on textual data and do not alter a hardware optimization search space in response to device behavior.

A second advantage is the integration of a lightweight evaluator model executing on or near the target hybrid compute substrate and trained online using discrepancies between predicted and measured performance. This on-device evaluator enables rapid screening of candidate artifacts under real hardware conditions, which is not addressed by existing AutoML, NAS, or code-generation systems (WO2006034352, WO2024156237, WO2008033344) that assume deterministic or simulable targets.

A third advantage arises in the robotic embodiments, where the invention provides language-model-driven generation and adaptation of per-node artifacts for distributed artificial neuron nodes connected by an event-based nerve bus with safety envelopes and rollback. Existing hardware-in-the-loop and code-generation techniques (WO2020102450, US10971931) do not describe LM-optimized, telemetry-driven calibration of distributed neuron-like nodes with bounded-latency reflex behavior and per-node safety constraints in embodied robotic systems.

Unlike prior AutoML, NAS, and compiler optimization systems that assume deterministic, simulable hardware targets, the disclosed approach is explicitly designed for under-specified compute elements whose behavior must be characterized and optimized through hardware-in-the-loop telemetry. The use of a language model to reshape the optimization search space itself based on observed device behavior is not taught by existing language-model or grammar-update techniques, which operate purely on textual or symbolic data.

SUMMARY OF THE INVENTION

The present disclosure describes systems and methods for closed-loop optimization of hybrid and under-specified compute architectures. The disclosed approach comprises:

1. **Candidate Generation:** A language model generates candidate artifacts specifying code for deterministic elements and configuration for under-specified elements.
2. **Hardware-in-the-Loop Evaluation:** Candidates are deployed to physical hardware (or calibrated surrogates), and telemetry is collected during execution.
3. **Lightweight Scoring:** A fast, on-device or near-device evaluator model scores candidates based on telemetry.
4. **Search-Space Shaping:** The language model not only generates candidates but also proposes modifications to the search space itself—including changes to intermediate representations, operators, grammars, and constraints.
5. **Deterministic Deployment:** The process produces a deployable artifact (bytecode, microcode, configuration bits) that meets hard constraints on resources, timing, and safety.
6. **Distributed Neuron Nodes:** In robotic embodiments, the system deploys optimized artifacts to distributed artificial neuron nodes (ANN-nodes) integrated into robotic skin, joints, and structures.

DEFINITIONS

The following terms are used throughout this specification:

- **Under-specified architecture:** A compute element whose functional mapping is not fully specified (or is device/time dependent), including analog/mixed-signal blocks, stochastic/ approximate operations, neuromorphic cores, near-threshold timing-variant blocks, or "opaque" accelerators with calibration-dependent semantics.
- **Hybrid architecture:** A computing system comprising at least one deterministic digital core and one or more under-specified compute elements.
- **Candidate artifact:** Any of: program, bytecode, microcode, kernel schedule, accelerator configuration bitstream, routing map, quantization table, or calibration parameters.
- **Inner loop evaluator:** A lightweight model executing on or near the target hardware that generates a score or events (accuracy proxy, stability metric, constraint violation flag).
- **Outer loop generator/controller:** A language model (or ensemble) that proposes candidates and modifies the search space (grammar/IR operators, constraints, priors).
- **Artificial neuron node (ANN-node):** A physical module comprising (i) one or more sensors/actuators, (ii) a local compute element (deterministic + optional under-specified block), (iii) a communication interface, and (iv) a local evaluation/telemetry path.
- **Nerve bus:** A wired or wireless interconnect distributing power, timing, and messages between ANN-nodes and at least one hub controller.
- **Neuromorphic behavior:** Event-driven encoding, sparse messaging, local temporal integration, local adaptation, and bounded-latency output.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated herein and constitute part of this specification, illustrate embodiments of the invention and, together with the general description given above and the detailed description given below, serve to explain the principles of the invention.

FIG. 1 is a block diagram illustrating the overall system architecture (100) comprising: a hybrid compute substrate (110) including a deterministic digital core (112) and under-specified compute elements (114); an instrumentation subsystem (120) for telemetry collection; an inner loop evaluator (130) executing on or near the target hardware; an outer loop generator (140) comprising a language model; a search controller (150); and a deployment component (160) producing deterministic runtime artifacts.

FIG. 2 is a flowchart illustrating the search-space shaping mechanism (200) comprising: an initial search space definition (210) including intermediate representation operators (212) and grammar rules (214); a modification proposal generator (220) driven by the language model; a modification validator (230) testing proposed changes against a validation set; and an updated search space (240) incorporating validated modifications.

FIG. 3 is a flowchart illustrating the closed-loop optimization process (300) comprising: candidate generation (310); artifact compilation (320); deployment to physical hardware (330); workload execution (340); telemetry collection (350); scoring by the evaluator model (360); generation strategy update (370); and termination check (380) with output of final deployable artifact (390).

FIG. 4 is a block diagram illustrating a distributed artificial neuron node (ANN-node) architecture (400) comprising: a sensor array (410); an analog preprocessing block (420); an analog-to-digital converter (430); a local microcontroller (440); a communication transceiver (450); and a telemetry path (460) for collecting node-level metrics.

FIG. 5 is a diagram illustrating a robotic embodiment (500) comprising: a robotic structure (510) such as a gripper or limb; a plurality of ANN-nodes (520) distributed across the structure; a nerve bus (530) connecting the nodes; a hub controller (540); and an optimization system (550) performing closed-loop learning on physical robot operation.

FIG. 6 is a block diagram illustrating the structure of a deterministic deployment artifact (600) comprising: bytecode for the deterministic core (610); configuration bits for under-specified elements (620); calibration parameters (630); safety envelope specification (640) including bounds and forbidden states; version metadata (650); and rollback pointer (660).

DETAILED DESCRIPTION OF THE INVENTION

1. System Architecture

The system comprises:

1. A **hybrid compute substrate** including at least one deterministic digital core and one or more under-specified compute elements.
2. **Instrumentation** to collect telemetry from the substrate, including power consumption, latency, error rates, stability metrics, temperature, and drift.
3. A **candidate generator** using a language model to propose candidate artifacts.
4. A **search controller** implementing reinforcement learning, Bayesian optimization, or evolutionary search to guide the optimization process.
5. An **on-device evaluator model** providing rapid scoring of candidates.
6. A **verifier/constraint enforcer** ensuring candidates meet hard constraints.
7. A **deployment component** producing deterministic runtime artifacts.

2. Candidate Generation

The language model generates candidate artifacts that may specify:

- Code for the deterministic core (e.g., DSL programs, bytecode sequences)
- Configuration for under-specified elements (e.g., analog bias settings, weight matrices, routing configurations)
- Message-passing rules for distributed systems
- Calibration parameters for per-device tuning

Generation is constrained by:
- Grammar-guided decoding ensuring syntactic validity
- Constraint checking ensuring resource bounds
- Type systems ensuring semantic compatibility

3. Search-Space Shaping

A key innovation is that the language model not only generates candidates within a fixed search space but also proposes modifications to the search space itself. This meta-optimization

enables discovery of hardware-specific patterns that were not expressible in the original formulation. This telemetry-driven modification of intermediate representation operators, grammar rules, and constraints for hardware code and configuration differs from prior dynamically updatable grammar models (US9922138, EP3385946), which adapt grammars for speech or text recognition rather than for generating executable artifacts for under-specified hardware.

3.1 Operator Mutation

The intermediate representation (IR) comprises a set of operators $O = \{o_1, o_2, ..., o_n\}$. Each operator specifies a computational primitive (e.g., `READ_SENSOR`, `APPLY_FILTER`, `SEND_MESSAGE`). The language model may propose:
- **Addition:** o_new = {name, signature, semantics} based on patterns in high-performing candidates
- **Removal:** Delete operators with low utilization or association with poor performance
- **Modification:** Adjust operator parameters (e.g., precision, quantization level)

3.2 Grammar Expansion

The grammar $G = (V, \Sigma, R, S)$ comprises variables V, terminals Σ, production rules R, and start symbol S. Grammar expansion adds new rules to R:
- Identify recurring subtrees in high-performing candidate ASTs
- Abstract the subtree into a new non-terminal
- Add production rules generating the abstracted pattern
- Validate that expanded grammar maintains bounded resource usage

3.3 Constraint Adjustment

Constraints $C = \{c_1, c_2, ..., c_m\}$ bound resource usage and define safety properties. Based on telemetry:
- **Tightening:** Reduce bounds in regions with observed instability or constraint violations
- **Relaxation:** Expand bounds where hardware demonstrates unused headroom
- **Addition:** Introduce new constraints based on discovered failure modes

3.4 Prior Shifting

The generation prior P(artifact | context) is a probability distribution over candidate features. Bayesian updating shifts the prior based on observed performance:
- $P_{new}(feature) \propto P_{old}(feature) \times L(performance \mid feature)$

- High-performing patterns receive increased probability mass
- Low-performing patterns are suppressed

3.5 Validation Protocol

Search-space modifications are validated before adoption:
1. Generate N_test candidates using the modified space
2. Verify all test candidates satisfy hard constraints (static analysis)
3. Execute test candidates on hardware or calibrated surrogate
4. Accept modification if: mean(performance) ≥ threshold AND max(constraint_violation) = 0

4. Hardware-in-the-Loop Evaluation

Candidate artifacts are executed on the physical target hardware (or a calibrated surrogate). Telemetry collected includes:

- **Functional metrics:** Accuracy, output quality, task completion
- **Resource metrics:** Power consumption, latency, memory usage
- **Stability metrics:** Variance across executions, drift over time
- **Thermal metrics:** Temperature, thermal headroom
- **Constraint violations:** Safety envelope breaches, rate limit violations

5. On-Device Evaluator

A lightweight learned model (e.g., small MLP, GNN, or spiking network) executes on or near the target hardware to provide rapid scoring. This enables:

- Evaluating many candidates without full system measurement overhead
- Detecting constraint violations before they cause harm
- Providing gradient-like signals for optimization

The on-device evaluator is periodically updated based on ground-truth measurements.

6. Deterministic Deployment

The optimization process produces a deployable artifact that is:

- **Deterministic:** Executes with bounded, predictable behavior
- **Resource-bounded:** Fits within specified memory and compute limits

- **Verified:** Passes static analysis for safety properties
- **Versioned:** Supports rollback to prior versions

The artifact format includes bytecode for the deterministic core, configuration bits for under-specified elements, and metadata specifying the safety envelope.

7. Distributed Artificial Neuron Nodes

In robotic embodiments, the system deploys optimized artifacts to distributed ANN-nodes. Existing hardware-in-the-loop and robotics control schemes (WO2020102450, US10971931) do not employ a language model to generate and continually adapt node-local programs and calibration parameters for distributed neuron-like nodes based on embodied telemetry, while enforcing safety envelopes and rollback at both node and system levels. Each node includes:

- One or more sensors (tactile, strain, temperature, proximity, IMU)
- Optional actuators (local heating, haptic feedback, LED indicators)
- A local compute element executing the deployed artifact
- A communication interface connected to the nerve bus
- Instrumentation for collecting local telemetry

ANN-nodes may be implemented on flexible substrates (flex PCB, elastomeric circuits, cable-like strips) and integrated into robotic skin, joints, grippers, or soft robotic structures.

8. Neuromorphic Behavior

ANN-nodes may implement neuromorphic behavior including:

- **Event-driven encoding:** Transmitting events (spikes) on threshold crossings rather than continuous streams
- **Local temporal integration:** Accumulating signals over time with leaky integration
- **Sparse messaging:** Transmitting only when significant changes occur
- **Local adaptation:** Adjusting local parameters based on recent history
- **Bounded-latency output:** Guaranteeing response within specified time bounds

9. Hierarchical Learning

The system supports hierarchical optimization where:

- A hub controller performs higher-level optimization across multiple nodes
- Node artifacts are periodically updated based on hub evaluation
- Per-node calibration compensates for manufacturing variability and mounting differences
- Global objectives (task success, energy, safety) are decomposed into per-node contributions

10. Safety Envelope

Deployed artifacts include safety constraints that:

- Bound actuator commands (maximum force, velocity, temperature)
- Limit rate-of-change (slew rate, acceleration)
- Cap message frequency (preventing bus flooding)
- Define forbidden states (e.g., simultaneous opposing actuations)
- Trigger rollback upon violation detection

ALTERNATIVE EMBODIMENTS

A. On-Device Generation

A quantized language model (4–10M parameters) stored in MCU Flash generates and refines artifacts locally, enabling fully autonomous adaptation.

B. FPGA-Based Evaluator

The inner loop evaluator is implemented in hardware logic for minimum latency and maximum throughput.

C. Multi-Substrate Heterogeneous Systems

The system optimizes across multiple different under-specified substrates (e.g., analog front-end + neuromorphic core + digital back-end).

D. Transfer Learning Across Devices

Optimization results from one device inform priors for new devices, enabling rapid deployment to manufacturing batches.

E. Formal Verification Integration

The verifier component incorporates formal methods (model checking, SMT solving) to prove safety properties before deployment.

ADVANTAGES OF THE INVENTION

- Enables automated programming of under-specified compute architectures
- Reduces time and expertise required for hardware/software co-design
- Adapts to per-device variability through closed-loop calibration
- Enables distributed intelligent sensing in robotic systems
- Produces deployable deterministic artifacts meeting hard constraints
- Supports iterative improvement through search-space shaping

---

EXAMPLES

Example 1: Analog Sensor Preprocessing

A tactile sensor array (16×16 taxels) uses analog preprocessing for energy efficiency. The under-specified elements comprise:
- Analog filter bank with programmable coefficients (RC time constants)
- Variable-gain amplifiers with 6-bit DAC-controlled gain
- Comparator thresholds for event detection

The deterministic core (ARM Cortex-M0+) executes post-processing code generated by the language model. The optimization objective minimizes power consumption while maintaining classification accuracy ≥95% on a texture discrimination task.

Results: The system discovers filter coefficients and post-processing code achieving 12× lower power (0.8mW vs 9.6mW) than a fully digital baseline while maintaining 96.2%

accuracy. Search-space shaping added a new operator `TEMPORAL_DIFF` after observing that high-performing candidates repeatedly implemented difference computations.

Example 2: Robotic Gripper with Reflex Control

A parallel-jaw gripper with 50 ANN-nodes distributed across finger surfaces implements slip detection and grip stabilization reflexes. Each ANN-node comprises:
- 4 taxels (piezoresistive pressure sensors)
- 1 strain gauge (tangential force)
- ATtiny1616 MCU (16KB Flash, 2KB RAM)
- I²C communication at 400kHz

The optimization system learns:
- Per-node calibration parameters θ_i compensating for manufacturing variation (±15% sensor sensitivity)
- Slip detection thresholds (tangential/normal force ratio)
- Reflex response gains (grip force adjustment per unit slip)

Results: The system achieves 2.8ms slip-to-correction latency (vs 45ms for centralized processing), successful grasp of fragile objects (eggs, foam cups) with <2N grip force, and 99.7% grasp success rate across 1000 trials. Search-space shaping discovered a `LEAKY_INTEGRATE` operator for temporal filtering not present in the initial IR.

Example 3: Soft Robot Skin

A soft robot (pneumatic actuator with silicone skin) has 200 strain sensors distributed over its body using ANN-nodes on flex PCB strips. Each node samples 4 strain gauges at 1kHz and must compress the data to event-based messages (≤10 messages/second) to avoid bus congestion.

The optimization objective balances:
- Shape reconstruction accuracy (measured via ground-truth motion capture)
- Message rate (constraint: ≤10 msg/s/node)
- Latency (constraint: ≤20ms from deformation to hub notification)

Results: The system learns compression mappings achieving 0.8mm shape reconstruction error (vs 2.1mm for uniform thresholding), 8.2 average messages/second, and 12ms average latency. Hierarchical optimization at the hub level coordinates node behaviors to avoid bus contention.

Example 4: Neuromorphic Sensor Fusion

A multi-modal sensor node combines:
- Analog neuromorphic preprocessor (memristor crossbar, 32×32)
- Digital spiking neural network (256 neurons, 16-bit membrane potentials)
- Event-based camera interface (DVS)

The under-specified element (memristor crossbar) exhibits ±30% conductance variation across devices. The optimization system learns per-device weight mappings achieving equivalent accuracy across a batch of 50 devices, despite hardware variation. Transfer learning priors reduce calibration time from 2 hours to 8 minutes for new devices.

CONCLUSION

The invention provides an integrated system for closed-loop optimization of hybrid and under-specified compute architectures, including distributed artificial neuron nodes in physical robots. The combination of language model-driven generation, hardware-in-the-loop evaluation, search-space shaping, and deterministic deployment enables automated programming of previously intractable hardware substrates.

While certain embodiments have been described in detail above, it will be appreciated that the scope of the invention is not limited to these embodiments, but rather extends to all variations, modifications, and equivalents as defined by the appended claims.

CLAIMS

The claims for this application are set forth in a separate claims document incorporated herein by reference.