Towards Algorithmically Grounded Embedded AI Models

Zhenyu Wang
University of North Carolina at Chapel Hill
zywang@cs.unc.edu

Md Yusuf Sarwar Uddin
University of Missouri Kansas City,
muddin@umkc.edu

Shahriar Nirjon
University of North Carolina at Chapel Hill
nirjon@cs.unc.edu

ABSTRACT

Embedded systems were once built with clarity—each line of code grounded in an algorithm, each behavior traceable and explainable. But as AI models rapidly replace classical methods in sensing, scheduling, decision, and control, we have gained accuracy at the cost of trust. Today's neural networks are black boxes, assembled by intuition or brute force, leaving us unable to explain, debug, or control their behavior. We argue this is not just a tooling issue, but a design flaw: explainability has long been an afterthought, with networks built first and interpreted later. We advocate a principled approach where networks are grounded in algorithms and designed with internal anchors—expected intermediate behaviors, invariants, or interpretable signals—that support debugging and interpretation. This paper presents *Algorithm-Informed Neural Networks* (AINN)—architectures shaped by algorithms as implicit inductive bias. By decomposing algorithms into logic blocks, we build modular networks that are easier to train, interpret, and debug. As proof of concept, we present two use cases—*fall detection* and *keyword spotting* problems—showing how algorithmic structure improves training efficiency and enables effective debugging.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; *Machine learning approaches*; • **Computer systems organization** → *Embedded and cyber-physical systems*; • **Human-centered computing** → *Ubiquitous and mobile computing systems and tools.*

KEYWORDS

Algorithm-informed Neural Networks, Explainable AI, Debuggable Neural Networks.

1 INTRODUCTION

From Algorithms to AI: A Paradigm Shift. Embedded Sensor systems have long been built on well-understood algorithms—compressed sensing [1], DVFS [2], Kalman/Bloom filters [3, 4], PID controllers [5], B/X-MAC protocols [6, 7], Markov models [8], tri-/multi-lateration [9, 10], LEACH [11], SPIN [12], Direct Diffusion [13], FTSP/TPSN [14, 15], RMS/EDF [16], Paxos/Raft [17–19]—designed with mathematical models, bounded guarantees, and tight control over system behavior. Every line of code has a purpose; every signal transformation follows from known principles. These systems are not just functional—they are interpretable, analyzable, and debuggable. But as AI has matured, neural networks have increasingly replaced these traditional approaches in applications—from gesture recognition [20, 21] and audio sensing [22, 23] to real-time control [24, 25]—offering impressive empirical gains. Yet these models are rarely derived from first principles. Their architectures are often shaped by intuition and trial-and-error, with little understanding of how or why they work. This marks a fundamental shift in design methodology: from systems constructed with intent to systems discovered through data.

The Trust Gap: Performance without Explanation or Control. In classical embedded systems, developers had precise insight into how each computation contributed to the final outcome [26]. In contrast, AI-powered systems increasingly behave as black boxes [27]. This has led to a wave of interest in explainable AI (XAI), with a proliferation of techniques such as saliency maps [28], activation heatmaps [29], feature attribution [30], and surrogate modeling [31]. But these approaches are fundamentally *post hoc*: they attempt to explain behavior that was never explicitly structured to be explainable. As a result, explanations are often vague, unstable across similar inputs, or divorced from the actual internal reasoning of the model [32–34]. Despite years of work, XAI has failed to deliver robust, actionable understanding—especially in safety-critical, real-time embedded contexts.

Why Explainability is Failing: No Expectations, No Accountability. At the root of this failure is a deeper problem: we are trying to explain AI models that were never designed with explainability in mind. In traditional computer programming, we can explain and debug a piece of code because every line is written with intent. We know what a loop should iterate over, what a condition should enforce, and how a variable's value should change. When an output is wrong, we can identify which statement violates its expected behavior. Neural networks, by contrast, are not built with this mindset. We do not assign any functional expectation to individual neurons or layers; we simply optimize them for end-to-end performance. When a neuron fails, we do not know what it is supposed to compute in the first place—so how can we tell what goes wrong? Explainability fails not because neural networks are inherently mysterious, but because *we have never imbued them with semantic intent.* Without expectation, there is nothing meaningful to explain—and debugging becomes guesswork.

Algorithmic Grounding as Inductive Bias. In traditional machine learning, inductive bias plays a crucial role in constraining the space of models a learner can explore. Even in data-driven settings like curve fitting, one begins with a strong prior: whether the underlying relationship is linear, quadratic, or exponential. This bias shapes not just the learning outcome but also the interpretability and generalizability of the model. In contrast, modern neural networks—particularly in embedded AI applications—are designed with little or no such discipline. Their architectures are assembled based on empirical success in other domains, manual tuning, or automated architecture search, rarely reflecting the structure or constraints of the underlying problem. We argue that classical algorithms—developed over decades for tasks such as signal processing, scheduling, control, and pattern recognition—should serve

as the *implicit algorithmic inductive bias* in neural network design. Algorithms offer a structured, interpretable, and domain-grounded scaffold that can constrain learning in a principled way and restore intent to neural computation.

A Vision for Modular and Observable Neural Architectures. We envision *Algorithm-Informed Neural Networks* (AINNs) as a new class of neural architectures that embed algorithmic structure directly into the network's design. In this vision, a classical algorithm is decomposed into functional or *logic* blocks, each mapped to a corresponding *neural* module. These modules preserve the input-output relationships and operational invariants of their source blocks, resulting in a network that mirrors the logic of the algorithm it replaces. The resulting architecture is inherently modular, interpretable, and traceable. It enables fine-grained observability at the block level, which facilitates anomaly detection through activation monitoring and execution path analysis. Two key benefits emerge: first, by constraining the learning space with algorithmic structure, these networks typically require less data and training effort to converge; and second, their modularity enables targeted debugging—errors can be localized to specific blocks, and internal failures can be diagnosed with semantic meaning, a capability lacking in conventional deep networks.

Positioning Relative to Existing Structured Neural Methods. Our approach is inspired in spirit by *physics-informed neural networks (PINNs)* [35], which incorporate physical laws into the training objective by embedding differential equations as soft constraints. However, unlike PINNs, algorithm-informed neural networks do not merely regularize the solution—they reconstruct the algorithm itself in neural form, preserving intermediate reasoning steps and ensuring interpretability throughout the execution flow. These networks are also distinct from *neuro-symbolic systems* [36], which combine symbolic logic and neural components in hybrid architectures. In contrast, proposed AINNs are *fully neural*, but their structure is guided by algorithms as an inductive bias—not layered on top as symbolic reasoning or rule engines. The algorithm serves as a blueprint, not a supervisory rule set, and thus shapes the very way the network processes information. This fusion of neural flexibility with algorithmic grounding positions our approach as a unique and promising direction for developing explainable, controllable, and efficient AI systems in embedded environments.

Proof of Concept and Evaluation. We demonstrate the viability of algorithm-informed neural networks through two sensor-enabled embedded AI applications. The first is *fall detection* from wearable accelerometer data that classifies *fall* vs *not fall*, guided by a Hidden Markov Model (HMM) blueprint [8]. The second is *keyword spotting* from audio that classifies keywords–such as *yes*, *no*, *up*, *down*, *left*, *right*, *on*, *off*–guided by dynamic time warping (DTW) [37] and K-nearest neighbors (KNN) [38]. In both cases, algorithmic components such as state updates, sequence alignment, and decision rules are translated into neural blocks that enforce invariants and are monitored for anomalous behavior. These algorithmic blueprints constrain network structure, simplify training, and enable interpretable execution—allowing errors to be traced to specific functional modules and demonstrating improved efficiency, transparency, and debuggability.

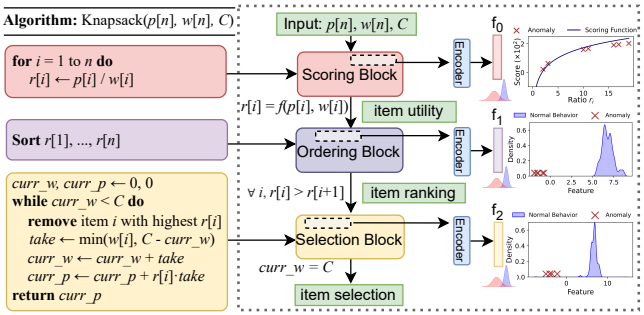


Figure 1: Algorithmic steps are mapped to neural blocks that enforce invariants, with each block's activation and invariant violations monitored for anomalies—enabling debugging and error localization.

2 ALGORITHM-INFORMED NEURAL MODEL

Overview. We define *Algorithm-Informed Neural Networks* (AINNs) as deep neural architectures whose structure, data flow, and interpretability are explicitly derived from a guiding algorithmic blueprint. In an AINN, each major step of the blueprint is represented by a corresponding *neural block* that mirrors the logic and constraints of that step. These blocks exchange intermediate outputs according to the blueprint's control flow and collectively form the full network. Each block enforces known invariants and exposes its activations for inspection, allowing errors to be localized to specific algorithmic stages. Blocks can be trained independently or jointly, and their internal representations define latent subspaces that are interpretable either by design (through known algorithmic semantics) or through post-hoc analysis after training.

Example. Figure 1 shows an AINN for the greedy knapsack [39] algorithm with three neural blocks. Each block enforces algorithmic logic and invariants and is monitored for out-of-distribution behavior; e.g., the *scoring* block learns a function $\alpha(\log p_i - \log w_i) + \beta$, notably closely correlated with the optimal p_i/w_i. During inference, if an incorrect result occurs from *scoring*, the error can be traced to this block. Likewise, errors in *sorting* or *selection* can be localized to their respective blocks through similar monitoring mechanisms.

Properties of AINNs. Three defining features characterize AINNs and distinguish them from conventional neural networks:

- First, *AINNs are constructed from an algorithmic blueprint.* Each part of the network mirrors an algorithmic step, yielding a coherent, logic-guided structure without trial-and-error design.
- Second, *AINNs execute information step by step, like algorithms.* Logic blocks are mapped to neural blocks that respect algorithmic dependencies, enabling an organized flow while allowing flexibility in implementation.
- Third, *AINNs support program-like debugging.* Errors can be traced to specific neural blocks corresponding to logic blocks, with block-level monitoring exposing intermediate behaviors for targeted fixes and interpretation.

Advantages of AINNs: AINNs offer two major advantages:

- First, *reduced training data requirement:* AINNs need less data because algorithmic blueprints constrain network structure—each neural block learns a well-defined step, enabling focused and efficient training.

- Second, *enabling error localization and debugging:* AINNs enable precise debugging since each neural block maps to a known logic block, and block-level monitoring reveals internal states that pinpoint errors.

3 FUNDAMENTAL CHALLENGES

3.1 Algorithm Grounding

Goal. Algorithms—in forms such as pseudocode, control flows, or high-level flowcharts—offer a rich inductive bias for building modular and interpretable neural networks. Our goal is to translate algorithmic blueprints into neural architectures by identifying and mapping logic blocks to neural blocks. Key challenges include handling diverse algorithm forms, identifying logical components, dealing with incomplete specifications, and ensuring fidelity and observability. We explore two strategies: a top-down approach that mirrors the algorithm's structure, and a bottom-up method that incrementally aligns neural components with logic.

Methodology. To build AINNs, we propose the following methodology: (1) *Decomposing algorithm forms*—addressing the complexity of diverse algorithm representations (e.g., programs, pseudocode, flowcharts, and data flow diagrams) by breaking them into *logic* blocks, identified through manual annotation, code analysis tools, or large language models (LLMs); (2) *Translating to neural blocks*—converting these logic blocks into neural counterparts, approximating missing information such as inputs, outputs, and pre/post-conditions to preserve fidelity to the original algorithm; (3) *Exploring construction approaches*—employing a bottom-up method that incrementally replaces logical blocks while preserving structure, and a top-down method that jointly trains neural blocks, incorporating constraints like expected outputs into the loss function; (4) *Ensuring observability and fidelity*—using additional encoding layers and diagnostic tools to monitor latent space relationships, enabling runtime error detection and ensuring the neural network adheres to the intended algorithmic behavior.

Incremental Neural Substitution. This approach constructs an AINN by incrementally replacing logic blocks with neural counterparts—a bottom-up strategy suited for cases where intermediate outputs are unavailable or costly to compute. It trains the neural blocks using available labels or by minimizing the discrepancy between the final algorithm output and that of the partially substituted model. Non-differentiable operations (e.g., sorting or discrete selection) are handled using differentiable relaxations such as soft sorting [40] or Sinkhorn normalization [41, 42], or via non-gradient optimizers [43, 44]. This enables logic-aligned neural behavior even without explicit supervision and facilitates the discovery of improved algorithmic heuristics.

Block-Wise Joint Training. This approach replaces each logic block in an algorithm with a neural block and trains them jointly. For each block k, the neural network $N_k(X_k; \theta_k)$ is trained to approximate the output Y_k of its corresponding logic block, minimizing a weighted sum of block-level losses: $\mathcal{L} = \sum_{k=1}^{n} \lambda_k \| Y_k - N_k(X_k; \theta_k) \|_m$, where λ_k controls each block's contribution. This joint optimization preserves local fidelity and interpretability while coordinating learning across the entire algorithmic structure for unified, end-to-end performance improvement.

Invariants. Each neural block in an AINN is constrained by a set of *invariants*—conditions that must hold true for its inputs, outputs, or intermediate representations throughout execution. Invariants serve as algorithm-informed guardrails that preserve logical or physical consistency, ensuring the block performs its intended function. They can express relationships such as *monotonicity* (e.g., scores must remain sorted after a ranking step), *boundedness* (e.g., probabilities in an HMM must sum to one), or *conservation* (e.g., total resource usage in a knapsack block cannot exceed capacity). Invariants may be directly derived from the guiding algorithm's specification, discovered through symbolic or analytical reasoning, or implemented as regularization terms during training. By enforcing these constraints, AINNs maintain structural fidelity to the algorithm, promote stable learning, and enable interpretable and verifiable block-level behavior.

3.2 Debuggable Neural Blocks

Goal. Enabling effective debugging is one of the core motivations for designing algorithm-informed neural networks. Our goal is to localize the source of errors in AINNs by monitoring the behavior of individual neural blocks during execution, even in the absence of intermediate ground-truth outputs. Key challenges include the difficulty of tracing errors through complex, interconnected blocks, the exponential number of possible execution paths, and the hidden nature of internal logic. We explore two core strategies: profiling subnetworks to isolate faults within large neural blocks, and encoding their activations or intermediate outputs to detect out-of-distribution behavior for interactive, program-like debugging.

Methodology. To model and monitor neural block behavior, we propose the following methodology: (1) *Selecting subnetworks*—instead of monitoring the entire neural block, which may be prohibitively large, a smaller subnetwork is chosen whose behavior is sufficient to distinguish between regular and irregular activation patterns. While this faces challenges such as the exponential number of possible subsets, it reduces complexity and focuses the analysis. (2) *Profiling and encoding the subnetwork*—the selected subnetwork is profiled using an encoder to produce latent feature vectors. By collecting data with known ground-truth behavior for the corresponding logic block, distributions of regular and irregular patterns are estimated. Out-of-distribution detection techniques are then applied to distinguish normal from anomalous activations. (3) *Discovering hidden logic within the neural block*—when the algorithmic logic is unknown but an equivalent neural representation exists, the goal is to reverse-engineer the embedded logical operations by inverting the logic-to-neural conversion. (4) *Developing a debugging tool*—the complete methodology is packaged into an interactive diagnostic tool that monitors network behavior, flags anomalies, and helps isolate, modify, and retrain faulty neural blocks.

Neural Block Monitoring. Neural block behavior can be monitored at inference time to support debugging by identifying the source of execution errors without requiring ground-truth intermediate outputs. This is achieved through two complementary

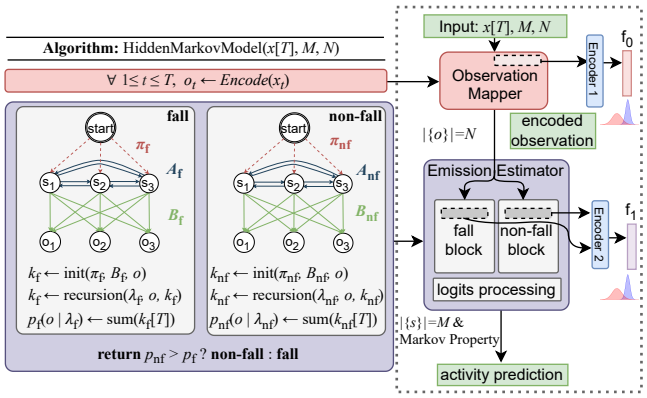
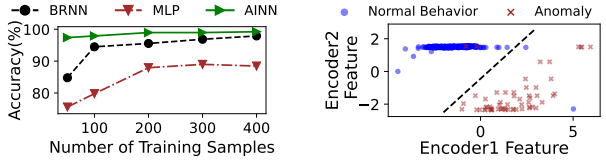


Figure 2: AINN is constructed by mapping HMM with M hidden states (s_i) and N observation states (o_i).



(a) Accuracy across baselines with different training sample sizes.

(b) Anomaly detection of AINN.

Figure 3: AINN-based fall detection: evaluation results.

strategies. First, the internal activation A of each neural block is encoded into a latent feature vector $z = E(A)$, and compared against a distribution $\mathcal{D}_{\text{normal}}$ built during training or profiling. Out-of-distribution detection methods flag a neural block as anomalous when its current activation deviates significantly from expected patterns. Second, the block's output $\hat{y}$ is evaluated against known logical properties $\mathcal{P}(x, \hat{y})$, such as bounds, monotonicity, or structural constraints derived from the original algorithm. When violations occur, they indicate that the error likely originates in or before the corresponding block.

Error Localization and Debugging. A key advantage of this approach is its ability to localize faults at the level of individual neural blocks—unlike existing debugging or interpretability methods, which focus on final outputs or coarse global patterns. It can detect "mismatched" cases where the output is correct but internal activations are anomalous, or vice versa—often signaling fragile generalization, silent failures, or overfitting. By isolating such anomalies, block-level monitoring provides a powerful new tool for understanding and improving neural network behavior.

4 EXAMPLE 1 – FALL DETECTOR

Sensing Problem. Fall detection from wearable is a canonical sequence–pattern recognition task: given a time series of accelerometer readings, decide whether a segment contains a fall. It is representative of broader physical-signal recognition problems—gestures and daily activities, speech and acoustic events, facial expressions, and physiological episodes—where temporal sensor streams are mapped to semantic labels [45–51]. Existing fall-detection systems increasingly rely on neural networks; in our study, we include two strong baselines—a bidirectional recurrent neural network (BRNN) [52] and a multilayer perceptron (MLP) [53]—implemented under identical model and deployment settings. We then construct an AINN for the same task and compare against these baselines to assess accuracy, data efficiency, and interpretability.

Algorithmic Blueprint. We select the *Hidden Markov Model (HMM)* as the algorithmic blueprint because fall detection from wearable accelerometers is inherently sequential, and HMMs provide interpretable state dynamics and principled probabilistic decoding [8]. Concretely, we model two class-specific HMMs (fall vs. non-fall) with parameters $\lambda = (\pi, A, B)$ capturing initial state distribution, state transitions, and emissions [54–56]. Given a sensor sequence $X = (x_1, \ldots, x_T)$, the forward algorithm computes class likelihoods $P(X \mid \lambda)$; classification follows by comparing these likelihoods.

AINN Construction. Our AINN uses two neural blocks aligned to the HMM blueprint: (i) an *Observation Mapper* (windowed accelerometer → latent features), implemented as a lightweight MLP for low-latency, on-device per-window mapping with few parameters; and (ii) an *Emission Estimator* (latent features → class/state emission scores), implemented as a small Transformer to capture short-/long-range temporal dependencies under tight memory/ runtime. The HMM forward recursion and final decision remain fixed, differentiable layers that consume neural emissions.

Invariants are enforced at block I/O: (i) Observation Mapper outputs are bounded; (ii) Emission Estimator scores are nonnegative and simplex-normalized; (iii) state transition probabilities form a row-stochastic process; (iv) cumulative forward score provides an approximation of the sequence likelihood along valid paths in log space —supporting interpretable execution, block-level monitoring, and error localization.

Training Process. We follow a two-stage procedure using the pretrained HMM as a reference. The accelerometer stream is segmented into ≈1 s windows (250 samples) and summarized (mean, std, median of magnitude), then fed to the *Observation Mapper* (MLP), trained with final labels to produce task-specific latents. Next, we freeze the mapper and train the *Emission Estimator* (small Transformer) to approximate emissions while aligning to HMM structure using a composite loss: $\mathcal{L} = \mathcal{L}_{\text{cls}} + \gamma \sum_{\eta \in \{f,nf\}} \left(\mathcal{L}_{l1}(\pi_\eta, \hat{\pi}_\eta) + \mathcal{L}_{l1}(A_\eta, \hat{A}_\eta)\right)$, where $\mathcal{L}_{\text{cls}}$ is the classification loss, $\mathcal{L}_{l1}$ is the ℓ_1 loss, and γ balances regularization. This staged training preserves interpretability while improving accuracy and data efficiency.

Result—Training Efficiency. We evaluate training efficiency using the publicly available UMAFall dataset [57], which contains 145 fall instances collected from wearable accelerometers. The goal is to assess whether AINN learns more effectively than typical neural networks. We compare against two baseline models: a state-of-the-art fall detection model (BRNN—bidirectional GRU layer followed by batch normalization and two fully connected layers) from [52], and an MLP [53] of comparable size to our AINN. All models are trained on a system with an Intel Core i9-9900K CPU, 32 GB RAM, and an NVIDIA RTX 2080 GPU. We vary the amount of training data and measure classification accuracy. As shown in Figure 3 (a), AINN achieves a 12.60%–21.78% accuracy gain over the baselines using only 6.58% of the training data, while operating with a 1.67–5.49× smaller model and a 5.72 ms average inference time on a Google Pixel 6a. This efficiency arises from each neural block being structurally constrained by the logic of the HMM component

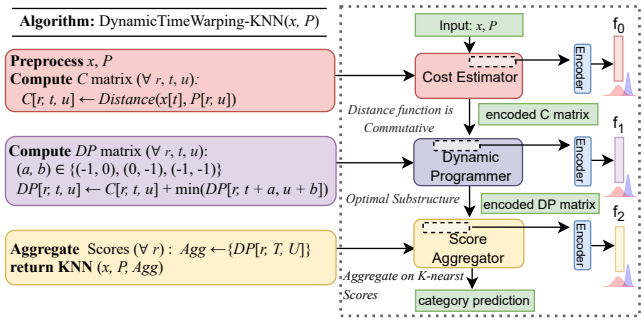
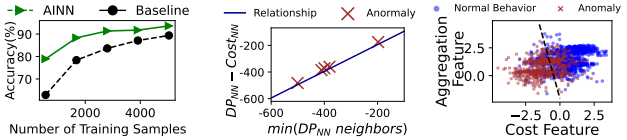


Figure 4: AINN construction: algorithm-to-neural mapping of DTW (distance + DP alignment) and KNN (neighbor selection) blocks.



(a) Accuracy across baselines with different training sample sizes.

(b) Relationship between Dynamic Programmer estimate and ground truth.

(c) Anomaly detection based on Cost and Aggregation features.

Figure 5: AINN-based keyword spotting: evaluation results.

it replaces, demonstrating the benefit of algorithm-informed design for sequence modeling tasks such as fall detection.

Result—Debuggability. The previously trained AINN achieved near-perfect accuracy (99.21%), leaving little room to observe failures. To evaluate debuggability, we train a reduced version with fewer iterations, yielding a lower accuracy of 84.51%. Encoders are attached to the Observation Mapper and Emission Estimator to monitor their internal behaviors. Each encoder outputs a scalar logit, and together the two logits form a 2D representation of the network's execution for each input sequence. As shown in Figure 3 (b), each execution is plotted as a point colored by correctness (blue for correct, red for incorrect), with a linear decision boundary separating anomalous executions at 98.95% accuracy. The boundary corresponds to a zero decision threshold ($p = 0.5$ after sigmoid), the standard equal-odds point in binary classification [58].

Because the boundary is a weighted sum of encoder logits, the sign and magnitude of each weighted logit indicate which block contributes most to an error. For example, a dominant positive logit from the emission encoder identifies the Emission Estimator as the likely source. This enables AINN to localize faults without ground truth, facilitating targeted and interpretable debugging.

5 EXAMPLE 2 – KEYWORD SPOTTER

Sensing Problem. Keyword spotting (KWS) from short speech utterances is a canonical sequence–pattern recognition task: given a time series of acoustic features, decide whether a segment contains a target keyword (e.g., "yes", "stop", or"Alexa"). It is representative of broader acoustic-signal recognition problems—phoneme and word recognition, wake-word detection, speaker identification, and environmental sound classification—where temporal audio streams are mapped to semantic labels [59–61]. Existing KWS systems increasingly rely on neural networks; in our study, we include a strong state-of-the-art baseline [62] implemented under identical model and deployment settings. We then construct an AINN for the same task and compare against this baseline to assess accuracy, robustness to temporal misalignment, and interpretability.

Algorithmic Blueprint. Each keyword class $k \in \{1, ..., S\}$ is represented by a small set of R_k reference utterances $\{P_k^m \in \mathbb{R}^{U \times F}\}$, each with a temporal length of U and F acoustic features. Given a query utterance x with temporal length of T, we compute its similarity to all references and classify it using a K-nearest neighbors [38] rule $\hat{y} = \arg\min_k \text{KNN}(\{D(x, P_k^m)\})$, where $D(x, P_k^m)$ measures dissimilarity. The distance $D(\cdot, \cdot)$ is computed via *Dynamic Time Warping (DTW)* [37], which aligns two sequences that differ in length or speaking rate. For each pair of frames, DTW accumulates a minimal path cost $DP[t, u] = d(x_t, p_u) + \min_{(a,b)\in\{(-1,0),(0,-1),(-1,-1)\}} DP[t + a, u + b]$, where $d(\cdot, \cdot)$ is the frame-level distance. The final cost $D(x, P_k^m) = DP[T, U]$ quantifies the optimal temporal alignment.

AINN Construction. The AINN mirrors the DTW–KNN algorithm through three neural blocks—*Cost Estimator*, *Dynamic Programmer*, and *Score Aggregator*—each corresponding to a key computation in the algorithmic blueprint. The *Cost Estimator* is a temporal CNN (Conv1D) encoder with a cosine—bilinear (Siamese-style) matcher that computes pairwise similarities between latent query and reference frame embeddings. It is applied $R \times T \times U$ times per utterance (via a single einsum operation) to produce the local cost matrices $C \in \mathbb{R}^{T \times U}$ and $C_{\text{all}} \in \mathbb{R}^{R \times T \times U}$, which serves as neural analogs of DTW's cost matrix, where $R = S \times R_k$. The *Dynamic Programmer* is a compact feedforward network (layer normalization + two dense layers) that learns a differentiable soft-min operation over three neighboring cells to emulate DTW's recursive update rule. It is applied to every cell of each cost matrix—$T \times U$ times per reference—to construct the accumulated alignment cost matrix $DP \in \mathbb{R}^{T \times U}$, capturing the minimal path costs across all time steps. The *Score Aggregator*—a differentiable sorting selector (TorchSort [63]) followed by an MLP head—softly ranks the reference sequences by their final alignment scores and aggregates the top-K alignment scores into class-level logits, corresponding to the reference selection and voting step of the DTW–KNN algorithm.

Invariants are imposed at the inputs and outputs of each block to preserve algorithmic fidelity and interpretability: (i) distance measurement preserves the commutative property; (ii) the dynamic programmer maintains cumulative cost evolution ensuring optimal substructure; (iii) score aggregation preserves class-level interpretability through explicit, focused score decomposition.

Training Process. All three neural blocks are trained jointly in an end-to-end manner, allowing alignment and classification to co-adapt without intermediate supervision. The *Cost Estimator* and *Dynamic Programmer* learn alignment structures guided solely by the final task loss. Training uses a composite objective: cross-entropy between predicted and ground-truth labels plus a regularization term on the Cost Estimator's distance computation to stabilize similarity learning and reduce overfitting.

Result—Training Efficiency. Figure 5(a) compares AINN and the baseline model [62] on the GSC dataset [64] across different training set sizes. All models are trained on a cluster node with Intel Xeon Platinum 8470 and NVIDIA H100 GPU. AINN achieves up to 7.75% higher accuracy using only half of the training data while maintaining a 46.16× smaller model and real-time inference (0.14

s) on a Google Pixel 6a. The results demonstrate that algorithm-informed initialization and structural constraints lead to more data-efficient learning without sacrificing performance.

Result—Debuggability. We evaluate AINN's debuggability by analyzing whether block-level behaviors can reveal the source of prediction errors. To make such failures observable, we train the AINN for fewer iterations, achieving a lower accuracy of 64.38%.

Figure 5 (b) illustrates the *Dynamic Programmer*'s learned soft-min relation between its predicted update and the true minimum of neighboring DP cells. The two remain highly correlated along the diagonal during normal execution, while deviations mark faulty or unstable updates, allowing detection of numerical or structural errors within this block.

To localize errors among the other neural blocks, we attach light-weight encoders to the *Cost Estimator* and *Score Aggregator* to monitor hidden activations. Each encoder projects its block's activity into a 2D feature space; as shown in Figure 5 (c), normal and anomalous executions form clusters. A linear boundary classifies anomalous executions with 77.38% accuracy. The boundary's weighted components further indicate which block contributes most to an error, enabling AINN to identify the faulty module without ground-truth supervision and supporting interpretable, block-level debugging.

6 DISCUSSION

Algorithmic Blueprint. A well-chosen blueprint can improve training efficiency by injecting task-specific rules that general networks struggle to learn from data alone. Gains arise when blocks are aligned with specific algorithmic roles, which narrows the hypothesis space and introduces strong inductive bias. Conversely, a misleading algorithm or an overly broad/mis-specified logic block will limit performance and negate training-efficiency advantages.

Classic Algorithm Instead of AINN. Classical algorithms are rigid: they work well when their assumptions hold but often underperform on noisy sensor data. AINNs add learnable capacity where rules are useful yet inexact, while the algorithmic scaffold preserves structure, interpretability, and block-level debugging. As a seminal vision, the AINNs described in this paper convert known algorithms to demonstrate gains in training efficiency and debuggability, establishing design principles before tackling harder problems with no exact algorithm or with NP-hard structure (e.g., approximation under violated assumptions).

Granularity of Error Localization. The AINNs presented here do not target fine-grained (e.g., neuron-level) error localization. Instead, they elevate explanation to algorithm-mapped neural blocks, enabling block-level tracing and fault localization—capabilities that a single end-to-end network does not provide. Advancing toward finer-grained localization within and across blocks, while preserving algorithmic interpretability, is an important future direction.

7 RELATED WORK

PINNs and Neuro-Symbolic Systems. Like PINNs, AINN blocks can incorporate constraint terms, but AINNs go further by encoding *discrete algorithmic invariants* (not only PDE residuals) and executing within a *modular, typed scaffold* that supports step-wise tracing and fault localization. Unlike neuro-symbolic hybrids that couple symbolic reasoning with neural modules, AINNs are fully neural yet guided by classical algorithms as inductive blueprints; bottom-up substitution can yield hybrids where some components remain algorithmic for efficiency or reliability, complementing and extending neuro-symbolic approaches. AINNs also differ from prior modular or concept-bottleneck designs that introduce structure without aligning to an algorithmic trace [65], and from neural algorithmic reasoning [66] or differentiable operators [67], which emphasize learned operators but provide limited visibility into intermediate computations. Finally, AINNs employ *block-level monitors*, enabling automatic fault detection and localization without additional intermediate labels.

Algorithmic Reasoning and Interpretability. Work in explainable AI [29, 68] improves visualization and interpretation of predictions but lacks logic-based, step-wise understanding akin to algorithms. Neuro-symbolic systems [36, 69] integrate logic with learning yet face scalability and sequential-reasoning challenges. Neural algorithmic reasoning [70, 71] simulates algorithmic behavior but offers limited flexible traceability. Differentiable programming [72] enables gradients through programmatic structures but struggles with mutable state and control flow. Function-based interpretability [73] provides mathematical transparency without algorithmic grounding. These limitations motivate embedding algorithmic logic within neural networks for structured, traceable, and data-efficient reasoning.

Subnetwork Debugging and Anomaly Detection. Tools like TensorBoard [74] and DeepVis [75] offer global insights but lack granularity for subnetwork anomaly localization. Subnetwork selection methods—from pruning [76, 77] to quantum [78] and policy-based approaches [79]—target optimization rather than anomaly detection. Cost-efficient fusion and block selection for transfer learning or on-device inference [80–82] do not surface latent anomalies. OOD detection [83–87] flags abnormal inputs but not irregular activations inside blocks. Rule extraction [88, 89] and symbolic regression [90, 91] provide structural understanding yet struggle to scale or reverse-engineer hidden logic—highlighting a gap in subnetwork-level anomaly detection and interpretability.

8 CONCLUSION AND FUTURE DIRECTIONS

This paper presents *Algorithm-Informed Neural Networks* (AINNs) as a seminal framework that embeds algorithmic structure within neural models to achieve interpretable, modular, and debuggable learning. We addressed two core challenges—construction and debugging—and demonstrated AINNs for two sensing problems, showing how algorithmic blueprints narrow hypothesis space, enable block-level monitoring, and support fault localization. While these results establish the viability of AINNs, they also expose key limitations and opportunities for improving reliability, scalability, sample/data efficiency, verifiability, and deployability. The open-source implementation of AINN is available at [92].

Open Problems. Many open problems remain and call for new research to make AINNs *more reliable, scalable, efficient, verifiable, and broadly applicable*: (i) *algorithmic foundations & theory*—formalize logic blocks, select decomposition granularity, and map control flow (loops, recursion, conditionals) to neural counterparts with

provable correctness and traceability; (ii) *automation & compilers*—automate block discovery and translation via compiler analyses, program synthesis, and LLM-assisted tooling; devise compiler-inspired methods to reverse-engineer learned blocks into reusable components; (iii) *anomaly detection & debugging*—improve detection accuracy with scalable subnetwork selection, principled statistics, and OOD reasoning; build tools that surface irregular within-block activations and support iterative diagnosis; (iv) *invariant extraction & verification*—derive block-level invariants from specifications; use symbolic regression and logic-based verification for runtime monitoring and formal guarantees; (v) *scalability, edge, & generalizability*—maintain algorithmic structure at scale for LLMs and generative systems [93–116]; identify practicality thresholds and exploit parallel/distributed training, accelerators, block merging, and NAS for lightweight edge deployments; (vi) *causal and hybrid reasoning*—formalize a taxonomy mapping algorithmic primitives to neural analogs to standardize block decomposition; embed causal and counterfactual reasoning in AINNs for proactive fault prevention and stronger interpretability; (vii) *applications*—pursue domains where interpretability and traceability are essential (drug discovery [117], healthcare decision-making [118], finance [119], robotics [120], intelligent transportation [121]) to stress-test AINNs and refine design principles.

ACKNOWLEDGMENT

This work was supported, in part, by grants NSF EAGER NAIRR Pilot 2503073 (PI Nirjon) and 2503074 (Co-PI Uddin) and NSF CAREER 2047461 (PI Nirjon).

REFERENCES

[1] David L Donoho. Compressed sensing. *IEEE Transactions on information theory*, 52(4):1289–1306, 2006.
[2] Shi Liu. A survey of dynamic voltage and frequency scaling for high-performance low-power systems. 2024.
[3] Greg Welch, Gary Bishop, et al. An introduction to the kalman filter. 1995.
[4] Andrei Broder and Michael Mitzenmacher. Network applications of bloom filters: A survey. *Internet mathematics*, 1(4):485–509, 2004.
[5] Kiam Heong Ang, Gregory Chong, and Yun Li. Pid control system analysis, design, and technology. *IEEE transactions on control systems technology*, 13(4):559–576, 2005.
[6] Joseph Polastre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 95–107, 2004.
[7] Michael Buettner, Gary V Yee, Eric Anderson, and Richard Han. X-mac: a short preamble mac protocol for duty-cycled wireless sensor networks. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 307–320, 2006.
[8] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 2002.
[9] Zheng Yang and Yunhao Liu. Quality of trilateration: Confidence-based iterative localization. *IEEE Transactions on parallel and distributed systems*, 21(5):631–640, 2009.
[10] Neal Patwari, Joshua N Ash, Spyros Kyperountas, Alfred O Hero, Randolph L Moses, and Neiyer S Correal. Locating the nodes: cooperative localization in wireless sensor networks. *IEEE Signal processing magazine*, 22(4):54–69, 2005.
[11] Fan Xiangning and Song Yulin. Improvement on leach protocol of wireless sensor network. In *2007 international conference on sensor technologies and applications (SENSORCOMM 2007)*, pages 260–264. IEEE, 2007.
[12] Joanna Kulik, Wendi Heinzelman, and Hari Balakrishnan. Negotiation-based protocols for disseminating information in wireless sensor networks. *Wireless networks*, 8:169–185, 2002.
[13] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 56–67, 2000.
[14] Miklós Maróti, Branislav Kusy, Gyula Simon, and Akos Lédeczi. The flooding time synchronization protocol. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 39–49, 2004.
[15] Saurabh Ganeriwal, Ram Kumar, and Mani B Srivastava. Timing-sync protocol for sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 138–149, 2003.
[16] Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.
[17] Tushar D Chandra, Robert Griesemer, and Joshua Redstone. Paxos made live: an engineering perspective. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 398–407, 2007.
[18] Heidi Howard, Malte Schwarzkopf, Anil Madhavapeddy, and Jon Crowcroft. Raft refloated: Do we have consensus? *ACM SIGOPS Operating Systems Review*, 49(1):12–21, 2015.
[19] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX annual technical conference (USENIX ATC 14)*, pages 305–319, 2014.
[20] Hoang Truong, Shuo Zhang, Ufuk Muncuk, Phuc Nguyen, Nam Bui, Anh Nguyen, Qin Lv, Kaushik Chowdhury, Thang Dinh, and Tam Vu. Capband: Battery-free successive capacitance sensing wristband for hand gesture recognition. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, pages 54–67, 2018.
[21] Viet Nguyen, Siddharth Rupavatharam, Luyang Liu, Richard Howard, and Marco Gruteser. Handsense: capacitive coupling-based dynamic, micro finger gesture recognition. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, pages 285–297, 2019.
[22] Nicholas D Lane, Petko Georgiev, and Lorena Qendro. Deepear: robust smartphone audio sensing in unconstrained acoustic environments using deep learning. In *Proceedings of the 2015 ACM international joint conference on pervasive and ubiquitous computing*, pages 283–294, 2015.
[23] Petko Georgiev, Sourav Bhattacharya, Nicholas D Lane, and Cecilia Mascolo. Low-resource multi-task audio sensing for mobile and embedded devices via shared deep neural network representations. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 1(3):1–19, 2017.
[24] Shuochao Yao, Shaohan Hu, Yiran Zhao, Aston Zhang, and Tarek Abdelzaher. Deepsense: A unified deep learning framework for time-series mobile sensing data processing. In *Proceedings of the 26th international conference on world wide web*, pages 351–360, 2017.
[25] Miaomiao Liu, Sikai Yang, Wyssanie Chomsin, and Wan Du. Real-time tracking of smartwatch orientation and location by multitask learning. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems*, pages 120–133, 2022.
[26] Thomas A Henzinger and Joseph Sifakis. The embedded systems design challenge. In *International Symposium on Formal Methods*, pages 1–15. Springer, 2006.
[27] Zachary C Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, 2018.
[28] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.
[29] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.
[30] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR, 2017.
[31] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.
[32] Pieter-Jan Kindermans, Sara Hooker, Julius Adebayo, Maximilian Alber, Kristof T Schütt, Sven Dähne, Dumitru Erhan, and Been Kim. The (un) reliability of saliency methods. *Explainable AI: Interpreting, explaining and visualizing deep learning*, pages 267–280, 2019.
[33] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. *Advances in neural information processing systems*, 31, 2018.
[34] Dylan Slack, Sophie Hilgard, Emily Jia, Sameer Singh, and Himabindu Lakkaraju. Fooling lime and shap: Adversarial attacks on post hoc explanation methods. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, pages 180–186, 2020.
[35] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.
[36] Efthymia Tsamoura, Timothy Hospedales, and Loizos Michael. Neural-symbolic integration: A compositional perspective. In *Proceedings of the AAAI conference*

on artificial intelligence, volume 35, pages 5051–5060, 2021.

[37] Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *Proceedings of the 3rd international conference on knowledge discovery and data mining*, pages 359–370, 1994.

[38] Oliver Kramer. K-nearest neighbors. In *Dimensionality reduction with unsupervised nearest neighbors*, pages 13–23. Springer, 2013.

[39] Donald Ervin Knuth. *The art of computer programming: sorting and searching*, volume 3. Pearson Education, 1998.

[40] Sebastian Prillo and Julian Eisenschlos. Softsort: A continuous relaxation for the argsort operator. In *International Conference on Machine Learning*, pages 7793–7802. PMLR, 2020.

[41] Gonzalo Mena, Erdem Varol, Amin Nejatbakhsh, Eviatar Yemini, and Liam Paninski. Sinkhorn permutation variational marginal inference. In *Symposium on advances in approximate Bayesian inference*, pages 1–9. PMLR, 2020.

[42] Ryan Prescott Adams and Richard S Zemel. Ranking via sinkhorn propagation. *arXiv preprint arXiv:1106.1925*, 2011.

[43] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. ieee, 1995.

[44] Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation*, 1(1):1–23, 1993.

[45] Florenc Demrozi, Graziano Pravadelli, Azra Bihorac, and Parisa Rashidi. Human activity recognition using inertial, physiological and environmental sensors: A comprehensive survey. *IEEE access*, 8:210816–210836, 2020.

[46] Habba'S Ngodjou Doukaga, Noro Haritiana Louisiane Randrianarivelo Rakotoarson, Gabriel Aubin-Morneau, Pascal Fortin, Julien Maitre, and Bruno Bouchard. Fine-grained human activity recognition in smart homes through photoplethysmography-based hand gesture detection. In *Proceedings of the 17th International Conference on PErvasive Technologies Related to Assistive Environments*, pages 163–168, 2024.

[47] Tao Liu, Mingyang Zhang, Zhihao Li, Hanjie Dou, Wangyang Zhang, Jiaqian Yang, Pengfan Wu, Dongxiao Li, and Xiaojing Mu. Machine learning-assisted wearable sensing systems for speech recognition and interaction. *Nature Communications*, 16(1):2363, 2025.

[48] Xue Wang, Zixiong Su, Jun Rekimoto, and Yang Zhang. Watch your mouth: Silent speech recognition with depth sensing. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, pages 1–15, 2024.

[49] Ivana Kiprijanovska, Simon Stankoski, M John Broulidakis, James Archer, Mohsen Fatoorechi, Martin Gjoreski, Charles Nduka, and Hristijan Gjoreski. Towards smart glasses for facial expression recognition using omg and machine learning. *Scientific Reports*, 13(1):16043, 2023.

[50] Hymalai Bello, Luis Alfredo Sanchez Marin, Sungho Suh, Bo Zhou, and Paul Lukowicz. Inmyface: Inertial and mechanomyography-based sensor fusion for wearable facial activity recognition. *Information Fusion*, 99:101886, 2023.

[51] Jesse Phipps, Bryant Passage, Kaan Sel, Jonathan Martinez, Milad Saadat, Teddy Koker, Natalie Damaso, Shakti Davis, Jeffrey Palmer, Kajal Claypool, et al. Early adverse physiological event detection using commercial wearables: challenges and opportunities. *NPJ Digital Medicine*, 7(1):136, 2024.

[52] Natalia Bartczak, Marta Glanowska, Karolina Kowalewicz, Maciej Kunin, and Robert Susik. Fall detection based on recurrent neural networks and accelerometer data from smartphones. *Applied Sciences*, 15(12):6688, 2025.

[53] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[54] Hee-Seon Park and Seong-Whan Lee. Off-line recognition of large-set handwritten characters with multiple hidden markov models. *Pattern Recognition*, 29(2):231–244, 1996.

[55] Kui Liu, Chen Chen, Roozbeh Jafari, and Nasser Kehtarnavaz. Multi-hmm classification for hand gesture recognition using two differing modality sensors. In *2014 IEEE Dallas Circuits and Systems Conference (DCAS)*, pages 1–4. IEEE, 2014.

[56] Maxime Kawawa-Beaudan, Srijan Sood, Soham Palande, Ganapathy Mani, Tucker Balch, and Manuela Veloso. Ensemble methods for sequence classification with hidden markov models. *arXiv preprint arXiv:2409.07619*, 2024.

[57] Eduardo Casilari, Jose A Santoyo-Ramón, and Jose M Cano-García. Umafall: A multisensor dataset for the research on automatic fall detection. *Procedia Computer Science*, 110:32–39, 2017.

[58] Harikrishna Narasimhan and Shivani Agarwal. On the relationship between binary classification, bipartite ranking, and binary class probability estimation. *Advances in neural information processing systems*, 26, 2013.

[59] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

[60] Sercan O Arik, Markus Kliegl, Rewon Child, Joel Hestness, Andrew Gibiansky, Chris Fougner, Ryan Prenger, and Adam Coates. Convolutional recurrent neural networks for small-footprint keyword spotting. *arXiv preprint arXiv:1703.05390*, 2017.

[61] Justin Salamon and Juan Pablo Bello. Deep convolutional neural networks and data augmentation for environmental sound classification. *IEEE Signal processing letters*, 24(3):279–283, 2017.

[62] Cong Shi, Tianfang Zhang, Zhuohang Li, Huy Phan, Tianming Zhao, Yan Wang, Jian Liu, Bo Yuan, and Yingying Chen. Audio-domain position-independent backdoor attack via unnoticeable triggers. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, pages 583–595, 2022.

[63] Mathieu Blondel, Olivier Teboul, Quentin Berthet, and Josip Djolonga. Fast differentiable sorting and ranking. In *International Conference on Machine Learning*, pages 950–959. PMLR, 2020.

[64] P. Warden. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *ArXiv e-prints*, April 2018.

[65] Ribana Roscher, Bastian Bohn, Marco F Duarte, and Jochen Garcke. Explainable machine learning for scientific insights and discoveries. *Ieee Access*, 8:42200–42216, 2020.

[66] Petar Veličković and Charles Blundell. Neural algorithmic reasoning. *Patterns*, 2(7), 2021.

[67] Marco Cuturi, Olivier Teboul, and Jean-Philippe Vert. Differentiable ranking and sorting using optimal transport. *Advances in neural information processing systems*, 32, 2019.

[68] Aditya Chattopadhay, Anirban Sarkar, Prantik Howlader, and Vineeth N Balasubramanian. Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks. In *2018 IEEE winter conference on applications of computer vision (WACV)*, pages 839–847. IEEE, 2018.

[69] Ivan Donadello, Luciano Serafini, and Artur D'Avila Garcez. Logic tensor networks for semantic image interpretation. *arXiv preprint arXiv:1705.08968*, 2017.

[70] Petar Veličković, Adrià Puigdomènech Badia, David Budden, Razvan Pascanu, Andrea Banino, Misha Dashevskiy, Raia Hadsell, and Charles Blundell. The clrs algorithmic reasoning benchmark. In *International Conference on Machine Learning*, pages 22084–22102. PMLR, 2022.

[71] Beatrice Bevilacqua, Kyriacos Nikiforou, Borja Ibarz, Ioana Bica, Michela Paganini, Charles Blundell, Jovana Mitrovic, and Petar Veličković. Neural algorithmic reasoning with causal regularisation. In *International Conference on Machine Learning*, pages 2272–2288. PMLR, 2023.

[72] Zygote. https://fluxml.ai/Zygote.jl/stable/.

[73] Seulki Lee and Shahriar Nirjon. Deep functional network (dfn) functional interpretation of deep neural networks for intelligent sensing systems. In *Proceedings of the 20th International Conference on Information Processing in Sensor Networks (co-located with CPS-IoT Week 2021)*, pages 191–206, 2021.

[74] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[75] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

[76] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[77] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

[78] Tim Whitaker. Quantum neuron selection: finding high performing subnetworks with quantum algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 2258–2264, 2022.

[79] Beatrice Bevilacqua, Moshe Eliasof, Eli Meirom, Bruno Ribeiro, and Haggai Maron. Efficient subgraph gnns by learning effective selection policies. *arXiv preprint arXiv:2310.20082*, 2023.

[80] Md Adnan Arefeen, Sumaiya Tabassum Nimi, Md Yusuf Sarwar Uddin, and Yugyung Lee. Transjury: Towards explainable transfer learning through selection of layers from deep neural networks. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 978–984. IEEE, 2021.

[81] Md Adnan Arefeen, Sumaiya Tabassum Nimi, Md Yusuf Sarwar Uddin, and Zhu Li. A lightweight relu-based feature fusion for aerial scene classification. In *2021 IEEE International Conference on Image Processing (ICIP)*, pages 3857–3861. IEEE, 2021.

[82] Sumaiya Tabassum Nimi, Md Adnan Arefeen, Md Yusuf Sarwar Uddin, Biplob Debnath, and Srimat Chakradhar. Factionformer: Context-driven collaborative vision transformer models for edge intelligence. In *2023 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 349–354. IEEE, 2023.

[83] Jinwon An and Sungzoon Cho. Variational autoencoder based anomaly detection using reconstruction probability. *Special lecture on IE*, 2(1):1–18, 2015.

[84] Zhaomin Chen, Chai Kiat Yeo, Bu Sung Lee, and Chiew Tong Lau. Autoencoder-based network anomaly detection. In *2018 Wireless telecommunications symposium (WTS)*, pages 1–5. IEEE, 2018.

[85] Alberto Caron, Chris Hicks, and Vasilios Mavroudis. A view on out-of-distribution identification from a statistical testing theory perspective. *arXiv preprint arXiv:2405.03052*, 2024.

[86] Laura O'Mahony, David JP O'Sullivan, and Nikola S Nikolov. On the detection of anomalous or out-of-distribution data in vision models using statistical techniques. In *The International Conference on Artificial Intelligence and Computer Vision*, pages 426–435. Springer, 2023.

[87] Sumaiya Tabassum Nimi, Adnan Arefeen, Yusuf Sarwar Uddin, and Yugyung Lee. Earlin: Early out-of-distribution detection for resource-efficient collaborative inference. In *Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part I 21*, pages 635–651. Springer, 2021.

[88] Congjie He, Meng Ma, and Ping Wang. Extract interpretability-accuracy balanced rules from artificial neural networks: A review. *Neurocomputing*, 387:346–358, 2020.

[89] Yu Chen, Tianyu Cui, Alexander Capstick, Nan Fletcher-Loyd, and Payam Barnaghi. Enabling regional explainability by automatic and model-agnostic rule extraction. *arXiv preprint arXiv:2406.17885*, 2024.

[90] Charles Fox, Neil D Tran, F Nikki Nacion, Samiha Sharlin, and Tyler R Josephson. Incorporating background knowledge in symbolic regression using a computer algebra system. *Machine Learning: Science and Technology*, 5(2):025057, 2024.

[91] Miles Cranmer. Interpretable machine learning for science with pysr and symbolicregression. jl. *arXiv preprint arXiv:2305.01582*, 2023.

[92] Ainn implementation. https://github.com/anonymoussubfile/sensys-ainn.

[93] Nikumbh Sarthak Sham, Sandip Chakraborty, and Shamik Sural. Generation of optimized solidity code for machine learning models using llms. *arXiv preprint arXiv:2503.06203*, 2025.

[94] Xuechen Zhang, Zijian Huang, Ege Onur Taga, Carlee Joe-Wong, Samet Oymak, and Jiasi Chen. Efficient contextual llm cascades through budget-constrained policy learning. *arXiv preprint arXiv:2404.13082*, 2024.

[95] Fanhang Man, Huandong Wang, Jianjie Fang, Zhaoyi Deng, Baining Zhao, Xinlei Chen, and Yong Li. Context-aware sentiment forecasting via llm-based multi-perspective role-playing agents. *arXiv preprint arXiv:2505.24331*, 2025.

[96] Mahmoud Srewa, Tianyu Zhao, and Salma Elmalaki. Pluralllm: pluralistic alignment in llms via federated learning. In *Proceedings of the 3rd International Workshop on Human-Centered Sensing, Modeling, and Intelligent Systems*, pages 64–69, 2025.

[97] Kevin Post, Reo Kuchida, Mayowa Olapade, Zhigang Yin, Petteri Nurmi, and Huber Flores. Contextllm: Meaningful context reasoning from multi-sensor and multi-device data using llms. In *Proceedings of ACM HOTMOBILE'25*. Association for Computing Machinery (ACM), 2025.

[98] Ayushi Mishra, Yang Bai, Priyadarshan Narayanasamy, Nakul Garg, and Nirupam Roy. Spatial audio processing with large language model on wearable devices. *arXiv preprint arXiv:2504.08907*, 2025.

[99] Yanming Xiu and Maria Gorlatova. Vision language model-based solution for obstruction attack in ar: A meta quest 3 implementation. In *2025 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pages 1638–1639. IEEE, 2025.

[100] Jiawei Hu, Hong Jia, Mahbub Hassan, Lina Yao, Brano Kusy, and Wen Hu. Lightllm: A versatile large language model for predictive light sensing. In *Proceedings of the 23rd ACM Conference on Embedded Networked Sensor Systems*, pages 158–171, 2025.

[101] Tanmay Srivastava, Prerna Khanna, Shijia Pan, Phuc Nguyen, and Shubham Jain. Unvoiced: Designing an llm-assisted unvoiced user interface using earables. In *Proceedings of the 22nd ACM Conference on Embedded Networked Sensor Systems*, pages 784–798, 2024.

[102] Sijie Ji, Xinzhe Zheng, Wei Gao, and Mani Srivastava. Transforming mental health care with autonomous llm agents at the edge. In *Proceedings of the 23rd ACM Conference on Embedded Networked Sensor Systems*, pages 692–693, 2025.

[103] Lilin Xu, Kaiyuan Hou, and Xiaofan Jiang. Exploring the capabilities of llms for imu-based fine-grained human activity understanding. In *Proceedings of the 2nd International Workshop on Foundation Models for Cyber-Physical Systems & Internet of Things*, pages 13–18, 2025.

[104] Heming Fu, Hongkai Chen, Shan Lin, and Guoliang Xing. Shade-ad: An llm-based framework for synthesizing activity data of alzheimer's patients. In *Proceedings of the 23rd ACM Conference on Embedded Networked Sensor Systems*, pages 290–296, 2025.

[105] Hung Manh Pham, Jialu Tang, Aaqib Saeed, and Dong Ma. Q-heart: Ecg question answering via knowledge-informed multimodal llms. *arXiv preprint arXiv:2505.06296*, 2025.

[106] Ziyan An, Xia Wang, Hendrik Baier, Zirong Chen, Abhishek Dubey, Taylor T Johnson, Jonathan Sprinkle, Ayan Mukhopadhyay, and Meiyi Ma. Combining llms with logic-based framework to explain mcts. *arXiv preprint arXiv:2505.00610*, 2025.

[107] Joykirat Singh, Akshay Nambi, and Vibhav Vineet. Exposing the achilles' heel: Evaluating llms ability to handle mistakes in mathematical reasoning. *arXiv preprint arXiv:2406.10834*, 2024.

[108] Yuan Sun, Navid Salami Pargoo, Peter Jin, and Jorge Ortiz. Optimizing autonomous driving for safety: A human-centric approach with llm-enhanced rlhf. In *Companion of the 2024 on ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 76–80, 2024.

[109] Xiaomin Ouyang and Mani Srivastava. Llmsense: Harnessing llms for high-level reasoning over spatiotemporal sensor traces. In *2024 IEEE 3rd Workshop on Machine Learning on Edge in Sensor Systems (SenSys-ML)*, pages 9–14. IEEE, 2024.

[110] Wen Cheng, Ke Sun, Xinyu Zhang, and Wei Wang. Security attacks on llm-based code completion tools. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 23669–23677, 2025.

[111] Nasir Hussain, Haohan Chen, Chanh Tran, Philip Huang, Zhuohao Li, Pravir Chugh, William Chen, Ashish Kundu, and Yuan Tian. Vulbinllm: Llm-powered vulnerability detection for stripped binaries. *arXiv preprint arXiv:2505.22010*, 2025.

[112] Pramuka Medaranga, Dhairya Shah, Savitha Viswanadh Kandala, and Ambuj Varshney. Poster: Simplifying the networking of wireless embedded systems using a large language model. In *Proceedings of the ACM SIGCOMM 2024 Conference: Posters and Demos*, pages 78–80, 2024.

[113] Xiaofan Yu, Lanxiang Hu, Benjamin Reichman, Rushil Chandrupatla, Dylan Chu, Xiyuan Zhang, Larry Heck, and Tajana S Rosing. A real time question answering system for multimodal sensors using llms. In *Proceedings of the 22nd ACM Conference on Embedded Networked Sensor Systems*, pages 836–837, 2024.

[114] Che Liu, Zhongwei Wan, Yuqi Wang, Hui Shen, Haozhe Wang, Kangyu Zheng, Mi Zhang, and Rossella Arcucci. Benchmarking and boosting radiology report generation for 3d high-resolution medical images. *arXiv preprint arXiv:2406.07146*, 2024.

[115] Yuting Huang, Leilei Ding, Zhipeng Tang, Tianfu Wang, Xinrui Lin, Wuyang Zhang, Mingxiao Ma, and Yanyong Zhang. A framework for benchmarking and aligning task-planning safety in llm-based embodied agents. *arXiv preprint arXiv:2504.14650*, 2025.

[116] Yinzhou Wang, Yimeng Wang, Ye Xiao, Liabette Escamilla, Bianca Augustine, Kelly Crace, Gang Zhou, and Yixuan Zhang. Evaluating an llm-powered chatbot for cognitive restructuring: Insights from mental health professionals. *arXiv preprint arXiv:2501.15599*, 2025.

[117] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *nature*, 596(7873):583–589, 2021.

[118] Andre Esteva, Alexandre Robicquet, Bharath Ramsundar, Volodymyr Kuleshov, Mark DePristo, Katherine Chou, Claire Cui, Greg Corrado, Sebastian Thrun, and Jeff Dean. A guide to deep learning in healthcare. *Nature medicine*, 25(1):24–29, 2019.

[119] Longbing Cao. Ai in finance: challenges, techniques, and opportunities. *ACM Computing Surveys (CSUR)*, 55(3):1–38, 2022.

[120] Sule Anjomshoae, Amro Najjar, Davide Calvaresi, and Kary Främling. Explainable agents and robots: Results from a systematic literature review. In *18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019*, pages 1078–1088. International Foundation for Autonomous Agents and Multiagent Systems, 2019.

[121] Jesse Levinson, Jake Askeland, Jan Becker, Jennifer Dolson, David Held, Soeren Kammel, J Zico Kolter, Dirk Langer, Oliver Pink, Vaughan Pratt, et al. Towards fully autonomous driving: Systems and algorithms. In *2011 IEEE intelligent vehicles symposium (IV)*, pages 163–168. IEEE, 2011.