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 19.1% 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 19.1% 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 and Self-Improvement

Recent advances in LLM capabilities have demonstrated their potential in complex reasoning and self-improvement tasks. These developments can be examined from two perspectives:

1) Reasoning and Verification: LLMs have shown remarkable capabilities in complex reasoning tasks, though with certain limitations that require careful consideration. Chen et al. [18] established chain-of-thought prompting as a fundamental technique for enhancing reasoning abilities, while Wang et al. [19] advanced this through a self-consistency framework that validates multiple reasoning paths. An et al. [26] further demonstrated that LLMs can learn from their mistakes to improve reasoning capabilities, particularly when provided with structured feedback. However, as Tyen et al. [16] revealed, while LLMs excel at correction with guidance, they struggle with autonomous error detection.

Recent work has focused on addressing these limitations through various approaches. Weng et al. [15] developed self-verification mechanisms that significantly improve reasoning reliability. Gou et al. [28] proposed CRITIC, a framework enabling LLMs to self-correct through tool-interactive critiquing. These advances in self-improvement capabilities are particularly relevant for complex system monitoring and failure prediction tasks, where accurate reasoning about system states is crucial.

2) Continuous Learning and Adaptation: The development of continuous learning capabilities in LLMs has seen significant progress, particularly in their ability to adapt and improve over time. Kumar et al. [23] introduced a self-improving framework where LLMs generate and evaluate their own optimization strategies. This work was complemented by Gao et al. [27], who developed a lifelong learning approach enabling autonomous experiential learning in LLMs.

Several frameworks have emerged to enhance LLMs' adaptive capabilities. Qiao et al. [29] demonstrated improved tool learning through execution feedback, while Shinn et al. [30] introduced Reflexion, a framework for verbal reinforcement learning in language agents. These advances in continuous learning and adaptation are particularly relevant for maintaining prediction accuracy in dynamic environments.

C. LLM Applications in Time Series and Agent Systems

The application of LLMs to time series analysis and agent-based systems represents a growing area of research, with particular relevance to failure prediction tasks.

1) Time Series Analysis and Forecasting: Recent work has demonstrated the effectiveness of LLMs in time series analysis tasks. Gruver et al. [36] showed that LLMs can perform zero-shot time series forecasting, while Jin et al. [37] developed Time-LLM, demonstrating how LLMs can be reprogrammed for time series forecasting tasks. Cao et al. [34] introduced TEMPO, a prompt-based framework specifically designed for time series prediction.

Several approaches have focused on enhancing LLMs' capabilities in handling temporal data. Chang et al. [35] developed

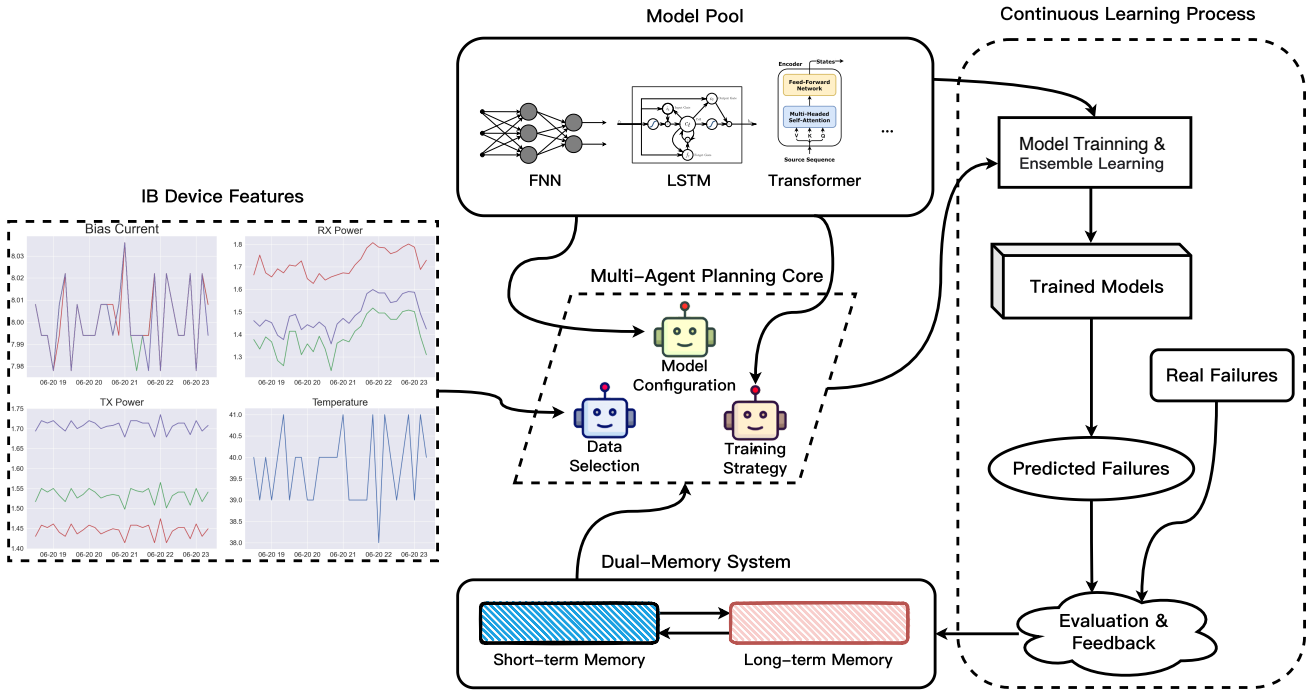

Fig. 1. Overview of SALCA-IB architecture. The system integrates (a) a model pool containing diverse deep learning models, (b) a multi-agent planning core for intelligent orchestration, (c) a dual-memory system for knowledge retention, and (d) a continuous learning process for adaptive optimization.

methods for aligning pre-trained LLMs as data-efficient time series forecasters, while Zhou et al. [40] demonstrated a unified approach for general time series analysis using pre-trained language models. Xue et al. [39] proposed PromptCast, introducing a new prompt-based learning paradigm specifically for time series forecasting.

2) Multi-Agent and Workflow Systems: The integration of LLMs into multi-agent and workflow systems has emerged as a promising direction for complex task management. Wang et al. [31] demonstrated that mixture-of-agents approaches can enhance LLM capabilities, particularly in handling complex, multi-step tasks. Wang et al. [32] introduced agent workflow memory, addressing the challenge of maintaining consistent state across extended operations.

These advances in agent-based systems have been particularly impactful in specific domains. Seff et al. [38] showed how LLM-based approaches can be applied to multi-agent motion forecasting, demonstrating the potential for LLMs to handle complex, multi-entity prediction tasks. These developments in agent-based systems and workflow management are particularly relevant for orchestrating complex failure prediction systems that must coordinate multiple components and adapt to changing conditions.

While these advances demonstrate significant progress in both LLM capabilities and their applications, their potential for 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. 2, 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. 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. 2.

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:

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

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

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

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:

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

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:

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

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:

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

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:

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

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:

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

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:

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

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:

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

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:

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

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:

$$M^*_{ensemble} = f^{train}_{LLM}(M_{ensemble}, \mathcal{D}_{train}, \mathcal{K}) \tag{11}$$

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:

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

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:

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

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:

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

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

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

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

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

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:

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

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:

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

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

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	70.2±2.2	67.1±2.5	68.6±2.4
SALCA-IB (Yi-1.5-34B)	83.2±0.9	79.5±1.1	81.3±1.0	77.5±1.8	74.2±2.1	75.8±2.0
SALCA-IB (Llama-3-70B)	84.5±0.8	80.8±1.0	82.6±0.9	79.6±1.7	76.5±2.0	78.0±1.9
SALCA-IB (GPT-4o)	**87.2±0.7**	**83.9±0.9**	**85.5±0.8**	**84.7±1.6**	**81.6±1.9**	**83.1±1.8**

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 III 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.

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 Scenarios: The experimental evaluation, conducted on our 8-month dataset, encompasses two distinct scenarios to comprehensively assess system performance. The static scenario evaluates models within a fixed 2-week time window, utilizing 70

For dynamic scenario evaluation, we selected a continuous 16-week period from the dataset, where significant maintenance events naturally occurred. The system was initially trained on the first 2-week data segment, followed by continuous evaluation over the subsequent 14 weeks. Two major maintenance events were recorded during this period: a Week

6 maintenance involving optical module replacements and firmware updates, and a Week 12 maintenance encompassing topology adjustments and switch configuration optimization. These events created natural feature distribution shifts, allowing evaluation of the system's adaptability to environmental changes.

2) Evaluation Metrics: The performance assessment employs ranking-based metrics with K set to 2% of total instances. Precision@K measures the proportion of true positives among predicted positives in top-K predictions. Recall@K quantifies the ratio of detected true positives to actual positives within top-K predictions. F1@K represents the harmonic mean of Precision@K and Recall@K, providing a balanced measure of prediction performance.

Statistical robustness is ensured through five independent experimental runs with different random seeds. Results are presented as mean values with standard deviations, and statistical significance is validated using paired t-tests (p ¡ 0.05). The system is evaluated on its ability to provide failure predictions within a 30-minute advance warning window, a critical requirement for practical deployment in production environments.

D. 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 performance degradation (Transformer: -13.0%, Weighted Voting: -16.4%), SALCA-IB (GPT-4o) maintains remarkably stable performance with an F1@K score of 83.1±1.8%, representing only a 2.4% decrease from its static scenario performance. This exceptional stability can be attributed to our fusion memory system, which effectively leverages both short-term memory for rapid adaptation and long-term memory for maintaining consistent performance baselines. The effectiveness of this dual-memory approach is consistently observed across different LLM scales, with even the 34B variant (75.8±2.0%) significantly outperforming all baseline methods.

TABLE II
ABLATION STUDY RESULTS

Model Variant	**Static F1@K (%)**	**Dynamic F1@K (%)**
SALCA-IB (Full)	**85.5±0.8**	**83.1±1.8**
w/o LLM Planning	79.8±1.3	67.2±2.4
w/o Dual Memory	81.3±1.2	73.5±2.2
w/o Continuous Learning	82.9±1.0	76.8±2.0
Weighted Voting	80.4±1.0	64.0±2.3

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, ±1.8% 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 maintains high performance levels through continuous adaptation and optimization.

E. 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 15.9% 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 67.2% in dynamic scenarios) is comparable to or slightly better 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.

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 9.6% performance degradation in dynamic scenarios. Interestingly, comparing the variants without memory (73.5%) and without continuous learning (76.8%) reveals an important insight: the absence of memory leads to a larger performance drop despite the presence of continuous learning. This suggests

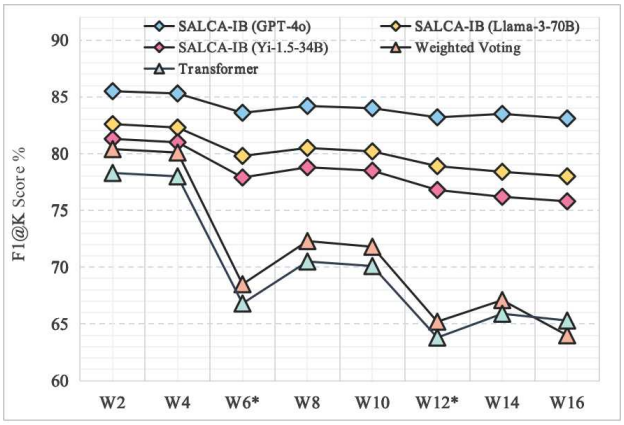

Fig. 2. Long-term performance analysis of SALCA-IB. Week 6 and Week 12 are the two maintenance events.

that the memory system provides crucial context that enables LLM to make more informed adaptations.

The continuous learning mechanism shows its value in sustaining prediction performance over time. Without this component, the system's F1@K score decreases by 6.3% 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 (83.1% vs 64.0%) scenarios, demonstrating the overall effectiveness of our integrated approach.

F. Long-term Performance Analysis

To evaluate the system's robustness in real-world scenarios, we conducted a 16-week longitudinal study during which two major equipment maintenance events occurred (Week 6 and Week 12). As shown in Table **??**, these maintenance activities significantly impacted the feature distributions and revealed striking differences in how various methods handled such environmental changes. During the first maintenance (Week 6), traditional methods experienced severe performance degradation, with Weighted Voting dropping by 11.6% (80.1% to 68.5%) and Transformer declining by 11.2% (78.0% to 66.8%). In contrast, SALCA-IB (GPT-4o) demonstrated remarkable resilience, showing only a 3.1% decrease (85.3% to 82.4%). This substantial difference in impact highlights the effectiveness of our LLM-driven planning core in adapting to feature distribution shifts.

Even more revealing is the comparison between the first and second maintenance events. During the second maintenance (Week 12), SALCA-IB (GPT-4o) showed further improved resilience with only a 1.1% performance drop (82.9% to 81.8%), significantly smaller than its first-event impact. This enhanced adaptability can be attributed to our dual-memory system, which effectively retained and leveraged the experience from the first maintenance event. The scale of LLM cores also played a crucial role in system resilience, as evidenced by the performance variations across different models - GPT-4o's drops (3.1% and 1.1%) were notably smaller than those of Llama-3-70B (4.2% and 2.4%) and Yi-1.5-34B (4.8% and 2.9%). These results collectively demonstrate SALCA-IB's robust performance in dynamic production environments, particularly through its effective combination of LLM-driven planning and experience-based adaptation.

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

[1] T. Ahmed, S. Ghosh, C. Bansal, T. Zimmermann, X. Zhang, and S. Rajmohan, "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] A. Cascajo, G. Gomez-Lopez, J. Escudero-Sahuquillo, P. J. Garcia, D. E. Singh, F. Alfaro-Cortés, F. J. Quiles, and J. Carretero, "Monitoring infiniband networks to react efficiently to congestion," IEEE Micro, vol. 43, no. 2, pp. 120-130, 2023.

[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, M. Hutton, S. Chan, R. De Gruijl, R. Govindaraju, N. Patil, and Y. Li, "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, Z. Li, C. Tan, R. Yang, G. Cao, Z. Liu, and C. Guo, "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, J. Xue, S. Gupta, T. Patel, C. Engelmann, E. Smirni, and D. Tiwari, "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, C. Jiang, X. Jin, Y. Qiao, T. Xiao, H. Ma, R. Wei, Z. Jing, J. Xu, and J. Lin, "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, S. Zhang, Y. Zhang, H. Zhang, P. Zhao, and Q. Jin, "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, A. Kumar, S. Basu, S. Maji, V. Balasubramanian, and P. Liang, "HierarchicalMem: An Adaptive Memory System for Large Language Models," in Proc. EMNLP 2023.
[21] J. Li, Z. Yang, X. Jiang, Y. Luo, S. Zhang, and J. Yang, "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, K. Wang, Q. Chen, Z. Yang, J. Deng, and D. Wang, "Iterative Refinement of Large Language Model Outputs with Feedback Loops," in Proc. ACL 2023.
[25] Z. Zhao, R. Liu, K. Chen, D. Wei, W. Shi, and D. Song, "Meta-Learning for Adaptive Large Language Models," in Proc. ICML 2023.
[26] S. An, Z. Ma, Z. Lin, N. Zheng, J.-G. Lou, and W. Chen, "Learning from mistakes makes LLM better reasoner," arXiv preprint arXiv:2310.20689, 2024.
[27] J. Gao, X. Ding, Y. Cui, J. Zhao, H. Wang, T. Liu, and B. Qin, "Self-evolving GPT: A lifelong autonomous experiential learner," arXiv preprint arXiv:2407.08937, 2024.
[28] Z. Gou, Z. Shao, Y. Gong, Y. Shen, Y. Yang, N. Duan, and W. Chen, "CRITIC: Large language models can self-correct with tool-interactive critiquing," arXiv preprint arXiv:2305.11738, 2024.
[29] S. Qiao, H. Gui, C. Lv, Q. Jia, H. Chen, and N. Zhang, "Making language models better tool learners with execution feedback," arXiv preprint arXiv:2305.13068, 2024.
[30] N. Shinn, F. Cassano, E. Berman, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," arXiv preprint arXiv:2303.11366, 2023.
[31] J. Wang, J. Wang, B. Athiwaratkun, C. Zhang, and J. Zou, "Mixture-of-agents enhances large language model capabilities," arXiv preprint arXiv:2406.04692, 2024.
[32] Z. Z. Wang, J. Mao, D. Fried, and G. Neubig, "Agent workflow memory," arXiv preprint arXiv:2409.07429, 2024.
[33] S. Yao et al., "ReAct: Synergizing reasoning and acting in language models," arXiv preprint arXiv:2210.03629, 2023.
[34] D. Cao, F. Jia, S. O. Arik, T. Pfister, Y. Zheng, W. Ye, and Y. Liu, "TEMPO: Prompt-based generative pre-trained transformer for time series forecasting," arXiv preprint arXiv:2310.04948, 2024.
[35] C. Chang, W.-Y. Wang, W.-C. Peng, and T.-F. Chen, "LLM4TS: Aligning pre-trained LLMs as data-efficient time-series forecasters," arXiv preprint arXiv:2308.08469, 2024.
[36] N. Gruver, M. Finzi, S. Qiu, and A. G. Wilson, "Large language models are zero-shot time series forecasters," in Advances in Neural Information Processing Systems, vol. 36, 2023, pp. 19622-19635.
[37] M. Jin et al., "Time-LLM: Time series forecasting by reprogramming large language models," arXiv preprint arXiv:2310.01728, 2024.
[38] A. Seff et al., "MotionLM: Multi-agent motion forecasting as language modeling," in Proc. IEEE/CVF Int. Conf. Computer Vision (ICCV), 2023, pp. 8579-8590.
[39] H. Xue and F. D. Salim, "PromptCast: A new prompt-based learning paradigm for time series forecasting," arXiv preprint arXiv:2210.08964, 2023.
[40] T. Zhou, P. Niu, X. Wang, L. Sun, and R. Jin, "One fits all: Power general time series analysis by pretrained LM," in Advances in Neural Information Processing Systems, vol. 36, 2023, pp. 43322-43355.

V. 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 III
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)	