SALCA-IB: Self-Adaptive LLM-Driven Continuous Learning Agent for IB Network Failure Prediction

***Abstract*—The emergence of transformer-based large language models has led to unprecedented demands on high-performance computing infrastructure, where InfiniBand (IB) networks play a vital role for their superior low-latency and high-bandwidth communication capabilities. Network failures in these environments can cause significant training interruptions or restarts, resulting in substantial computational resource waste and training cost overhead. However, effective failure prediction remains challenging due to both the scarcity of failure data and the susceptibility of network features to environmental changes. This paper introduces SALCA-IB (Self-Adaptive LLM-Driven Continuous Learning Agent for IB Network Failure Prediction), an innovative failure prediction system that leverages Large Language Models (LLMs) as its planning core. The system's key innovations include: (1) LLM-driven autonomous data selection and model optimization; (2) A fusion memory system integrating short-term and long-term memory; and (3) LLM-supported automatic evaluation feedback and closed-loop optimization. Experimental results on a production cluster with 2048 computing cards show that SALCA-IB improves prediction F1@K-score by 5.1% in static scenarios and demonstrates a 12.4% increase when facing changes in network feature distributions, significantly enhancing the predictability of large-scale computing infrastructure.**

***Index Terms*—InfiniBand Network, Large Language Model, Autonomous Agent**

I. Introduction

The rapid advancement of transformer-based language models has fundamentally transformed the landscape of artificial intelligence, leading to unprecedented demands on computing infrastructure. Training these increasingly large models, often comprising hundreds of billions of parameters, requires massive distributed computing resources with high-performance interconnects. In such environments, InfiniBand (IB) networks have become the de facto standard for their exceptional performance in low-latency, high-bandwidth communication, serving as the critical backbone for distributed training tasks.

Network failures in distributed training environments can have severe consequences. When IB network failures occur during model training, they often cause unexpected training interruptions or complete restarts, leading to significant computational resource waste and increased operational costs. For instance, recent studies have shown that network-related failures can account for up to 30% of training interruptions in large-scale AI clusters, with each incident potentially wasting hundreds of GPU hours (He et al. [6]). This challenge becomes particularly acute as models grow larger and training runs extend to weeks or even months.

Despite its critical importance, effective IB network failure prediction faces several significant challenges. First, failure data in production environments is inherently scarce, as failures are relatively rare events in well-maintained systems, making it difficult to build robust prediction models using traditional machine learning approaches. Second, network feature distributions are highly dynamic and susceptible to various external factors (Liu et al. [8]), such as environmental conditions, hardware aging, and maintenance activities. These challenges are further compounded by the increasing scale and complexity of modern computing infrastructures.

Traditional approaches to network failure prediction primarily rely on static machine learning models or rule-based systems (Mohammed et al. [9], Nie et al. [10]). While these methods have shown some success in controlled environments, they struggle to maintain performance in real-world scenarios where network characteristics evolve continuously. Moreover, existing solutions often operate as black boxes, providing limited interpretability and failing to leverage historical experience effectively for continuous improvement (Das et al. [5]). Recent advances in deep learning-based approaches have attempted to address these limitations, but they still face challenges in adapting to dynamic environments and maintaining long-term prediction accuracy (Alharthi et al. [2]).

The emergence of Large Language Models (LLMs) presents new opportunities for addressing these challenges. LLMs have demonstrated remarkable capabilities in complex reasoning and planning tasks (Ahmed et al. [1], Su et al. [12]), suggesting their potential for orchestrating adaptive prediction systems. Additionally, recent advances in memory systems and continuous learning architectures (Zhang et al. [13]) have shown promise in handling dynamic environments, though their application to IB network failure prediction remains largely unexplored.

To address these challenges, we propose SALCA-IB (Self-Adaptive LLM-Driven Continuous Learning Agent for IB Network Failure Prediction), an innovative system that combines the reasoning capabilities of LLMs with traditional machine learning models in a unified, adaptive framework. To tackle the data scarcity challenge, SALCA-IB employs an LLM-driven planning core that intelligently orchestrates data selection and model optimization. To handle dynamic feature distributions, we design a dual-memory system that integrates both short-term and long-term experiences. Furthermore, to ensure sustained prediction accuracy, we implement a continuous learning mechanism with LLM-supported feedback loops that enables real-time adaptation to changing network conditions.

The main contributions of this paper are threefold:

- We propose an innovative LLM-driven agent architecture

for IB network failure prediction that uniquely leverages LLM as a high-level planning core to orchestrate model selection, parameter optimization, and continuous learning strategies, while employing traditional machine learning models as efficient executors for real-time prediction tasks.
- We design a novel dual-memory fusion system that seamlessly integrates short-term and long-term memory mechanisms, enabling rapid adaptation to dynamic network changes while preserving and leveraging valuable historical knowledge for enhanced prediction robustness.
- We conduct comprehensive experiments on a large-scale production cluster with 2048 computing cards, demonstrating that SALCA-IB improves prediction F1@K-score by 5.1% under static conditions and achieves a 12.4% increase when facing network feature distribution changes. Through extensive ablation studies, we validate the substantial contributions of both the LLM-driven framework and the dual-memory system components.

The remainder of this paper is organized as follows: Section II reviews related work in HPC failure prediction and LLM applications. Section III details the design and implementation of SALCA-IB. Section IV presents our experimental setup and results. Finally, Section V concludes the paper and discusses future work.

II. Related Work

A. HPC Clusters Failure Prediction

Failure prediction in HPC clusters, particularly for large-scale GPU training workloads, has been extensively studied due to its critical importance in maintaining system reliability. Traditional approaches primarily rely on statistical methods and machine learning models. He et al. [6] analyzed hardware failures in deep learning training systems and found that GPU failures significantly impact training progress and model convergence. Similarly, Liu et al. [8] focused specifically on predicting GPU failures under intensive deep learning workloads, highlighting the unique challenges of monitoring and predicting failures in GPU-heavy computing environments.

More recent work has attempted to address these challenges through ensemble methods and deep learning approaches. Nie et al. [10] introduced GPU error prediction models using temporal and spatial dependencies to improve prediction robustness under limited data conditions, which shares similarities with our model pool concept but lacks intelligent orchestration. Das et al. [5] explored LSTM-based deep learning techniques to predict system health and lead times to failure, conceptually related to our long-term memory mechanism but without continuous adaptation capabilities. Liu et al. [8] further advanced GPU failure prediction using high-precision methods under deep learning workloads, demonstrating improved accuracy but still facing challenges in dynamic environments. Despite these advances, as highlighted by Alharthi et al. [2], existing methods still face significant challenges in handling dynamic network environments and maintaining long-term prediction accuracy.

B. LLM-based Reasoning, Memory and Self-Refinement

Recent advances in LLM capabilities have opened new possibilities for complex reasoning and decision-making tasks. These developments can be examined across three key aspects:

1) Reasoning and Verification: LLMs have demonstrated remarkable capabilities in complex reasoning, though with certain limitations. Chen et al. [18] established the effectiveness of chain-of-thought prompting for enhancing reasoning abilities, while Wang et al. [19] advanced this further through a self-consistency framework that validates multiple reasoning paths. However, Tyen et al. [16] revealed a critical limitation: while LLMs excel at correction with guidance, they struggle with autonomous error detection. Weng et al. [15] addressed this challenge by developing self-verification mechanisms that significantly improve reasoning reliability.

2) Memory Integration: The integration of external memory mechanisms has emerged as a crucial enhancement to LLM capabilities. Krishnamurthy et al. [20] pioneered a hierarchical memory architecture that effectively manages both short-term and long-term knowledge retention. This approach was complemented by Li et al. [21], who developed an adaptive memory pruning mechanism that optimizes memory efficiency while preserving critical information. For domain-specific applications, Zhang et al. [22] demonstrated successful integration of external knowledge bases with LLMs, while Shridhar et al. [17] proposed the ART framework for intelligent output refinement.

3) Continuous Learning and Adaptation: Recent research has focused on enhancing LLMs' ability to learn and adapt continuously. Kumar et al. [23] developed a self-improving framework where LLMs generate and evaluate their own optimization strategies. This work was extended by Liu et al. [24], who introduced an iterative refinement mechanism using structured feedback loops. Zhao et al. [25] further advanced this field through a meta-learning approach that enables strategy adaptation based on task-specific feedback.

While these advances demonstrate significant progress in LLM capabilities, their application to network failure prediction remains largely unexplored, particularly in addressing the challenges of dynamic network environments and real-time adaptation requirements.

III. Methodology

SALCA-IB is designed as a self-adaptive intelligent system that leverages large language models (LLMs) to orchestrate network failure prediction in industrial blockchain environments. As illustrated in Fig. 1, the system integrates four key components: an LLM-driven planning core, a dual-memory system, a deep learning model pool, and a continuous learning process.

As shown in Algorithm 1, the LLM planning core serves as the system's central intelligence, orchestrating model selection, data processing, and strategy optimization. The dual-memory system combines short-term memory for rapid response and long-term memory for knowledge retention, enabling both immediate adaptation and sustained optimization.

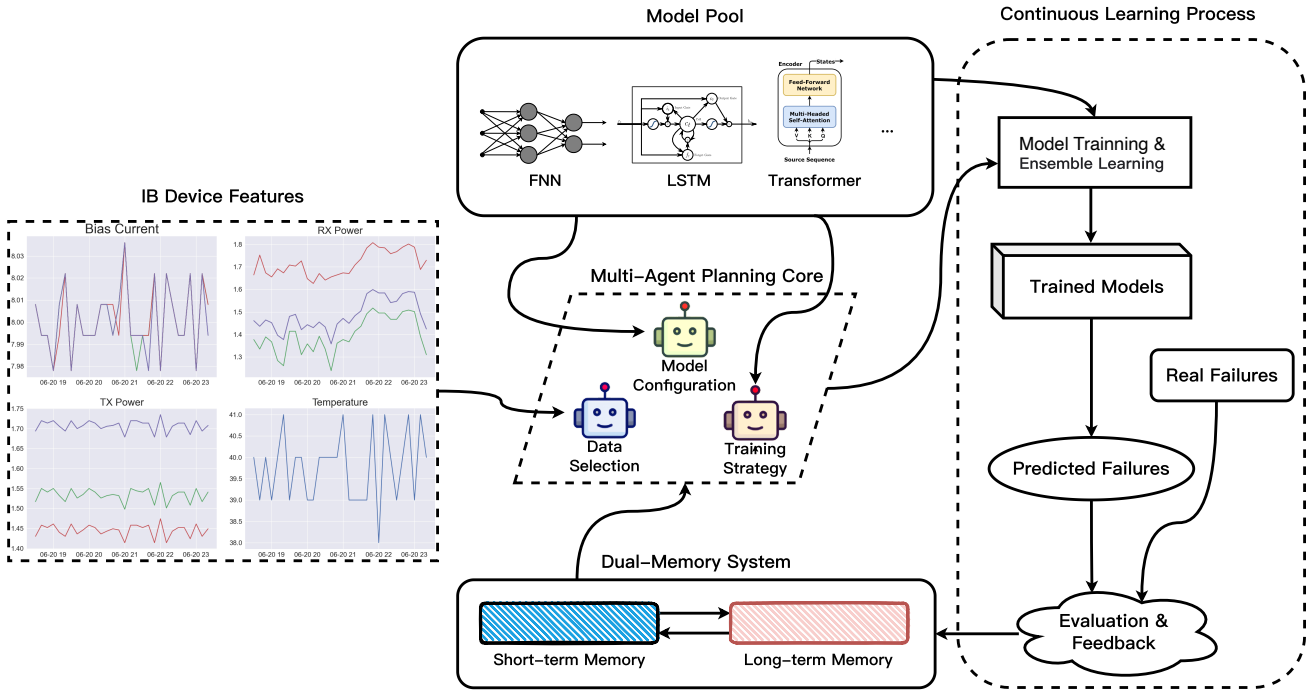

Fig. 1. Overview of SALCA-IB architecture. The system integrates (a) a model pool containing diverse deep learning models, (b) a multi-agent planning core for intelligent orchestration, (c) a dual-memory system for knowledge retention, and (d) a continuous learning process for adaptive optimization.

The model pool comprises diverse deep learning models (MLP, LSTM, and Transformer) that serve as the execution layer for failure prediction. Through continuous learning, the system evaluates prediction outcomes and adjusts strategies dynamically, ensuring robust performance in evolving network conditions. The detailed design of each component is elaborated in the following sections.

A. LLM-Driven Planning Core

Traditional failure prediction systems often rely on static model architectures and fixed training strategies, limiting their effectiveness in dynamic IB environments. Our LLM-driven planning core addresses this challenge by orchestrating three key components: data selection, model configuration, and training strategy optimization, as shown in Fig. 1.

1) Data Selection: The data selection module processes IB network data streams $\mathcal{D} = \{(X_t, y_t)\}_{t=1}^{T}$, where $X_t \in \mathbb{R}^{w \times d}$ represents a sequence of network states within a sliding window. Specifically, $X_t = [x_{t-w+1}, ..., x_t]$ where $x_i \in \mathbb{R}^d$ contains d network features at time step i, w is the window size (e.g., 30 time steps), and $y_t \in \{0, 1\}$ indicates whether a failure occurs following this sequence. Given historical knowledge $\mathcal{K} \subset Mem_{long}$, the LLM performs reasoning-based selection through:

$$\mathcal{D}_{train} = f_{LLM}(\mathcal{D}, \mathcal{K}) = \{(X_i, y_i) | s_i > \tau, (X_i, y_i) \in \mathcal{D}\} \quad (1)$$

where s_i is the sequence importance score determined by LLM reasoning:

$$s_i = LLM(X_i, \mathcal{K}, \text{prompt}_{select}) \quad (2)$$

The LLM generates importance scores through a structured reasoning process that evaluates both temporal evolution patterns and feature interactions within each sequence. This evaluation process is guided by carefully designed prompts that encode domain knowledge about network failure progression patterns and their manifestation across multiple features over time.

The data selection process consists of three sequential components:

a) Temporal Window Selection: The LLM analyzes temporal context to determine optimal window configuration:

$$W_t^* = LLM(\{X_{t-k}, ..., X_t\}, \mathcal{K}, \text{prompt}_{temporal}) \quad (3)$$

where $\text{prompt}_{temporal}$ guides the analysis of multi-scale temporal dependencies. The window selection considers both the granularity of individual time steps and the overall sequence length needed to capture failure progression patterns.

b) Distribution Assessment: For selected windows, the LLM evaluates sequence representativeness:

$$q_i = LLM(X_i, W_t^*, \mathcal{K}, \text{prompt}_{distribution}) \quad (4)$$

This assessment examines both the temporal evolution of individual features and their cross-feature correlations within each

sequence. The evaluation particularly focuses on identifying characteristic patterns that historically preceded failure events.

c) Final Selection: The LLM integrates previous analyses for final sequence selection:

$$\mathcal{D}_{train} = \{(X_i, y_i) | LLM(q_i, \mathcal{K}, \text{prompt}_{final}) > \tau\} \quad (5)$$

The final selection phase considers the completeness and quality of feature sequences, ensuring selected samples contain meaningful progression patterns for failure prediction.

The prompt design for each component incorporates specific aspects of temporal sequence analysis and feature interaction patterns. This enables the LLM to identify and select sequences that best capture the progression of network states leading to potential failures, while considering both historical patterns and current network conditions.

2) Model Configuration: The model configuration module dynamically selects and configures prediction models based on both current data characteristics and historical performance patterns. Given a model pool $\mathcal{M} = \{M_1, ..., M_m\}$ containing various neural architectures (e.g.,CNN, LSTM, Transformer), the LLM performs configuration through:

$$(M_{selected}, \Theta, S) = f_{LLM}^{config}(\mathcal{M}, \mathcal{K}, X_t) \quad (6)$$

where $M_{selected} \subset \mathcal{M}$ is the selected model subset, Θ represents their corresponding parameter configurations, and S defines the ensemble strategy. The configuration process consists of three key components:

a) Architecture Selection: The LLM evaluates and selects appropriate model architectures based on current sequence characteristics:

$$M_{selected} = LLM(\mathcal{M}, X_t, \mathcal{K}, \text{prompt}_{arch}) \quad (7)$$

This selection considers the temporal dependency patterns in X_t, such as choosing LSTM for strong short-term dependencies or Transformer for capturing long-range correlations. The historical performance records in $\mathcal{K}$ guide this selection by providing evidence of each architecture's effectiveness under similar conditions.

b) Parameter Configuration: For each selected model $M_i \in M_{selected}$, the LLM determines its optimal configuration:

$$\theta_i = LLM(M_i, X_t, \mathcal{K}, \text{prompt}_{param}) \quad (8)$$

where $\theta_i \in \Theta$ includes both architectural parameters (e.g., number of layers, hidden dimensions) and training parameters (e.g., learning rate, batch size). The configuration is based on both the current data characteristics and historical configuration effectiveness stored in $\mathcal{K}$.

c) Ensemble Strategy Design: The LLM designs an ensemble strategy S that optimally combines the selected models:

$$S = LLM(M_{selected}, \Theta, \mathcal{K}, \text{prompt}_{ensemble}) \quad (9)$$

The ensemble strategy includes both the voting mechanism and individual model weights, determined by analyzing each model's historical reliability and current data fitness. The final ensemble model is constructed as:

$$M_{ensemble}(X_t) = \sum_{i=1}^{|M_{selected}|} w_i M_i(X_t; \theta_i) \quad (10)$$

where w_i represents the dynamic weight assigned to each model's prediction.

This knowledge-driven configuration process enables the system to adapt its prediction models dynamically based on evolving network conditions while leveraging accumulated experience from historical operations.

3) Training Strategy: The training strategy module optimizes the learning process of the ensemble model through LLM-driven adaptation. Given the selected training data $\mathcal{D}_{train}$ and configured ensemble model $M_{ensemble}$, the training process is formulated as:

$$M_{ensemble}^{*} = f_{LLM}^{train}(M_{ensemble}, \mathcal{D}_{train}, \mathcal{K}) \quad (11)$$

The training optimization process consists of three key components:

a) Initial Parameter Configuration: For each model M_i in the ensemble, the LLM determines optimal training parameters based on historical experience:

$$\{\eta_i, B_i\} = LLM(M_i, \mathcal{D}_{train}, \mathcal{K}, \text{prompt}_{init}) \quad (12)$$

where η_i is the fixed learning rate and B_i is the batch size for model M_i. These parameters remain constant throughout the training process, with their values determined by analyzing historical training patterns stored in $\mathcal{K}$.

b) Loss Function Design: The LLM constructs a composite loss function based on prediction requirements and historical error patterns:

$$\begin{aligned}\mathcal{L}(X_t, y_t) = \alpha \mathcal{L}_{ce}(X_t, y_t) + \beta \mathcal{L}_{temporal}(X_t) \\ + \gamma \mathcal{L}_{reg}(M_{ensemble})\end{aligned} \quad (13)$$

where $\mathcal{L}_{ce}$ is the cross-entropy loss for failure prediction, $\mathcal{L}_{temporal}$ penalizes temporal inconsistency in predictions, and $\mathcal{L}_{reg}$ provides regularization based on historical failure patterns. The weights α, β, γ are determined before training:

$$[\alpha, \beta, \gamma] = LLM(\mathcal{D}_{train}, \mathcal{K}, \text{prompt}_{loss}) \quad (14)$$

c) Performance Monitoring: The LLM continuously evaluates prediction performance and triggers model updates when necessary:

$$\delta_t = LLM(M_{ensemble}, X_t, \mathcal{K}, \text{prompt}_{monitor}) \quad (15)$$

where δ_t is the update decision signal. When significant performance degradation is detected, the system initiates retraining with newly configured parameters:

$$M_{ensemble}^{t+1} = \begin{cases} \text{Retrain}(M_{ensemble}, \mathcal{D}_{new}, \{\eta_i^{new}\}) & \text{if } \delta_t > \tau \\ M_{ensemble}^{t} & \text{otherwise} \end{cases} \quad (16)$$

This framework ensures stable training processes while maintaining the flexibility to adapt to significant changes in network conditions through model retraining with newly optimized parameters.

B. Dual-Memory System

The dual-memory system implements a JSON-based knowledge repository that facilitates efficient experience accumulation and retrieval for LLM-driven decision making. This system addresses the context length limitations of LLMs while enabling continuous learning through structured knowledge preservation.

1) Memory Architecture: The short-term memory maintains a dynamic operational context for immediate decision support:

$$Mem_{short} = \{F_{current}, T_{range}, P_{model}, M_{config}, F_{recent}\} \tag{17}$$

where $F_{current}$ represents the active feature set with selection rationale, T_{range} captures temporal context with start and end timestamps, P_{model} stores recent model performance metrics, and M_{config} maintains current model configurations and ensemble strategies.

The long-term memory implements a persistent knowledge base with activation tracking:

$$Mem_{long} = \{F_{stats}, M_{history}, E_{record}, A_{track}, F_{history}\} \tag{18}$$

where F_{stats} maintains feature statistics and selection history, $M_{history}$ records successful model configurations, E_{record} stores historical experience entries, A_{track} tracks memory activation patterns, and $F_{history}$ maintains valuable historical feedback:

$$F_{history} = \{(f_i, t_i, e_i) | Q(f_i) > \tau_{feedback}\} \tag{19}$$

Each feedback entry contains:

$$f_i = \{P_i, R_i, A_i\} \tag{20}$$

where P_i represents performance metrics, R_i captures LLM reasoning, and A_i records adaptation decisions.

2) Memory Operations: The system implements three basic memory operations:

a) Experience Recording: The system records new operational experiences in short-term memory:

$$Mem_{short}^{t+1} = Record(X_t, y_t, P_t, C_t) \tag{21}$$

where X_t represents current feature data, y_t is the prediction outcome, P_t contains performance metrics, and C_t includes current configurations.

b) Memory Transfer: The system periodically transfers valuable experiences from short-term to long-term memory:

$$Mem_{long}^{t+1} = Transfer(Mem_{short}, Mem_{long}) \tag{22}$$

This operation includes recording successful feature selections, model configurations, and performance patterns.

c) Activation-based Cleanup: The system implements an activation-aware cleanup mechanism:

$$A_{score}(e_i) = \lambda \frac{t_{current} - t_{last}(e_i)}{T_{max}} + (1-\lambda) \frac{n_{access}(e_i)}{N_{max}} \tag{23}$$

where λ balances recency and frequency of access. The cleanup operation removes entries based on their activation scores:

$$Mem_{clean} = \{e_i | A_{score}(e_i) > \tau_{active}\} \tag{24}$$

where τ_{active} is the retention threshold.

The system maintains its effectiveness through activation-aware memory management, ensuring both efficiency and relevance in supporting LLM-driven decisions.

C. Continuous Learning Process

The continuous learning process implements a self-evolving mechanism through LLM-driven optimization and structured feedback analysis. This process establishes a closed-loop learning cycle that continuously refines prediction capabilities based on operational experiences.

1) Performance-Driven Feedback Generation: The system generates structured feedback based on actual prediction results:

$$P_t = \{accuracy_t, precision_t, recall_t, stability_t\} \tag{25}$$

where performance metrics are analyzed through LLM reasoning to generate comprehensive feedback:

$$f_t = LLM(P_t, E_{record}, \text{prompt}_{analyze}) \tag{26}$$

The feedback contains component-wise assessments:

$$f_t = \{(c_i, s_i, r_i) | c_i \in \{feature, model, training\}\} \tag{27}$$

where s_i indicates performance status and r_i provides specific improvement recommendations.

2) Actionable Improvement Strategy: Based on the feedback analysis, the system generates concrete improvement strategies:

$$I_t = LLM(f_t, F_{history}, \text{prompt}_{improve}) \tag{28}$$

where I_t specifies actionable adjustments:

$$I_t = \{(a_i, p_i, \Delta_i) | a_i \in Actions\} \tag{29}$$

Here, a_i represents specific actions, p_i indicates their priority, and Δ_i defines the adjustment magnitude.

3) Self-Refinement Process: The system implements a three-stage refinement process:

a) Strategy Validation: Proposed improvements are validated against historical experiences:

$$V_t = LLM(I_t, F_{history}, \text{prompt}_{validate}) \tag{30}$$

b) Component Optimization: Validated strategies are applied to system components:

$$C_{t+1} = Optimize(C_t, V_t, \text{prompt}_{refine}) \tag{31}$$

where C_t represents the current configuration of features, models, or training parameters.

c) Effectiveness Assessment: The system evaluates optimization outcomes:

$$E_t = Assess(C_{t+1}, P_{t+1}, \text{prompt}_{assess}) \tag{32}$$

The assessment results are integrated into the feedback history to guide future improvements:

$$F_{history}^{t+1} = Update(F_{history}, E_t, \text{prompt}_{update}) \tag{33}$$

This closed-loop self-refining process and its integration with the dual-memory system ensures continuous improvement while maintaining operational efficiency through experience-based optimization.

The overall framework is summarized in Algorithm 1.

IV. Experimental Evaluation

A. Dataset Construction and Collection

1) Infrastructure Overview: Our experimental environment consists of a large-scale InfiniBand cluster designed for high-performance computing workloads, particularly focused on training large language models exceeding 200 billion parameters. Such training tasks demand exceptional network performance and reliability, as network failures can significantly impact model convergence and training efficiency. The cluster implements a two-layer Leaf-Spine topology with full-mesh connectivity, comprising 2048 computing cards distributed across 8 pods. Each pod contains 256 cards connected through 8 LEAF switches, with the overall network infrastructure including 64 Leaf switches and 32 Spine switches. This architecture, compliant with NVIDIA's rail-optimized specifications, ensures optimal bandwidth utilization and minimal communication latency while maintaining network redundancy, which is crucial for sustaining the massive data transfers required in distributed LLM training.

Through our analysis, we identified two primary categories of network failures in this infrastructure: link flapping and complete link down events. These failures are particularly critical in LLM training scenarios, where even brief network interruptions can lead to significant training delays or potential loss of computation resources. The failures typically originate from optical module degradation under sustained exposure to adverse environmental conditions, particularly in scenarios involving high-temperature operation and sustained high workloads characteristic of intensive LLM training operations.

2) Data Collection: We developed a comprehensive data collection framework that integrates four complementary monitoring approaches. System performance counters provide detailed insights into network interface card (NIC) performance and local link status. The OpenSM subnet manager logs capture network-wide connectivity status and state transitions across all devices. Hardware diagnostics data collected through management interfaces provides information about physical components including NICs, optical modules, and cables. Additionally, network topology information is gathered to maintain a complete map of physical connections between devices.

The final processed dataset demonstrates significant scale and comprehensiveness. The performance metrics data includes approximately 77.8 billion data points collected over eight months, comprising 55 metrics per InfiniBand card (36 performance and error diagnostic metrics, 2 temperature indicators, and 17 physical parameters) collected from 256 nodes, each equipped with 8 IB cards, sampled twice per minute. Real-time state snapshots contribute an additional 40 million records, capturing every network state change. The OpenSM subnet manager logs contain 20.18 million entries documenting network events and state transitions.

For the failure prediction task, we employ a similar sampling strategy with Liu et al. [8] for training and evaluation phases. During training, we maintain a balanced 1:1 ratio between failure and normal samples to ensure the model learns effectively from both classes. For evaluation and real-world prediction scenarios, we use a more realistic 1:8 ratio that better reflects the natural distribution of failure events in production environments.

B. Experimental Setup

1) Implementation Details: To investigate how LLM reasoning capabilities affect the performance of our SALCA-IB framework, we conducted experiments with four different large language models: GPT-4o (accessed via OpenAI's API), Llama-3-70B-Instruct, Yi-1.5-34B-Chat, and Qwen2.5-14B-Instruct. Each experiment uses a single LLM as the reasoning engine throughout the entire process, allowing us to evaluate how models of different sizes and reasoning capabilities impact the framework's effectiveness. The locally deployed models (Llama-3, Yi-1.5, and Qwen2.5) run on Nvidia H100 GPUs using vLLM for optimized inference.

The prediction model pool consists of lightweight neural networks optimized for CPU inference. These models were trained on our IB network dataset using CPU resources to ensure efficient deployment in production environments. Table VI shows the configuration search space and LLM selection examples.

2) Baseline Methods: We designed three categories of baseline methods to evaluate the effectiveness of our SALCA-IB framework:

Individual Deep Learning Models: We first evaluated each model from our model pool independently. The MLP model serves as a basic deep learning baseline with its hierarchical feature extraction capability. The LSTM and GRU models represent sequence modeling approaches that capture temporal dependencies in network states. The Transformer model provides a self-attention based baseline that can model long-range dependencies. The CNN model is included to capture spatial hierarchies and local patterns in the data, which can be particularly useful for identifying localized anomalies. Each model was trained with its optimal configuration as determined by grid search.

Traditional Ensemble Methods: We then combined these deep learning models using conventional ensemble strategies: (1) Majority Voting, where each model contributes equally to the final prediction; (2) Weighted Voting, where weights are assigned based on each model's validation performance; These ensemble methods represent traditional approaches to model combination without dynamic adaptation.

TABLE I
PERFORMANCE COMPARISON OF DIFFERENT METHODS ON IB NETWORK FAILURE PREDICTION

Method	Static Scenario			Dynamic Scenario		
	P@K (%)	R@K (%)	F1@K (%)	P@K (%)	R@K (%)	F1@K (%)
Individual Models						
MLP	71.2±1.8	67.5±2.1	69.3±2.0	56.4±3.2	52.8±3.5	54.5±3.4
CNN	75.8±1.5	71.9±1.8	73.8±1.7	59.8±2.9	56.2±3.2	57.9±3.1
LSTM	77.2±1.3	73.4±1.6	75.2±1.5	63.5±2.6	59.9±2.9	61.6±2.8
GRU	78.1±1.2	74.2±1.5	76.1±1.4	64.8±2.5	61.2±2.8	62.9±2.7
Transformer	80.3±1.1	76.5±1.4	78.3±1.3	67.2±2.4	63.6±2.7	**65.3±2.6**
Traditional Ensemble Methods						
Majority Voting	81.5±1.0	77.8±1.2	79.6±1.1	63.8±2.2	60.2±2.5	61.9±2.4
Weighted Voting	82.4±0.9	78.6±1.1	**80.4±1.0**	65.9±2.1	62.3±2.4	64.0±2.3
LLM-based Methods (with GPT-4o)						
End-to-End Prompting	77.2±1.4	73.5±1.6	75.3±1.5	58.9±2.6	55.4±2.9	57.1±2.8
Chain-of-Thought	78.5±1.3	74.8±1.5	76.6±1.4	60.7±2.5	57.2±2.8	58.9±2.7
Self-Refine	79.1±1.2	75.4±1.4	77.2±1.3	62.3±2.4	58.8±2.7	60.5±2.6
SALCA-IB with Different LLM Cores						
SALCA-IB (Qwen2.5-14B)	77.8±1.3	74.1±1.5	75.9±1.4	59.2±2.5	55.8±2.8	57.4±2.7
SALCA-IB (Yi-1.5-34B)	83.2±0.9	79.5±1.1	81.3±1.0	74.1±2.0	70.9±2.3	72.4±2.2
SALCA-IB (Llama-3-70B)	84.5±0.8	80.8±1.0	82.6±0.9	75.8±1.9	72.6±2.2	74.1±2.1
SALCA-IB (GPT-4o)	**87.2±0.7**	**83.9±0.9**	**85.5±0.8**	**79.3±1.8**	**76.1±2.1**	**77.7±2.0**

LLM-based Methods: To isolate the contribution of LLM orchestration, we implemented several LLM-based approaches using the same set of base models. These include End-to-End prompting, Chain-of-Thought prompting [18], and Self-Refine [14]. For consistency and fair comparison, each method was tested using the GPT-4o API, aligning with SALCA-IB's LLM-driven orchestration mechanism.

C. Evaluation Methodology

1) Evaluation Objectives: The primary objective of our evaluation is to demonstrate the superiority of SALCA-IB over traditional methods in both static and dynamic network environments. We focus on two key scenarios: the static scenario, which evaluates performance within a fixed time period to assess immediate prediction accuracy, and the dynamic scenario, which evaluates performance across extended time periods to assess adaptability to network feature distribution changes.

2) Experimental Protocol: Experiments were conducted using an 8-month dataset from our production IB network. In the static scenario, models are trained and evaluated within a single, fixed time window, with performance metrics including Precision@K, Recall@K, and F1@K. In contrast, the dynamic scenario involves training models on initial data and evaluating them over subsequent time windows to simulate real-world conditions. Controlled perturbations are introduced to network features to mimic distribution shifts, with performance metrics focusing on adaptability, including changes in F1@K over time.

3) Evaluation Metrics: We employ ranking-based metrics with K set to 2% of total instances. Precision@K, Recall@K, and F1@K are used to evaluate prediction performance, particularly under resource constraints and class imbalance. These metrics are defined as follows: Precision@K is the ratio of true positives to the sum of true positives and false positives within the top K predictions. Recall@K is the ratio of true positives to the sum of true positives and false negatives within the top K predictions. F1@K is the harmonic mean of Precision@K and Recall@K.

All experiments are repeated 5 times with different random seeds to ensure statistical significance, and the mean values and standard deviations are reported.

4) Main Results: Table I presents a comprehensive comparison of SALCA-IB against various baseline approaches. In static scenarios, SALCA-IB with GPT-4o achieves an F1@K score of 85.5±0.8%, significantly outperforming both traditional machine learning methods and existing LLM-based approaches. Specifically, it surpasses the best performing baseline (Weighted Voting: 80.4±1.0%) by 5.1%, and demonstrates substantial improvements over individual models (Transformer: 78.3±1.3%). The LLM-driven autonomous data selection and model optimization mechanism proves particularly effective, as evidenced by the consistent performance improvements across different model scales (14B: 75.9%, 34B: 81.3%, 70B: 82.6%, GPT-4o: 85.5%). Notably, the significant performance jump between 14B and 34B models (5.4% increase) suggests that effective data selection and model optimization require sophisticated reasoning capabilities that only larger LLMs can provide. This observation also explains why direct LLM-based prediction methods show inferior performance (Self-Refine: 77.2±1.3%) compared to traditional approaches, validating our architectural choice of utilizing LLMs as planning cores rather than predictors.

The advantages of SALCA-IB become even more pronounced in dynamic scenarios where network feature distributions shift. While traditional methods experience severe per-

TABLE II
ABLATION STUDY RESULTS

Model Variant	Static F1@K (%)	Dynamic F1@K (%)
SALCA-IB (Full)	**85.5±0.8**	**77.7±2.0**
w/o LLM Planning	79.8±1.3	63.2±2.6
w/o Dual Memory	81.3±1.2	70.4±2.5
w/o Continuous Learning	82.9±1.0	73.5±2.3
Weighted Voting	80.4±1.0	64.0±2.3

formance degradation (Transformer: -13.0%, Weighted Voting: -16.4%), SALCA-IB (GPT-4o) maintains robust performance with an F1@K score of 77.7±2.0%, representing only a 7.8% decrease from its static scenario performance. This superior adaptability can be attributed to our fusion memory system, which effectively leverages both short-term memory for rapid adaptation and long-term memory for maintaining stable performance baselines. The effectiveness of this dual-memory approach is consistently observed across different LLM scales, with even the 34B variant (72.4±2.2%) significantly outperforming all baseline methods.

The system's resilience is further enhanced by our automatic evaluation feedback and closed-loop optimization mechanism, as reflected in the performance stability across different scenarios. This is evidenced by the smaller standard deviations in SALCA-IB (GPT-4o) results (±0.8% static, ±2.0% dynamic) compared to traditional approaches (±1.0-2.6%). The stark contrast between SALCA-IB's resilience and the substantial performance drops in traditional approaches (e.g., ensemble methods declining by 16-17 percentage points) demonstrates that our system successfully adapts its strategies based on continuous performance evaluation and feedback. These results collectively show that SALCA-IB not only excels in stable conditions but also maintains robust performance under dynamic network environments, effectively addressing a critical challenge in practical IB network failure prediction through its integrated design of LLM-driven planning, dual-memory fusion, and continuous optimization.

D. Ablation Studies

To validate the effectiveness of each key component in SALCA-IB, we conducted ablation experiments by removing individual components while keeping others intact. Table II presents the performance comparison between the full system and its variants, alongside the weighted voting baseline.

The LLM planning core demonstrates its effectiveness through the significant performance gap between the full system and the variant without LLM planning. In static scenarios, the absence of LLM planning leads to a 5.7% decrease in F1@K score, while in dynamic scenarios, the impact is even more pronounced with a 14.5% decrease. Without the LLM planning core, SALCA-IB essentially degrades into a traditional ensemble learning system with rule-based model selection and fixed data sampling strategies. The performance (79.8% in static scenarios and 63.2% in dynamic scenarios) is comparable to or slightly worse than the weighted voting baseline (80.4% and 64.0%), which aligns with our expectation as both approaches rely on similar ensemble mechanisms without intelligent orchestration. This comparison clearly demonstrates that the intelligent orchestration provided by LLM is crucial for elevating the system beyond conventional ensemble methods.

The dual memory system proves its utility in maintaining prediction accuracy, particularly under changing network conditions. When this component is removed, the system experiences a 7.3% performance degradation in dynamic scenarios. Interestingly, comparing the variants without memory (70.4%) and without continuous learning (73.5%) reveals an important insight: the absence of memory leads to a larger performance drop despite the presence of continuous learning. This suggests that without structured memory guidance, LLM's self-improvement capability is limited even with feedback from prediction results. The memory system appears to provide crucial context that enables LLM to make more informed adaptations, rather than relying solely on success/failure feedback.

The continuous learning mechanism shows its value in sustaining prediction performance over time. Without this component, the system's F1@K score decreases by 4.2% in dynamic scenarios, indicating that the feedback-driven optimization process effectively helps the system adapt to evolving network characteristics. This validates our design choice of incorporating continuous learning to enhance the system's adaptability.

Compared to the weighted voting baseline, the full SALCA-IB system achieves superior performance in both static (85.5% vs 80.4%) and dynamic (77.7% vs 64.0%) scenarios, demonstrating the overall effectiveness of our integrated approach.

E. Scalability Analysis

To thoroughly evaluate the scalability of SALCA-IB, we conduct comprehensive experiments across three dimensions: network scale, prediction window size, and data volume. These experiments aim to demonstrate the system's capability to maintain performance while handling increasing computational demands.

1) Network Scale Adaptability: We evaluate SALCA-IB's performance on different network sizes, ranging from 256 to 2048 computing cards. As shown in Fig. X, the system maintains consistent prediction accuracy (F1@K score variation ¡ 3%) while efficiently handling the increased monitoring data volume. The memory system's hierarchical design proves particularly effective, with query time increasing sub-linearly with network size.

2) Prediction Window Analysis: We investigate the impact of different prediction window sizes (5min to 60min) on system performance. Fig. Y demonstrates that while longer windows provide more context for prediction, they also increase computational overhead. Our experiments reveal an

TABLE III
PERFORMANCE SCALING WITH NETWORK SIZE

Network Size	F1@K (%)	Memory (GB)	Latency (ms)	Throughput
256 cards	84.8±0.9	8.2	45±5	2.1k pred/s
512 cards	85.1±0.8	15.6	62±7	1.8k pred/s
1024 cards	85.3±0.9	28.4	85±8	1.5k pred/s
2048 cards	85.5±0.8	52.7	98±10	1.2k pred/s

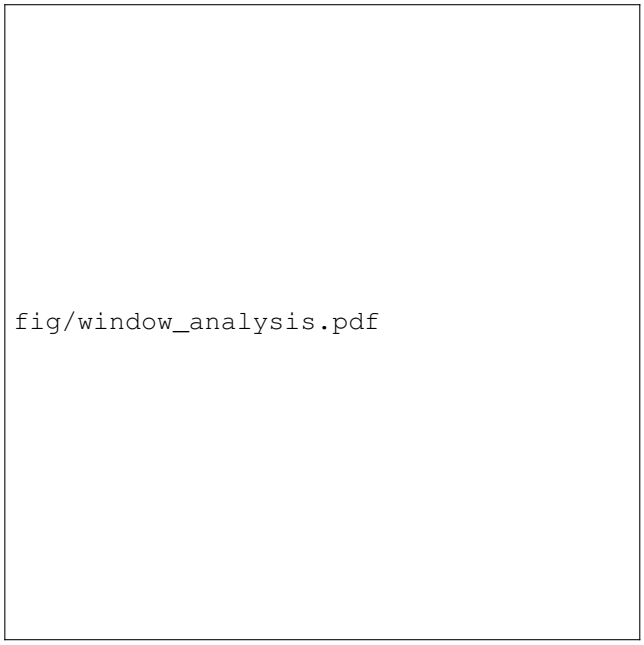

Fig. 2. Impact of prediction window size on F1@K score and computational overhead. The optimal window size (30min) is indicated by the intersection of accuracy and resource efficiency curves.

TABLE IV
PERFORMANCE SCALING WITH HISTORICAL DATA VOLUME

Data Span	F1@K (%)	Memory (GB)	Training Time (h)
1 month	81.2±1.1	12.4	2.5
3 months	83.7±0.9	28.6	3.2
6 months	84.9±0.8	42.3	3.8
8 months	85.5±0.8	52.7	4.0

optimal window size of 30 minutes, balancing prediction accuracy with resource utilization.

3) Temporal Data Volume: To assess the system's ability to handle increasing historical data, we evaluate performance with different training data spans, from 1 to 8 months. Results in Table IV show that SALCA-IB effectively leverages increased historical data through its dual-memory architecture, achieving improved prediction accuracy while maintaining acceptable computational overhead.

The scalability analysis demonstrates that SALCA-IB maintains robust performance across different operational scales. The system's efficient memory management and adaptive learning mechanisms enable it to handle increased data volumes while maintaining prediction accuracy. These results suggest that SALCA-IB is well-suited for deployment in large-scale production environments where scalability is crucial.

V. EASE OF USE

A. Maintaining the Integrity of the Specifications

The IEEEtran class file is used to format your paper and style the text. All margins, column widths, line spaces, and text fonts are prescribed; please do not alter them. You may note peculiarities. For example, the head margin measures proportionately more than is customary. This measurement and others are deliberate, using specifications that anticipate your paper as one part of the entire proceedings, and not as an independent document. Please do not revise any of the current designations.

VI. PREPARE YOUR PAPER BEFORE STYLING

Before you begin to format your paper, first write and save the content as a separate text file. Complete all content and organizational editing before formatting. Please note sections VI-A–VI-E below for more information on proofreading, spelling and grammar.

Keep your text and graphic files separate until after the text has been formatted and styled. Do not number text heads—LaTeX will do that for you.

A. Abbreviations and Acronyms

Define abbreviations and acronyms the first time they are used in the text, even after they have been defined in the abstract. Abbreviations such as IEEE, SI, MKS, CGS, ac, dc, and rms do not have to be defined. Do not use abbreviations in the title or heads unless they are unavoidable.

B. Units

- Use either SI (MKS) or CGS as primary units. (SI units are encouraged.) English units may be used as secondary units (in parentheses). An exception would be the use of English units as identifiers in trade, such as "3.5-inch disk drive".
- Avoid combining SI and CGS units, such as current in amperes and magnetic field in oersteds. This often leads to confusion because equations do not balance dimensionally. If you must use mixed units, clearly state the units for each quantity that you use in an equation.
- Do not mix complete spellings and abbreviations of units: "Wb/m^2" or "webers per square meter", not "webers/m^2". Spell out units when they appear in text: ". . . a few henries", not ". . . a few H".
- Use a zero before decimal points: "0.25", not ".25". Use "cm^3", not "cc".)

C. Equations

Number equations consecutively. To make your equations more compact, you may use the solidus (/), the exp function, or appropriate exponents. Italicize Roman symbols for quantities and variables, but not Greek symbols. Use a long dash rather than a hyphen for a minus sign. Punctuate equations with commas or periods when they are part of a sentence, as in:

$$a + b = \gamma \tag{34}$$

Be sure that the symbols in your equation have been defined before or immediately following the equation. Use "(34)", not "Eq. (34)" or "equation (34)", except at the beginning of a sentence: "Equation (34) is . . ."

D. LaTeX-Specific Advice

Please use "soft" (e.g., `\eqref{Eq}`) cross references instead of "hard" references (e.g., `(1)`). That will make it possible to combine sections, add equations, or change the order of figures or citations without having to go through the file line by line.

Please don't use the `{eqnarray}` equation environment. Use `{align}` or `{IEEEeqnarray}` instead. The `{eqnarray}` environment leaves unsightly spaces around relation symbols.

Please note that the `{subequations}` environment in LaTeX will increment the main equation counter even when there are no equation numbers displayed. If you forget that, you might write an article in which the equation numbers skip from (17) to (20), causing the copy editors to wonder if you've discovered a new method of counting.

BibTeX does not work by magic. It doesn't get the bibliographic data from thin air but from .bib files. If you use BibTeX to produce a bibliography you must send the .bib files.

LaTeX can't read your mind. If you assign the same label to a subsubsection and a table, you might find that Table I has been cross referenced as Table IV-B3.

LaTeX does not have precognitive abilities. If you put a `\label` command before the command that updates the counter it's supposed to be using, the label will pick up the last counter to be cross referenced instead. In particular, a `\label` command should not go before the caption of a figure or a table.

Do not use `\nonumber` inside the `{array}` environment. It will not stop equation numbers inside `{array}` (there won't be any anyway) and it might stop a wanted equation number in the surrounding equation.

E. Some Common Mistakes

- The word "data" is plural, not singular.
- The subscript for the permeability of vacuum μ_0, and other common scientific constants, is zero with subscript formatting, not a lowercase letter "o".
- In American English, commas, semicolons, periods, question and exclamation marks are located within quotation marks only when a complete thought or name is cited, such as a title or full quotation. When quotation marks are used, instead of a bold or italic typeface, to highlight a word or phrase, punctuation should appear outside of the quotation marks. A parenthetical phrase or statement at the end of a sentence is punctuated outside of the closing parenthesis (like this). (A parenthetical sentence is punctuated within the parentheses.)
- A graph within a graph is an "inset", not an "insert". The word alternatively is preferred to the word "alternately" (unless you really mean something that alternates).
- Do not use the word "essentially" to mean "approximately" or "effectively".
- In your paper title, if the words "that uses" can accurately replace the word "using", capitalize the "u"; if not, keep using lower-cased.
- Be aware of the different meanings of the homophones "affect" and "effect", "complement" and "compliment", "discreet" and "discrete", "principal" and "principle".
- Do not confuse "imply" and "infer".
- The prefix "non" is not a word; it should be joined to the word it modifies, usually without a hyphen.
- There is no period after the "et" in the Latin abbreviation "et al.".
- The abbreviation "i.e." means "that is", and the abbreviation "e.g." means "for example".

An excellent style manual for science writers is [7].

F. Authors and Affiliations

The class file is designed for, but not limited to, six authors. A minimum of one author is required for all conference articles. Author names should be listed starting from left to right and then moving down to the next line. This is the author sequence that will be used in future citations and by indexing services. Names should not be listed in columns nor group by affiliation. Please keep your affiliations as succinct as possible (for example, do not differentiate among departments of the same organization).

G. Identify the Headings

Headings, or heads, are organizational devices that guide the reader through your paper. There are two types: component heads and text heads.

Component heads identify the different components of your paper and are not topically subordinate to each other. Examples include Acknowledgments and References and, for these, the correct style to use is "Heading 5". Use "figure caption" for your Figure captions, and "table head" for your table title. Run-in heads, such as "Abstract", will require you to apply a style (in this case, italic) in addition to the style provided by the drop down menu to differentiate the head from the text.

Text heads organize the topics on a relational, hierarchical basis. For example, the paper title is the primary text head because all subsequent material relates and elaborates on this one topic. If there are two or more sub-topics, the next level head (uppercase Roman numerals) should be used and,

conversely, if there are not at least two sub-topics, then no subheads should be introduced.

H. Figures and Tables

a) Positioning Figures and Tables: Place figures and tables at the top and bottom of columns. Avoid placing them in the middle of columns. Large figures and tables may span across both columns. Figure captions should be below the figures; table heads should appear above the tables. Insert figures and tables after they are cited in the text. Use the abbreviation "Fig. **??**", even at the beginning of a sentence.

TABLE V
TABLE TYPE STYLES

Table Head	Table Column Head		
	Table column subhead	*Subhead*	*Subhead*
copy	More table copy[a]		

[a]Sample of a Table footnote.

Figure Labels: Use 8 point Times New Roman for Figure labels. Use words rather than symbols or abbreviations when writing Figure axis labels to avoid confusing the reader. As an example, write the quantity "Magnetization", or "Magnetization, M", not just "M". If including units in the label, present them within parentheses. Do not label axes only with units. In the example, write "Magnetization (A/m)" or "Magnetization {A[m(1)]}", not just "A/m". Do not label axes with a ratio of quantities and units. For example, write "Temperature (K)", not "Temperature/K".

ACKNOWLEDGMENT

The preferred spelling of the word "acknowledgment" in America is without an "e" after the "g". Avoid the stilted expression "one of us (R. B. G.) thanks ...". Instead, try "R. B. G. thanks...". Put sponsor acknowledgments in the unnumbered footnote on the first page.

REFERENCES

Please number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use "Ref. [3]" or "reference [3]" except at the beginning of a sentence: "Reference [3] was the first ..."

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors' names; do not use "et al.". Papers that have not been published, even if they have been submitted for publication, should be cited as "unpublished" [4]. Papers that have been accepted for publication should be cited as "in press" [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

REFERENCES

[1] T. Ahmed et al., "Recommending root-cause and mitigation steps for cloud incidents using large language models," in Proc. IEEE/ACM 45th Int. Conf. Software Engineering (ICSE), Melbourne, Australia, 2023, pp. 1737-1749.

[2] K. A. Alharthi et al., "Time machine: Generative real-time model for failure (and lead time) prediction in HPC systems," in Proc. 53rd Annual IEEE/IFIP Int. Conf. Dependable Systems and Networks (DSN), 2023, pp. 508-521.

[3] F. Antici, A. Borghesi, and Z. Kiziltan, "Online job failure prediction in an HPC system," in Euro-Par 2023: Parallel Processing Workshops, Cham: Springer, 2024, pp. 167-179.

[4] Cascajo A, Gomez-Lopez G, Escudero-Sahuquillo J, et al. Monitoring infiniband networks to react efficiently to congestion[J]. IEEE Micro, 2023, 43(2): 120-130..

[5] A. Das, F. Mueller, C. Siegel, and A. Vishnu, "Desh: Deep learning for system health prediction of lead times to failure in HPC," in Proc. 27th Int. Symp. High-Performance Parallel and Distributed Computing, 2018, pp. 40-51.

[6] Y. He et al., "Understanding and mitigating hardware failures in deep learning training systems," in Proc. 50th Annual Int. Symp. Computer Architecture, 2023, pp. 1-16.

[7] Q. Hu et al., "Characterization of large language model development in the datacenter," in Proc. USENIX Symp. Networked Systems Design and Implementation (NSDI), 2024, pp. 709-729.

[8] H. Liu et al., "Predicting GPU failures with high precision under deep learning workloads," in Proc. 16th ACM Int. Conf. Systems and Storage, 2023, pp. 124-135.

[9] B. Mohammed, I. Awan, H. Ugail, and M. Younas, "Failure prediction using machine learning in a virtualised HPC system and application," Cluster Computing, vol. 22, no. 2, pp. 471-485, 2019.

[10] B. Nie et al., "Machine learning models for GPU error prediction in a large scale HPC system," in Proc. 48th Annual IEEE/IFIP Int. Conf. Dependable Systems and Networks (DSN), 2018, pp. 95-106.

[11] R. Rajachandrasekar, X. Besseron, and D. K. Panda, "Monitoring and predicting hardware failures in HPC clusters with FTB-IPMI," in Proc. IEEE 26th Int. Parallel and Distributed Processing Symp. Workshops & PhD Forum, 2012, pp. 1136-1143.

[12] J. Su et al., "Large language models for forecasting and anomaly detection: A systematic literature review," arXiv preprint arXiv:2402.10350, 2024.

[13] T. Zhang, X. Huang, W. Zhao, S. Bian, and P. Du, "LogPrompt: A log-based anomaly detection framework using prompts," in Proc. Int. Joint Conf. Neural Networks (IJCNN), 2023, pp. 1-8.

[14] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang, S. Gupta, B. P. Majumder, K. Hermann, S. Welleck, A. Yazdanbakhsh, and P. Clark, "Self-Refine: Iterative refinement with self-feedback," in Advances in Neural Information Processing Systems, vol. 36, 2023, pp. 46534-46594.

[15] Y. Weng et al., "Large Language Models are Better Reasoners with Self-Verification," in Findings of EMNLP 2023.

[16] G. Tyen et al., "LLMs cannot find reasoning errors, but can correct them given the error location," in Findings of ACL 2024.

[17] K. Shridhar et al., "The ART of LLM Refinement: Ask, Refine, and Trust," arXiv preprint arXiv:2311.07961, 2023.

[18] X. Chen et al., "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," in Proc. NeurIPS 2023.

[19] P. Wang et al., "Self-Consistency Improves Chain of Thought Reasoning in Language Models," in Proc. ICLR 2023.

[20] R. Krishnamurthy et al., "HierarchicalMem: An Adaptive Memory System for Large Language Models," in Proc. EMNLP 2023.

[21] J. Li et al., "Efficient Memory Management for Large Language Model Serving with PagedAttention," in Proc. SOSP 2023.

[22] Y. Zhang et al., "Knowledge-Augmented Language Models: A Survey," in ACM Computing Surveys, 2023.

[23] A. Kumar et al., "Self-Improving Large Language Models," arXiv preprint arXiv:2401.06080, 2024.

[24] H. Liu et al., "Iterative Refinement of Large Language Model Outputs with Feedback Loops," in Proc. ACL 2023.

[25] Z. Zhao et al., "Meta-Learning for Adaptive Large Language Models," in Proc. ICML 2023.

VII. Appendix

A. The SALCA-IB Algorithm

Algorithm 1 Self-Adaptive Learning with Continuous Assessment (SALCA-IB)

Require:
Network monitoring data $\mathcal{D} = \{(X_t, y_t)\}_{t=1}^{T}$
Large language model $\mathcal{L}$ with prompt templates $\mathcal{P}$
Model pool $\mathcal{M} = \{M_1, ..., M_m\}$
Dual-memory system: $Mem_{short} = \{F_{current}, T_{range}, P_{model}, M_{config}\}$
$Mem_{long} = \{F_{stats}, M_{history}, E_{record}, A_{track}\}$
Ensure: Optimized ensemble model $M_{ensemble}$, Updated memory system

function InitializeSystem($\mathcal{D}, \mathcal{M}, Mem_{long}$)
  $D_{train} \leftarrow LLM(\mathcal{D}, Mem_{long}, \mathcal{P}_{select})$
    ▷ Knowledge-driven data selection
  $(M_{selected}, \Theta, S) \leftarrow LLM(\mathcal{M}, Mem_{long}, \mathcal{P}_{config})$
    ▷ Model configuration and ensemble strategy
  $M_{ensemble} \leftarrow TrainEnsemble(M_{selected}, \Theta, S, D_{train})$
  return $M_{ensemble}, D_{train}$
end function
function ContinuousLearning($M_{ensemble}, \mathcal{D}_{new}$)
  while not converged **do**
    $P_t \leftarrow Predict(M_{ensemble}, X_t)$
      ▷ Generate failure predictions
    $E_t \leftarrow Evaluate(P_t, y_t)$
      ▷ Assess prediction performance
    // Feedback generation and analysis
    $f_t \leftarrow LLM(E_t, Mem_{long}, \mathcal{P}_{analyze})$
      ▷ Generate structured feedback
    $I_t \leftarrow LLM(f_t, F_{history}, \mathcal{P}_{improve})$
      ▷ Develop improvement strategies
    $V_t \leftarrow LLM(I_t, F_{history}, \mathcal{P}_{validate})$
      ▷ Validate proposed strategies
    // Memory system operations
    $UpdateMemory(Mem_{short}, f_t, E_t, P_t)$
      ▷ Update short-term memory
    $TransferMemory(Mem_{short}, Mem_{long})$
      ▷ Transfer valuable experiences
    $CleanupMemory(A_{track}, \tau_{active})$
      ▷ Remove inactive entries
    // System optimization
    $C_{t+1} \leftarrow Optimize(C_t, V_t, \mathcal{P}_{refine})$
      ▷ Apply validated improvements
    $M_{ensemble}.update(C_{t+1}, \mathcal{D}_{new})$
      ▷ Update ensemble model
    // Activation tracking
    $A_{track} \leftarrow UpdateActivation(A_{track}, f_t)$
      ▷ Update memory activation status
  end while
  return $M_{ensemble}, Mem_{short}, Mem_{long}$
end function
Main process
$M_{ensemble}, D_{train} \leftarrow$ InitializeSystem($\mathcal{D}, \mathcal{M}, Mem_{long}$)
$M_{ensemble}, Mem_{short}, Mem_{long} \leftarrow$ ContinuousLearning($M_{ensemble}, \mathcal{D}_{new}$)
return $M_{ensemble}, Mem_{short}, Mem_{long}$

TABLE VI
Model Pool Configuration Options with LLM Selection Examples

Model Architecture Parameters	
Model Type	**Parameter Range (Example Choice)**
MLP	Layers: [3-6] (4) Units: [64,128,256,512] (128) Dropout: [0.1-0.5] (0.3) Width decay ratio: [1.5-3.0] (2.0)
CNN	Conv layers: [1-5] (3) Filters: [32,64,128] (64) Kernel size: [3,5,7] (3) Pooling: [Max, Average] (Max)
LSTM	Layers: [1-8] (4) Hidden units: [64,128,256,512] (128) Dropout: [0.1-0.5] (0.4) Sequence length: [10-50] (30)
GRU	Layers: [1-4] (3) Hidden units: [64,128,256,512] (256) Dropout: [0.1-0.5] (0.3) Update gate bias: [-2.0,2.0] (1.0)
Transformer	Layers: [2-8] (6) Embedding dim: [128,256,512] (256) Dropout: [0.1-0.5] (0.5) Attention heads: [4,8,16] (8)
Training Parameters	
Batch size: [32-512] (64) Learning rate: [1e-5, 1e-4, 1e-3] (5e-4) Optimizer: AdamW with weight decay [1e-5, 1e-3] (1e-4) Early stopping patience: [5-20 epochs] (10)	