Master's Thesis

Microservice Anomaly Detection and Root Cause Analysis Using Temporal Graph Networks with Adaptive Memory Protection

2410014 Tomoya Ueno

Supervisor	BEURAN Razvan, Associate Professor
Committee	TAN, Yasuo, Professor
	LIM, Yuto, Professor
	UDA, Satoshi, Associate Professor

Japan Advanced Institute of Science and Technology
Graduate School of Advanced Science and Technology
(Information Science)

March 2026

Abstract

Microservice architecture has become a standard design pattern in modern large-scale distributed systems. Hundreds to thousands of independent services collaborate, forming chains of Remote Procedure Calls (RPCs) where a single user request propagates across multiple services. While this architecture provides development flexibility and scalability, it significantly increases the complexity of operational monitoring. Local failures can rapidly cascade throughout the entire system. For example, a memory leak or CPU anomaly in one service can cause latency increases in dependent upstream services, ultimately leading to degraded user experience or system downtime. In such environments, it is crucial not only to detect anomalies quickly but also to identify the true root cause.

Traditional threshold-based monitoring and univariate time series analysis cannot account for complex inter-service dependencies and tend to generate many false positives. This leads to alert fatigue among operators and makes rapid identification of the true root cause difficult. To address this challenge, Graph Neural Networks (GNNs) that explicitly handle service dependencies (topology) have gained attention. In particular, Temporal Graph Networks (TGN), equipped with memory mechanisms that store long-term state for each node, possess an ideal model structure for learning the evolution of normal service behavior.

In this study, we identified the **memory contamination problem in GRU-based TGN** as a critical challenge for anomaly detection. The original TGN was designed for dynamic graph learning tasks such as link prediction and node classification, unconditionally incorporating all input events into GRU memory. In the context of anomaly detection, this design causes anomalous data to accumulate in memory, corrupting the normal profile and leading to increased false positives and missed detections.

Additionally, microservice environments frequently exhibit **partial anomalies**, where normal and anomalous metrics coexist within a single node—for example, CPU metrics may be anomalous while latency remains normal. Existing approaches lack fine-grained control over memory updates at the metric level, making it difficult to selectively suppress memory updates for anomalous metric groups while maintaining normal updates for unaffected groups.

To address these challenges, we propose **ADA-TGN (Temporal Graph Networks with Adaptive Memory Protection)**. The contributions of this study are threefold. First, we identified the memory contamination problem in GRU-based TGN and clarified the fundamental challenge in memory-based anomaly detection. Second, we introduced a **Dual-Stage Soft-Gating mechanism** to achieve adaptive memory protection that selectively suppresses memory updates during anomalies. Third, we enabled independent control of memory updates at the metric group level (Latency,

Memory, CPU, Network), allowing continued memory updates for normal metric groups even during partial anomalies.

The Soft-Gating mechanism of ADA-TGN operates in two stages. In the first stage (GRU Input Soft-Gating), input signals to the GRU are attenuated according to the anomaly degree, suppressing the inflow of anomalous data. In the second stage (Memory Soft-Gating), the memory state after GRU update is interpolated with the previous memory, retaining the previous memory during anomalies. This two-stage protection effectively blocks the influence of anomalous data on memory. For anomaly degree estimation, reconstruction errors for each metric group are used, with adaptive thresholds set based on the interquartile range (IQR). This design enables suppressing memory updates for anomalous metric groups while continuing memory updates for normal metric groups even during partial anomalies.

For feature extraction, 12-dimensional features across 4 metric groups are extracted from microservice metrics data. These features are standardized using P99 scaling and Asinh transformation on a per-service basis. Inter-service communication relationships are represented as graph structure and utilized for neighborhood information aggregation through Graph Attention.

In evaluation experiments, we used the RE2-OB dataset from the RCAEval benchmark. For each experiment, an independent model was trained using only normal period data to construct the normal profile. Through ablation studies, we confirmed that Vanilla-TGN (without Soft-Gating mechanism) experiences memory contamination during anomaly periods, resulting in unstable reconstruction from contaminated memory. In contrast, ADA-TGN suppresses memory contamination and maintains stable reconstruction even during anomaly periods.

For anomaly detection performance, the method combining ADA-TGN with dynamic SPOT (dSPOT) achieved Precision 0.923, Recall 1.000, and F1 score 0.960. For root cause analysis performance, ADA-TGN achieved Avg@5 of 0.900, outperforming BARO's 0.745 by 20.8% when evaluated on the same dataset.

This study presents a solution to the memory contamination problem that was unavoidable in TGN-based anomaly detection, and demonstrates improved accuracy in anomaly detection and root cause analysis for microservice environments.

Keywords: Temporal Graph Networks, Microservices, Anomaly Detection, Root Cause Analysis, Memory Contamination, Soft-Gating

Acknowledgments

I would like to express my deepest gratitude to my supervisor, Associate Professor Razvan Beuran, for his invaluable guidance, insightful suggestions, and continuous support throughout this research.

I am also grateful to the committee members, Professor Yasuo Tan, Professor Yuto Lim, and Associate Professor Satoshi Uda, for their constructive feedback and valuable suggestions that helped improve this thesis.

Finally, I would like to thank my family and friends for their unwavering encouragement and support.

In addition, I acknowledge the use of generative AI tools during the preparation of this thesis. Specifically, I used Gemini 3 (Google) for brainstorming ideas and refining research concepts, and Claude Code (v2.1.29, Anthropic) to assist with the coding and implementation of the proposed method. I have reviewed and verified all generated content and take full responsibility for the final work.

List of Abbreviations

GNN	Graph Neural Network
TGN	Temporal Graph Network
GRU	Gated Recurrent Unit
LSTM	Long Short-Term Memory
MLP	Multi-Layer Perceptron
GELU	Gaussian Error Linear Unit
AD	Anomaly Detection
RCA	Root Cause Analysis
MSA	Microservice Architecture
GT	Ground Truth
TP	True Positive
FP	False Positive
FN	False Negative
IQR	Interquartile Range
EVT	Extreme Value Theory
BOCPD	Bayesian Online Change Point Detection
GPD	Generalized Pareto Distribution
PCA	Principal Component Analysis
CI	Confidence Interval
dSPOT	Streaming Peaks-Over-Threshold with Drift
ADWIN	Adaptive Windowing
BARO	Bayesian Root cause analysis for Observability data
SVDD	Support Vector Data Description
RPC	Remote Procedure Call
SRE	Site Reliability Engineering
AIOps	Artificial Intelligence for IT Operations

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Background and Motivation

The adoption of Microservice Architecture (MSA) has brought unprecedented scalability and development flexibility to modern software systems [8]. In large-scale commercial systems, hundreds to thousands of independent services collaborate, forming chains of Remote Procedure Calls (RPCs) that propagate across multiple services for a single user request [7]. This architecture enables independent development, deployment, and scaling of each service, significantly improving development velocity and system availability.

However, this comes at the cost of substantially increased observability complexity. Local failures can rapidly propagate throughout the entire system, causing cascade failures. For example, a memory leak or CPU anomaly in one service can cause latency increases in dependent upstream services, ultimately leading to degraded user experience or system downtime. In such environments, it is crucial not only to detect anomalies quickly but also to identify the true root cause.

Traditional threshold-based monitoring and univariate time series analysis cannot account for the complex inter-service dependencies, and tend to generate many false positives. This leads to alert fatigue [2] among operators, making rapid identification of the true root cause difficult. To address this challenge, AIOps (Artificial Intelligence for IT Operations) [6], which applies machine learning to IT operations, has gained attention. However, industrial surveys reveal that intelligent trace analysis in microservices is still in its infancy, with machine learning and data mining techniques seldom used in practice [33]. In particular, Graph Neural Networks (GNN) [26], which explicitly handle service dependencies (topology), are expected to be powerful tools for modeling microservice dependencies. However, many existing GNN-based methods treat time series data as static snapshots, lacking the ability to learn long-term temporal patterns.

In contrast, Temporal Graph Networks (TGN) [21], which feature a "memory mechanism" that stores long-term state for each node, have an

ideal model structure for learning the evolution of normal service behavior. TGN integrates graph structure and temporal information, enabling each service (node) to maintain a memory vector that compresses and retains long-term historical information. However, when applying TGN to anomaly detection, there are critical challenges described below.

1.2 Challenges with Existing Approaches

1.2.1 TGN Memory Contamination Problem

The original TGN was designed for dynamic graph learning tasks such as link prediction and node classification, and unconditionally incorporates all input events into memory. In the context of anomaly detection, this design causes the **memory contamination problem in GRU-based TGN**, where anomalous data accumulates in memory and corrupts the normal profile. Once memory is contaminated, sensitivity to subsequent anomalies is lost (False Negative), and false positives occur after returning to normal operation.

This thesis formalizes the memory contamination problem in Chapter 3 and proposes a solution using a Soft-Gating mechanism. The memory contamination problem and existing defense methods are detailed in Section 2.2.

1.2.2 Partial Anomaly Problem

In microservices, "partial anomalies" frequently occur where only some features are anomalous while others remain normal, such as CPU being anomalous while latency is normal. Existing approaches such as MemStream [3] lack fine-grained control over memory updates at the metric level, making it difficult to selectively suppress memory updates for anomalous metric groups while maintaining normal updates for unaffected groups.

In this research, we address this **partial anomaly problem** by classifying metrics into groups by type (Latency, Memory, CPU, Network) and performing independent anomaly estimation and memory control for each group.

1.3 Research Objectives and Contributions

The objective of this research is to solve the GRU-based TGN memory contamination problem and achieve high-accuracy anomaly detection and root cause analysis in microservice systems. The main contributions are as follows:

1. **Formalization of the Memory Contamination Problem**: We identified and formalized the memory contamination problem in GRU-

based TGN for anomaly detection, clarifying the fundamental challenge that anomalous data accumulates in memory and corrupts the normal profile.

2. **Soft-Gating Mechanism for Adaptive Memory Protection**: We proposed a Dual-Stage Soft-Gating mechanism that continuously suppresses memory updates during anomalies while maintaining update capability during normal periods.

3. **Group-wise Independent Control for Partial Anomalies**: We enabled independent memory control at the metric group level (Latency, Memory, CPU, Network), allowing continued memory updates for normal groups even during partial anomaly situations.

4. **Experimental Validation**: We demonstrated the effectiveness of ADA-TGN through comprehensive experiments using the RE2-OB dataset, achieving improvements in both anomaly detection (F1: 0.960) and root cause analysis (Avg@5: 0.900).

The implementation of ADA-TGN is publicly available at https://github.com/cyb3rlab/ADA-TGN.

1.4 Thesis Organization

This thesis is organized as follows. Chapter 2 reviews related work and clarifies the positioning of this research. Chapter 3 formalizes the memory contamination problem. Chapter 4 explains the architecture of the proposed ADA-TGN and the Soft-Gating mechanism. Chapter 5 describes the dataset, feature extraction, and preprocessing methods. Chapter 6 presents experimental results and discussion. Finally, Chapter 7 provides conclusions and future work. Appendix A presents supplementary analysis on how GRU parameter training affects memory contamination progression dynamics.

Chapter 2

Related Work

This chapter reviews prior research on Temporal Graph Networks, existing defense methods against memory contamination, and root cause analysis in microservices, which form the foundation of this research.

2.1 Temporal Graph Networks

To handle dynamic graph structures and temporal information in an integrated manner, Rossi et al. [21] proposed Temporal Graph Networks (TGN). TGN treats interactions between graph nodes as temporal events and maintains a persistent memory vector for each node, enabling compressed retention of long-term history. RNN-based units are used for memory updates, and this research adopts GRU (Gated Recurrent Unit). Details are described in Chapter 3.

TGN was originally proposed for link prediction and node classification tasks. However, the original TGN unconditionally incorporates all input events into memory and lacks a mechanism to distinguish anomalous data. In the context of anomaly detection, this design becomes problematic because input data contains "noise (anomalies)" that should not be learned.

2.2 Memory Contamination Problem and Existing Defense Methods

The vulnerability of TGN's memory mechanism has been increasingly revealed in recent research. This section reviews existing defense methods against the memory contamination problem and clarifies the characteristics and limitations of each approach.

2.2.1 Hard Threshold Filtering

T-Shield [14] is a robust training method for TGN that combines Edge Filtering and Temporal Smoothness. While T-Shield employs a learned edge scorer to identify potentially adversarial edges during training, it operates without explicit labels of attacks, using cosine annealing to adaptively adjust filtering thresholds throughout the training process. However, T-Shield was primarily designed for **link prediction** tasks rather than unsupervised anomaly detection scenarios.

MemStream [3] is a streaming anomaly detection method that maintains a **FIFO queue** of recent observations and prevents contamination by not adding data exceeding anomaly score thresholds to memory. Unlike TGN's implicit memory (learned hidden states), MemStream stores learned latent representations via a Denoising Autoencoder (DAE) in a FIFO queue, enabling exact retrieval but lacking the compressed temporal pattern learning capability of RNN-based memory.

Despite their different designs, these methods share common limitations:

- **Hard Threshold Limitation**: Using binary decisions (reject/accept) causes abrupt transitions near the threshold, lacking smooth control.
- **Inability to Handle Partial Anomalies**: Processing entire nodes uniformly cannot handle microservice-specific situations where "only some metrics are anomalous."

2.2.2 Loss Function-Based Approaches

TGN-SVDD [16] is a method that incorporates robustness at the loss function level. It integrates the hypersphere loss of Deep SVDD into TGN's learning objective, suppressing the influence of anomalous data during training. As with typical one-class classification methods, the model lacks flexibility for adaptation to changing patterns during inference. Also, since node embeddings as a whole are placed on the hypersphere, these methods do not support independent control for each metric group as required in microservice environments.

2.2.3 Our Approach

To overcome the limitations identified above, this research proposes a Soft-Gating mechanism with two key innovations. First, unlike hard threshold methods (Section 2.2.1), our approach employs continuous control via Sigmoid function, enabling smooth transitions near decision boundaries rather than abrupt reject/accept decisions. Second, unlike existing approaches that process nodes uniformly, our mechanism enables group-wise independent control, allowing selective memory protection for anomalous metric groups while maintaining updates for normal groups.

Details of the proposed method, including the Dual-Stage architecture and group-wise independent control, are described in Chapter 4.

2.3 Root Cause Analysis in Microservices

In handling failures in microservice systems, it is important not only to detect anomalies but also to quickly identify their root cause. Various graph-based RCA methods have been proposed for microservice environments.

MicroRCA [25] constructs an attributed graph from service call relationships and metric anomaly scores, then applies Personalized PageRank to identify the root cause by tracing anomaly propagation paths.

This method incorporates sophisticated causal reasoning to distinguish root causes from symptoms. In contrast, this research focuses primarily on **anomaly detection with memory protection**, employing a simpler RCA approach based on anomaly score ranking. Integrating more advanced RCA techniques such as graph-based causal analysis remains an important direction for future work (discussed in Chapter 7).

Chapter 3

Problem Formulation

This chapter theoretically formalizes the **memory contamination problem** in memory-based time series models. First, we explain the structure of memory-based time series models (particularly GRU) (Section 3.1), then mathematically formalize the memory contamination problem (Section 3.2). Finally, we describe the direction for solution (Section 3.3).

3.1 Memory-Based Time Series Models

This section explains the general structure of memory-based models for processing time series data. In particular, we focus on the GRU (Gated Recurrent Unit) [5]-based memory update mechanism adopted in Temporal Graph Networks (TGN) [21]. GRU is a simplified architecture of LSTM (Long Short-Term Memory) [13], enabling learning of long-term dependencies through two gate mechanisms: the update gate and reset gate. While LSTM has three gates (input, forget, output) and a cell state, GRU consists of only two gates, resulting in fewer parameters and better computational efficiency. Cho et al. [5] reported that GRU achieves performance comparable to LSTM on many tasks, and GRU is adopted in the reference implementation of TGN [20, 21].

3.1.1 GRU Memory Update Mechanism

In memory-based time series models, each node v has a memory vector $\mathbf{m}_v^{(t)} \in \mathbb{R}^D$ that retains temporal patterns. TGN [21] adopts GRU (Gated Recurrent Unit) as the memory update function, with the following update equations:

$$\mathbf{r}_t = \sigma(\mathbf{W}_{ir}\mathbf{x}_t + \mathbf{b}_{ir} + \mathbf{W}_{hr}\mathbf{h}_{t-1} + \mathbf{b}_{hr}) \tag{3.1}$$

$$\mathbf{z}_t = \sigma(\mathbf{W}_{iz}\mathbf{x}_t + \mathbf{b}_{iz} + \mathbf{W}_{hz}\mathbf{h}_{t-1} + \mathbf{b}_{hz}) \tag{3.2}$$

$$\tilde{\mathbf{n}}_t = \tanh(\mathbf{W}_{in}\mathbf{x}_t + \mathbf{b}_{in} + \mathbf{r}_t \odot (\mathbf{W}_{hn}\mathbf{h}_{t-1} + \mathbf{b}_{hn})) \tag{3.3}$$

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \tilde{\mathbf{n}}_t + \mathbf{z}_t \odot \mathbf{h}_{t-1} \tag{3.4}$$

where:

- $\mathbf{r}_t \in [0,1]^D$: **Reset Gate**
- $\mathbf{z}_t \in [0,1]^D$: **Update Gate**
- $\tilde{\mathbf{n}}_t \in \mathbb{R}^D$: **Candidate Hidden State**
- $\mathbf{h}_t \in \mathbb{R}^D$: **Hidden State**
- $\mathbf{W}_*, \mathbf{b}_*$: Learned parameters (weight matrices, biases)
- $\sigma(\cdot)$: Sigmoid function, $\odot$: Element-wise product

3.1.2 Separation of Weights and Hidden State

In GRU, **weights** $\mathbf{W}_*, \mathbf{b}_*$ are optimized during training and fixed during inference, while **hidden state** $\mathbf{h}_t$ changes dynamically depending on input data even during inference. This property is the cause of the memory contamination problem (detailed in Section 3.2).

3.1.3 Equivalence of TGN Memory and GRU Hidden State

In TGN, each node v's memory $\mathbf{m}_v^{(t)}$ is **technically equivalent** to GRU's hidden state $\mathbf{h}_t$:

$$\mathbf{m}_v^{(t)} = \mathrm{GRU}(\mathbf{x}_v^{(t)}, \mathbf{m}_v^{(t-1)}) = \mathbf{h}_t \tag{3.5}$$

In TGN, GRU weights $\mathbf{W}_*, \mathbf{b}_*$ are shared across all nodes, but memory vector $\mathbf{m}_v^{(t)}$ is maintained independently by each node. When a new observation is input to node v, only that node's memory is updated using the shared GRU. Due to this structure, properties of GRU's hidden state (vulnerability to noise, retention of long-term dependencies, etc.) directly apply to each node's memory in TGN.

3.2 Memory Contamination Problem

This section formalizes the **memory contamination problem** in memory-based time series models. This problem occurs at two levels: GRU hidden state level and TGN memory level.

3.2.1 GRU Hidden State Contamination Mechanism

3.2.1.1 Hidden State Contamination by Anomalous Input

During inference, GRU weights $\mathbf{W}_*, \mathbf{b}_*$ are fixed, but hidden state $\mathbf{h}_t$ changes dynamically depending on input data. When anomalous data $\mathbf{x}_{\text{anom}}$ is input, the following chain occurs:

1. **Gate Value Anomalization**: In Equations (3.1)(3.2), anomalous input $\mathbf{x}_{\text{anom}}$ causes gate values $\mathbf{r}_t, \mathbf{z}_t$ to take values different from normal.

2. **Candidate State Anomalization**: In Equation (3.3), anomalous input and gate values cause candidate hidden state $\tilde{\mathbf{n}}_t$ to transition to an anomalous vector space.

3. **Hidden State Contamination**: Through Equation (3.4), the contaminated candidate state is incorporated into hidden state $\mathbf{h}_t$.

4. **Contamination Persistence**: The contaminated $\mathbf{h}_t$ becomes input for the next timestep, causing the effect to persist.

3.3 Direction for Solution

To solve the memory contamination problem formalized in the previous section, a **conditional memory update** mechanism that selectively suppresses memory updates by anomalous data is necessary.

Specifically, the following requirements must be met:

- Dynamically control memory update rate based on anomaly estimation from reconstruction error
- Ensure stability in boundary regions through continuous control rather than hard thresholds
- Implement multi-stage protection mechanisms to prevent gradual accumulation of contamination in memory

The next chapter explains the specific design of the **Soft-Gating mechanism** that meets these requirements and the architecture of the proposed ADA-TGN.

Chapter 4

Proposed Method: ADA-TGN

This chapter presents the proposed method **ADA-TGN**, which addresses the memory contamination problem formalized in Chapter 3. ADA-TGN retains TGN's dynamic adaptation capability while preventing memory contamination by anomalous data through the Soft-Gating mechanism.

4.1 Architecture Overview

ADA-TGN is an architecture based on Temporal Graph Networks (TGN) [21], integrated with a Group-wise Soft-Gating mechanism. Figure 4.1 illustrates the overall structure of ADA-TGN.

4.1.1 Processing Flow

ADA-TGN performs the following processing at each time step t. A critical design principle is that **reconstruction uses only the old memory (information from time $t-1$)**, while the current input $\mathbf{x}^{(t)}$ is used solely for memory update in the subsequent time step. This design follows the original TGN implementation [20, 21] and prevents data leakage in online anomaly detection.

Anomaly Detection Path (Steps ①–④, detailed in Section 4.3):

1. **GNN (Graph Attention)**: Apply Graph Attention to the old memory to generate embeddings that incorporate inter-service dependencies.

2. **Group-wise Decoder**: Partition the embedding by feature groups (Latency, Memory, CPU, Network) and obtain reconstructions using group-specific decoders.

3. **Anomaly Score Computation**: Compute feature-wise reconstruction errors between observed and reconstructed values.

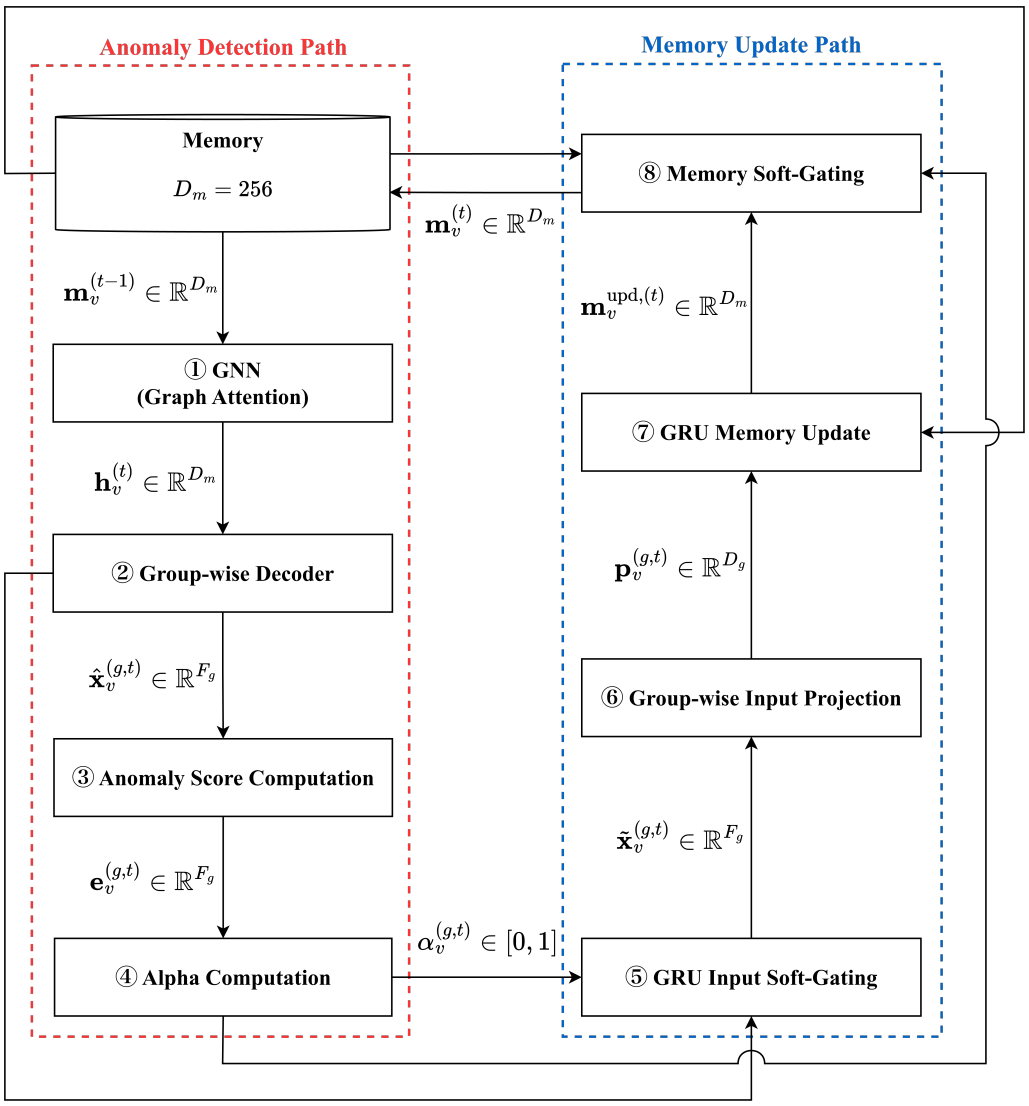


Figure 4.1: ADA-TGN architecture with Dual Soft-Gating mechanism. Blue dashed line: Memory Update Path; Red dashed line: Anomaly Detection Path. The observed value $\mathbf{x}_v^{(g,t)}$ is used as input to both ③ Anomaly Score Computation and ⑤ GRU Input Soft-Gating.

4. **Alpha Computation**: Standardize reconstruction errors using IQR and transform them to Alpha values $\alpha \in [0, 1]$ via the Sigmoid function.

Memory Update Path (Steps ⑤–⑧, detailed in Section 4.4):

5. **GRU Input Soft-Gating (Stage 1)**: Based on Alpha values, compute a weighted mixture of observed and reconstructed values as input to the GRU.

6. **Group-wise Input Projection**: Project the gated input by feature groups into the memory dimension.

7. **GRU Memory Update**: Concatenate projection vectors and generate candidate memory through the shared GRU.

8. **Memory Soft-Gating (Stage 2)**: Based on Alpha values, blend the candidate memory with the old memory to perform protected memory updates.

Algorithm 4.1 presents the pseudocode for processing a single time step.

Algorithm 4.1 ADA-TGN Forward Pass (Single Time Step with Dual-Stage Soft-Gating)

Input: Input features $\mathbf{x}^{(t)}$, old memory $\mathbf{m}^{(t-1)}$, graph edges $\mathcal{E}$
Output: Reconstruction $\hat{\mathbf{x}}^{(t)}$, anomaly score $\mathbf{err}^{(t)}$, new memory $\mathbf{m}^{(t)}$

// Anomaly Detection Path
$\mathbf{h}^{(t)} \leftarrow \text{TransformerConv}(\mathbf{m}^{(t-1)}, \mathcal{E})$ {① GNN (Graph Attention)}
$\hat{\mathbf{x}}^{(t)} \leftarrow \text{GroupDecoders}(\mathbf{h}^{(t)})$ {② Group-wise Decoder}
$\mathbf{err}^{(t)} \leftarrow (\mathbf{x}^{(t)} - \hat{\mathbf{x}}^{(t)})^2$ {③ Anomaly Score Computation}
$\boldsymbol{\alpha}^{(t)} \leftarrow \text{ComputeAlpha}(\mathbf{err}^{(t)})$ {④ Alpha Computation (Alg. 4.2)}

// Memory Update Path
$\tilde{\mathbf{x}}^{(t)} \leftarrow \boldsymbol{\alpha}^{(t)} \odot \mathbf{x}^{(t)} + (1 - \boldsymbol{\alpha}^{(t)}) \odot \hat{\mathbf{x}}^{(t)}$ {⑤ GRU Input Soft-Gating (Stage 1)}
$\mathbf{p}^{(t)} \leftarrow \text{GroupProjection}(\tilde{\mathbf{x}}^{(t)})$ {⑥ Group-wise Input Projection}
$\mathbf{m}^{\text{upd},(t)} \leftarrow \text{GRU}(\mathbf{p}^{(t)}, \mathbf{m}^{(t-1)})$ {⑦ GRU Memory Update}
$\mathbf{m}^{(t)} \leftarrow \boldsymbol{\alpha}^{(t)} \odot \mathbf{m}^{\text{upd},(t)} + (1 - \boldsymbol{\alpha}^{(t)}) \odot \mathbf{m}^{(t-1)}$ {⑧ Memory Soft-Gating (Stage 2)}

return $\hat{\mathbf{x}}^{(t)}$, $\mathbf{err}^{(t)}$, $\mathbf{m}^{(t)}$

4.2 Group-wise Memory Structure

ADA-TGN partitions TGN memory into regions corresponding to feature groups. This structure enables independent memory control for each feature

group, addressing partial anomalies where only certain features exhibit anomalous behavior.

4.2.1 Memory Partitioning

The memory vector $\mathbf{m}_v \in \mathbb{R}^{256}$ for each node v is partitioned into regions corresponding to four feature groups:

$$\mathbf{m}_v = [\mathbf{m}_v^{\text{Lat}}, \mathbf{m}_v^{\text{Mem}}, \mathbf{m}_v^{\text{CPU}}, \mathbf{m}_v^{\text{Net}}] \tag{4.1}$$

4.2.2 Custom Dimension Allocation

Memory dimensions for each feature group are allocated based on feature characteristics:

- **Latency Group**: $\mathbf{m}_v^{\text{Lat}} \in \mathbb{R}^{48}$ (18.75%)
- **Memory Group**: $\mathbf{m}_v^{\text{Mem}} \in \mathbb{R}^{48}$ (18.75%)
- **CPU Group**: $\mathbf{m}_v^{\text{CPU}} \in \mathbb{R}^{96}$ (37.5%)
- **Network Group**: $\mathbf{m}_v^{\text{Net}} \in \mathbb{R}^{64}$ (25%)

Rationale for Custom Allocation: CPU utilization and network traffic often exhibit spike-like rapid fluctuations, while latency and memory usage exhibit relatively stable variations. These dimension allocations were empirically determined based on the complexity of each metric's temporal patterns.

4.3 Anomaly Detection Path

This section details each component (Steps ①–④) of the Anomaly Detection Path, indicated by the red dashed line in Figure 4.1. This path generates reconstructions from the old memory and computes Alpha values by comparing with observed values.

4.3.1 GNN / Graph Attention (Step ①)

Graph Transformer [22] is applied to the old memory $\mathbf{m}_v^{(t-1)}$ to generate embeddings $\mathbf{h}_v^{(t)}$ that incorporate inter-service dependencies:

$$\mathbf{h}_v^{(t)} = \text{TransformerConv}(\mathbf{m}_v^{(t-1)}, \mathcal{E}) \tag{4.2}$$

Here, $\mathcal{E}$ represents the inter-service call relationships (edge set). TransformerConv uses the PyTorch Geometric [10] implementation, where each

node aggregates memory information from neighboring nodes via the attention mechanism.

Effect of Graph Attention: In microservice environments, call relationships exist between services, where the state of one service affects its neighbors. Through Graph Attention, each service obtains embeddings that aggregate neighboring services' memory information via the attention mechanism. This is expected to enable more context-aware reconstruction by incorporating inter-service dependencies.

4.3.2 Group-wise Decoder (Step ②)

ADA-TGN employs a **Group-wise Decoder**, where each feature group has a dedicated decoder to prevent interference between feature groups.

The embedding $\mathbf{h}_v^{(t)}$ is partitioned by feature groups, and each group-specific decoder produces the reconstruction $\hat{\mathbf{x}}_v^{(g,t)}$:

$$\mathbf{h}_v^{(g,t)} = \text{split}_g(\mathbf{h}_v^{(t)}), \quad \hat{\mathbf{x}}_v^{(g,t)} = \text{Decoder}_g(\mathbf{h}_v^{(g,t)}) \tag{4.3}$$

4.3.2.1 Group-wise MLP Decoder

Each feature group has a dedicated MLP decoder that maps the group-specific node embedding to the corresponding feature values:

$$\hat{\mathbf{x}}_v^{(g,t)} = \text{Softplus}(\text{MLP}_g(\mathbf{h}_v^{(g,t)})) \tag{4.4}$$

where $\mathbf{h}_v^{(g,t)}$ is the group-specific portion of the node embedding, and MLP_g consists of two linear layers with RMSNorm normalization and GELU activation.

Design Rationale for MLP Decoder: In the original TGN implementation [20, 21], MLP-based decoders (MergeLayer) are used for edge prediction and node classification tasks. Following this design principle, ADA-TGN uses MLP decoders for feature reconstruction. The node embedding $\mathbf{h}_v^{(g,t)}$ is a latent representation without temporal or spatial structure, making fully-connected layers the natural choice for mapping embeddings to feature values.

The Softplus activation function ensures non-negative outputs, as all target metrics are inherently non-negative.

4.3.3 Anomaly Score Computation (Step ③)

Feature-wise reconstruction errors are computed from the reconstructed features $\hat{\mathbf{x}}_v^{(t)}$ and observed values $\mathbf{x}_v^{(t)}$:

$$e_v^{(i,t)} = (x_v^{(i,t)} - \hat{x}_v^{(i,t)})^2 \tag{4.5}$$

Feature errors are then aggregated by feature group to compute the Group-wise Anomaly Score $s_v^{(g,t)}$. ADA-TGN employs **MAX aggregation**:

$$s_v^{(g,t)} = \max_{i \in \mathcal{I}_g} e_v^{(i,t)} \tag{4.6}$$

Here, $\mathcal{I}_g$ denotes the index set of features belonging to feature group g.

Rationale for MAX Aggregation: MEAN aggregation causes normal features to "dilute" the errors of anomalous features, reducing detection sensitivity to partial anomalies. MAX aggregation ensures that the error of the most anomalous feature within a group directly becomes the score, reliably capturing partial anomalies.

4.3.4 Alpha Computation (Step ④)

The Anomaly Score is transformed to Alpha values $\alpha_v^{(g,t)} \in [0, 1]$. These Alpha values serve as control signals for the Soft-Gating mechanism.

4.3.4.1 IQR Standardization

Since the score $s_v^{(g,t)}$ has different baselines across nodes and feature groups, a uniform threshold cannot be directly applied. Standardization unifies the normal-period distributions for each node and feature group, enabling anomaly determination with a common threshold τ.

Using statistics computed from normal-period data, the standardized score for each node and feature group is calculated. This study adopts **IQR (Interquartile Range)-based standardization**, which is robust to outliers:

$$z_v^{(g,t)} = \frac{s_v^{(g,t)} - \tilde{\mu}^{(g)}}{\mathrm{IQR}^{(g)}} \tag{4.7}$$

Here, $\tilde{\mu}^{(g)}$ is the median of the normal period, and $\mathrm{IQR}^{(g)} = Q_3 - Q_1$ is the interquartile range.

Advantages of IQR Standardization: Compared to Z-score standardization using mean and standard deviation, IQR-based methods are less susceptible to outliers present in the normal period. This enables more stable anomaly estimation.

To prevent division by zero when IQR values are extremely small (i.e., when reconstruction errors are nearly constant), a minimum IQR value is enforced.

4.3.4.2 Alpha Computation via Sigmoid Function

The standardized score is passed through the Sigmoid function to compute Alpha values $\alpha_v^{(g,t)} \in [0, 1]$:

$$\alpha_v^{(g,t)} = \frac{1}{1 + \exp(k \cdot (z_v^{(g,t)} - \tau))} \tag{4.8}$$

Two hyperparameters play critical roles:

- τ (threshold): Determines the boundary between normal and anomalous. When $z_v^{(g,t)} = \tau$, $\alpha = 0.5$.
- k (slope): Controls the steepness of the Sigmoid function. Larger k values produce sharper transitions near the threshold.

Specific parameter values are described in Chapter 6.

Behavior of Alpha Values:

- $z_v^{(g,t)} \ll \tau$ (normal): $\alpha_v^{(g,t)} \approx 1.0 \rightarrow$ Memory is updated normally
- $z_v^{(g,t)} \approx \tau$ (boundary): $\alpha_v^{(g,t)} \approx 0.5 \rightarrow$ Candidate memory and old memory are blended equally
- $z_v^{(g,t)} \gg \tau$ (anomalous): $\alpha_v^{(g,t)} \approx 0.0 \rightarrow$ Memory update is suppressed, preserving old memory

Algorithm 4.2 presents the pseudocode for Group-wise Alpha value computation.

Algorithm 4.2 Group-wise Soft-Gating (Alpha Computation)

Input: Reconstruction error $\mathbf{err}^{(t)}$, normal-period statistics $\tilde{\mu}^{(g)}, \mathrm{IQR}^{(g)}$
Output: Group-wise Alpha values $\boldsymbol{\alpha}^{(t)}$

for each group $g \in \{\mathrm{Lat}, \mathrm{Mem}, \mathrm{CPU}, \mathrm{Net}\}$ **do**
  // Group score via MAX aggregation
  $s^{(g,t)} \leftarrow \max_{i \in \mathcal{I}_g} \mathbf{err}_i^{(t)}$

  // IQR standardization
  $z^{(g,t)} \leftarrow (s^{(g,t)} - \tilde{\mu}^{(g)})/\mathrm{IQR}^{(g)}$

  // Transform to Alpha via Sigmoid function
  $\alpha^{(g,t)} \leftarrow 1/(1 + \exp(k \cdot (z^{(g,t)} - \tau)))$
end for
return $\boldsymbol{\alpha}^{(t)} = [\alpha^{(\mathrm{Lat},t)}, \alpha^{(\mathrm{Mem},t)}, \alpha^{(\mathrm{CPU},t)}, \alpha^{(\mathrm{Net},t)}]$

4.4 Memory Update Path

This section details each component (Steps ⑤–⑧) of the Memory Update Path, indicated by the blue dashed line in Figure 4.1. This path updates

memory while protecting it through Dual-Stage Soft-Gating based on Alpha values.

The Soft-Gating mechanism dynamically controls the memory update rate based on reconstruction errors, preventing anomalous data from being stored in TGN memory. The design of this mechanism is inspired by the gating mechanism of Gated Recurrent Units (GRU) [5], which regulate information flow through learned gates. Similar to how GRU's update gate controls the blend between old hidden state and new candidate, Soft-Gating controls the blend between old memory and updated memory based on reconstruction errors.

ADA-TGN employs **Dual-Stage Soft-Gating** to protect memory at two stages:

- **Stage 1 (Step ⑤ GRU Input Stage)**: During anomaly detection, switch GRU input from observed values to reconstructed values.
- **Stage 2 (Step ⑧ Memory Update Stage)**: Blend the GRU output (candidate memory) with old memory using Alpha values to produce protected memory.

Necessity of Dual-Stage: With Stage 1 alone, unless Alpha values are exactly zero, small amounts of anomalous data enter the GRU, potentially causing gradual memory contamination. Adding Stage 2 provides a second layer of defense, preventing contamination generated within the GRU from propagating to memory.

Why External Gating is Essential: GRU's internal update gate cannot fully block incoming information. Due to the sigmoid function's output range $(0, 1)$, the update gate z_t can never reach exactly zero. Even when the GRU attempts to fully preserve old memory, the update gate remains slightly positive ($z_t > 0$), meaning a portion of the new candidate state always enters the hidden state.

Furthermore, unlike explicit memory systems such as FIFO queues used in MemStream [3], where contaminated entries can be evicted over time or diluted through K-nearest neighbor discounting, GRU's compressed hidden state does not support selective removal of specific contamination. While contamination may gradually decay through subsequent updates, there is no mechanism to target and remove particular anomalous information once it enters the hidden state.

This fundamental difference between explicit and implicit memory motivates our external Soft-Gating design:

- **Explicit Memory (FIFO)**: Contamination persists until evicted, but self-recovery is possible through eviction and discounting mechanisms.
- **Implicit Memory (GRU)**: Contamination may decay but cannot be selectively removed; prevention at input is essential.

By filtering anomalous data *before* it reaches the GRU (Stage 1) and blending outputs *after* GRU processing (Stage 2), ADA-TGN provides defense in depth against memory contamination.

4.4.1 GRU Input Soft-Gating (Step ⑤, Stage 1)

Stage 1 controls the input to the GRU based on Alpha values. During anomaly detection ($\alpha \approx 0$), reconstructed values are used, preventing anomalous data from entering the GRU:

$$\tilde{\mathbf{x}}_v^{(g,t)} = \alpha_v^{(g,t)} \cdot \mathbf{x}_v^{(g,t)} + (1 - \alpha_v^{(g,t)}) \cdot \hat{\mathbf{x}}_v^{(g,t)} \tag{4.9}$$

Effect of Stage 1:

- $\alpha \approx 1$ (normal): Use observed values $\mathbf{x}_v^{(g,t)}$ directly to learn new information
- $\alpha \approx 0$ (anomalous): Use reconstructed values $\hat{\mathbf{x}}_v^{(g,t)}$ to block anomalous data
- $0 < \alpha < 1$ (boundary): Blend observed and reconstructed values for gradual transition

Since the reconstructed values $\hat{\mathbf{x}}_v^{(g,t)}$ are generated from the old memory $\mathbf{m}_v^{(t-1)}$, they approximate expected values during normal periods. This ensures that the GRU receives near-normal inputs even during anomaly periods, preventing abrupt changes in internal state.

4.4.2 Group-wise Input Projection (Step ⑥)

The gated input $\tilde{\mathbf{x}}_v^{(g,t)}$ is transformed to the corresponding memory region dimension through group-specific projection layers:

$$\mathbf{p}_v^{(g,t)} = \text{GELU}(\text{RMSNorm}(\mathbf{W}_g \tilde{\mathbf{x}}_v^{(g,t)} + \mathbf{b}_g)) \tag{4.10}$$

The input features $\mathbf{x}_v^{(t)} \in \mathbb{R}^{12}$ are decomposed into four feature groups (Latency: 4 dimensions, Memory: 3 dimensions, CPU: 2 dimensions, Network: 3 dimensions). The specific feature composition of each group is detailed in Chapter 5.

Rationale for RMSNorm: Compared to Layer Normalization, RMSNorm [28] is more efficient as it does not compute mean shift. Additionally, the GELU activation function [12] provides smoother nonlinearity than ReLU and mitigates the vanishing gradient problem.

4.4.3 GRU Memory Update (Step ⑦)

Using the projection vectors $\mathbf{p}_v^{(t)}$ and old memory $\mathbf{m}_v^{(t-1)}$ as inputs, a single GRU (256 dimensions) generates the candidate memory:

$$\mathbf{p}_v^{(t)} = \text{concat}(\mathbf{p}_v^{(1,t)}, \ldots, \mathbf{p}_v^{(G,t)}) \tag{4.11}$$

$$\mathbf{m}_v^{\text{upd},(t)} = \text{GRU}(\mathbf{p}_v^{(t)}, \mathbf{m}_v^{(t-1)}) \tag{4.12}$$

The internal structure of the GRU follows Equations (3.1)–(3.4) in Chapter 3.

Design Rationale for Shared GRU: Following the original TGN paper [21], a single GRU is shared across all nodes. This reduces parameter count and improves computational efficiency. Since the characteristics of each feature group are preserved in the projection vectors through Group-wise Input Projection, a single GRU can appropriately process information from each feature group.

4.4.4 Memory Soft-Gating (Step ⑧, Stage 2)

Stage 2 performs protected memory updates by blending the candidate memory with old memory based on Alpha values:

$$\mathbf{m}_v^{(g,t)} = \alpha_v^{(g,t)} \cdot \mathbf{m}_v^{\text{upd},(g,t)} + (1 - \alpha_v^{(g,t)}) \cdot \mathbf{m}_v^{(g,t-1)} \tag{4.13}$$

Effect of Stage 2:

- $\alpha \approx 1$ (normal): Adopt the candidate memory $\mathbf{m}_v^{\text{upd},(t)}$ to store new information
- $\alpha \approx 0$ (anomalous): Preserve old memory $\mathbf{m}_v^{(t-1)}$ to prevent memory contamination
- $0 < \alpha < 1$ (boundary): Blend candidate and old memory to allow partial updates

Effect of Group-wise Independent Updates: Since Alpha values are computed independently for each feature group, even when one feature group is anomalous, memory for normal feature groups is updated normally. This enables continued memory updates for normal feature groups even during partial anomaly periods.

4.5 Training Strategy: Raw Message Store

ADA-TGN employs the **Raw Message Store** training strategy proposed in the original TGN paper [21] (Section 3.2). This strategy enables gradient flow

through the GRU by updating memory *before* prediction using the previous step's input.

Note on Training vs. Inference: During training, ADA-TGN uses only the Raw Message Store strategy without applying the Soft-Gating mechanism. This allows the GRU to learn normal time series patterns from training data through standard gradient descent. The Soft-Gating mechanism described in Section 4.4 is applied only during inference to dynamically protect memory from anomalous inputs. This two-phase design first optimizes the model on normal data during training, then activates the Soft-Gating mechanism during inference to handle anomalous inputs.

4.5.1 Problem: GRU Gradient Disconnection

In a naive implementation where memory update occurs *after* prediction, the following processing flow is used:

1. Compute embeddings from old memory: $\mathbf{h}^{(t)} = \text{GNN}(\mathbf{m}^{(t-1)})$
2. Decode to prediction: $\hat{\mathbf{x}}^{(t)} = \text{Decoder}(\mathbf{h}^{(t)})$
3. Compute loss: $\mathcal{L} = \|\mathbf{x}^{(t)} - \hat{\mathbf{x}}^{(t)}\|^2$
4. Update memory: $\mathbf{m}^{(t)} = \text{GRU}(\mathbf{x}^{(t)}, \mathbf{m}^{(t-1)})$

In this flow, the memory $\mathbf{m}^{(t)}$ must be *detached* from the computation graph to prevent gradient accumulation across time steps, which would cause memory overflow during training. However, this detachment disconnects the GRU from the loss function, preventing the GRU parameters from receiving gradients during backpropagation. As a result, the GRU is effectively not trained, and its memory update capability is not optimized.

4.5.2 Solution: Raw Message Store Strategy

The Raw Message Store strategy solves this problem by storing the previous step's input and using it to update memory *before* prediction:

1. Retrieve stored input from previous step: $\mathbf{x}_{\text{stored}}^{(t-1)}$
2. Update memory using stored input: $\mathbf{m}^{(t)} = \text{GRU}(\text{Proj}(\mathbf{x}_{\text{stored}}^{(t-1)}), \mathbf{m}^{(t-1)})$
3. Compute embeddings from **updated** memory: $\mathbf{h}^{(t)} = \text{GNN}(\mathbf{m}^{(t)})$
4. Decode to prediction: $\hat{\mathbf{x}}^{(t)} = \text{Decoder}(\mathbf{h}^{(t)})$
5. Compute loss: $\mathcal{L} = \|\mathbf{x}^{(t)} - \hat{\mathbf{x}}^{(t)}\|^2$
6. Store current input for next step: $\mathbf{x}_{\text{stored}}^{(t)} = \mathbf{x}^{(t)}$ (detached), which will be used in Step 2 of time step $t + 1$

7. Store updated memory for next step: $\mathbf{m}_{\text{stored}}^{(t)} = \mathbf{m}^{(t)}$ (detached), which will be used as $\mathbf{m}^{(t)}$ in Step 2 of time step $t+1$

This creates a gradient path through the GRU:

$$\mathbf{x}_{\text{stored}}^{(t-1)} \xrightarrow{\text{Proj}} \text{GRU} \rightarrow \mathbf{m}^{(t)} \xrightarrow{\text{GNN}} \mathbf{h}^{(t)} \xrightarrow{\text{Decoder}} \hat{\mathbf{x}}^{(t)} \rightarrow \mathcal{L} \tag{4.14}$$

The GRU is now part of the computation graph, and its parameters receive gradients during backpropagation. This enables end-to-end learning of the memory update mechanism, allowing the GRU to learn optimal memory update strategies for the reconstruction task.

Key Insight: Using the previous step's input $\mathbf{x}^{(t-1)}$ (rather than current input $\mathbf{x}^{(t)}$) for memory update before predicting $\hat{\mathbf{x}}^{(t)}$ avoids information leakage while maintaining the gradient connection. The current input $\mathbf{x}^{(t)}$ is only stored (detached) for use in the next time step.

Chapter 5

Feature Extraction and Preprocessing

This chapter describes the feature extraction methods and graph structure definition for ADA-TGN inputs. Section 5.1 presents an overview of the RE2-OB dataset. Section 5.2 explains the design of node features. Section 5.3 describes the graph structure construction. Section 5.4 details the time series feature extraction methods. Section 5.5 explains preprocessing and standardization. Section 5.6 describes the dataset conversion to PyTorch Geometric format.

5.1 RE2-OB Dataset

This study uses the RE2-OB (RCAEval: Online Boutique) [19] dataset, a benchmark for Root Cause Analysis (RCA) in microservice systems. RE2-OB is a dataset that records metrics and traces after injecting anomalies into the "Online Boutique" microservice demo application published by Google Cloud.

5.1.1 Dataset Overview

The RE2-OB dataset has the following characteristics:

- **Experimental configuration**: 5 services × 4 anomaly types × 3 replications = 60 experiments
- **Anomaly injection target services**: checkoutservice, currencyservice, emailservice, productcatalogservice, recommendationservice
- **Anomaly types**: cpu (elevated CPU utilization), mem (elevated memory usage), delay (latency delay), loss (packet loss)
- **Data sources**:

- metrics.csv: Metric time series data collected by Prometheus, cAdvisor, and Istio (1-second intervals)
- inject_time.txt: Anomaly injection timestamp (Unix timestamp)

- **Experiment duration**: Approximately 24 minutes per experiment (approximately 12 minutes normal period + approximately 12 minutes anomaly period)

5.1.2 Monitored Services

While the RE2-OB dataset contains 11 microservices, this study targets only the **10 services with Istio Sidecar injection**. The redis service was excluded because the Istio Sidecar is not injected, making Istio-related metrics (istio_*) unavailable.

Table 5.1 lists the 10 monitored services and their roles.

Table 5.1: Online Boutique monitored services (10 services)

Service Name	**Notes**
frontendservice	
checkoutservice	
productcatalogservice	
currencyservice	
recommendationservice	
cartservice	
paymentservice	
shippingservice	
emailservice	
adservice	
redis	*Excluded due to lack of Istio Sidecar injection*

5.2 Node Feature Definition

This section describes the 12-dimensional features extracted for each service (node). Features are classified into four groups corresponding to the Group-wise Soft-Gating mechanism described in Chapter 4.

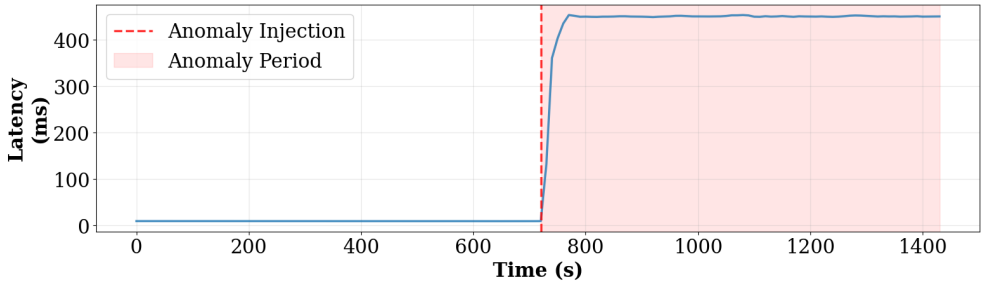
5.2.1 12-Dimensional Feature Composition

Table 5.2 shows the 12-dimensional node features used in this study.

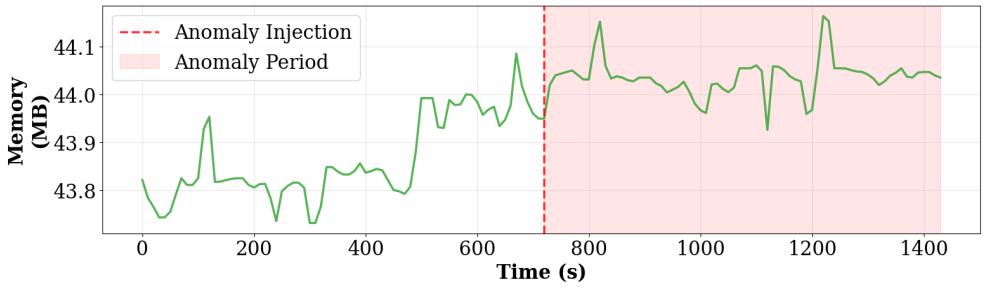
Table 5.2: Composition of 12-dimensional node features

Group	Dim	Feature Name	Description
Latency	0	istio_latency_50	50th percentile latency
	1	istio_latency_90	90th percentile latency
	2	istio_latency_95	95th percentile latency
	3	istio_latency_99	99th percentile latency
Memory	4	container_memory_usage_bytes	Memory usage (bytes)
	5	container_memory_rss	Resident Set Size
	6	container_memory_working_set_bytes	Working set (bytes)
CPU	7	container_cpu_user_seconds_total	User mode CPU time
	8	container_cpu_system_seconds_total	System mode CPU time
Network	9	istio_request_total	Total request count
	10	container_network_receive_bytes_total	Received bytes
	11	container_network_transmit_bytes_total	Transmitted bytes
Total: 12 dimensions			

Figures 5.1 and 5.2 show the time series variation of representative features from the four groups during a delay anomaly injection experiment on recommendationservice. A notable increase in Latency Group values is observed at the anomaly injection time (red dashed line). This confirms that different groups are affected depending on the anomaly type.

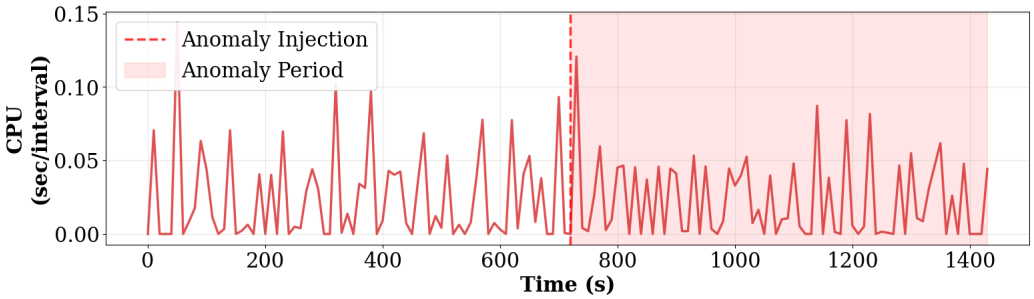


(a) Latency Group (istio_latency_90)

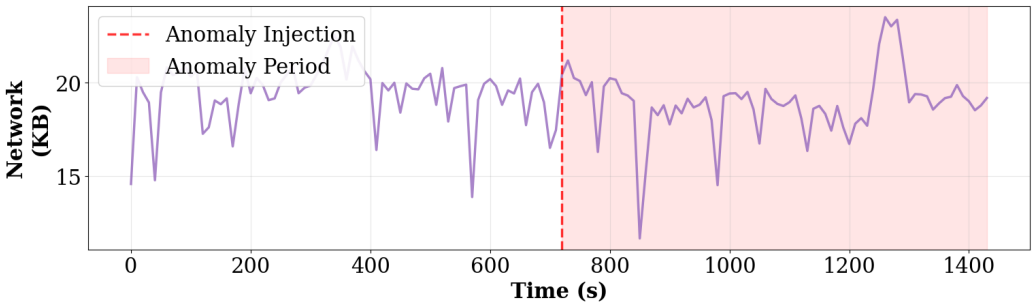


(b) Memory Group (container_memory_working_set_bytes)

Figure 5.1: Time series variation of representative features from four groups (1/2): Latency and Memory (recommendationservice_delay experiment). Red background: anomaly period.



(a) CPU Group (container_cpu_user_seconds_total)



(b) Network Group (container_network_receive_bytes_total)

Figure 5.2: Time series variation of representative features from four groups (2/2): CPU and Network (recommendationservice_delay experiment). Red background: anomaly period.

5.2.2 Design Rationale for Group-wise Structure

The design of dividing 12-dimensional features into four groups is based on SRE's **Four Golden Signals** (Latency, Traffic, Errors, Saturation) [2]. This study instantiates these as four groups: Latency, Memory, CPU, and Network (Table 5.3).

Table 5.3: Correspondence between Golden Signals and feature groups

Golden Signal	SRE Definition	Corresponding Group
Latency	Request processing time	Latency Group
Traffic	System demand	Network Group
Saturation	Resource utilization	CPU + Memory Group
Errors	Request failure rate	(Indirect)

This group partitioning enables the Soft-Gating mechanism described in Chapter 4 to compute independent Alpha values for each group, addressing partial anomalies. Since each anomaly type primarily affects specific groups (cpu anomaly → CPU Group, delay anomaly → Latency Group, etc.), appropriate memory protection is achieved for each anomaly type.

5.3 Graph Structure Construction

This section describes how to represent inter-microservice dependencies as a graph structure. ADA-TGN constructs a directed graph $G = (V, E)$ where services are nodes and inter-service call relationships are edges.

5.3.1 Service Dependency Extraction

In microservice systems, inter-service call relationships provide crucial information. In the RE2-OB dataset, distributed trace data is recorded in traces.csv, from which service dependencies can be dynamically extracted.

However, since the Online Boutique architecture is known in this study, service dependencies are defined statically.

Figure 5.3 shows the service dependencies of Online Boutique. frontend-service serves as the entry point, receiving user requests. checkoutservice is a core service that calls multiple backend services (cart, currency, email, payment, productcatalog, shipping).

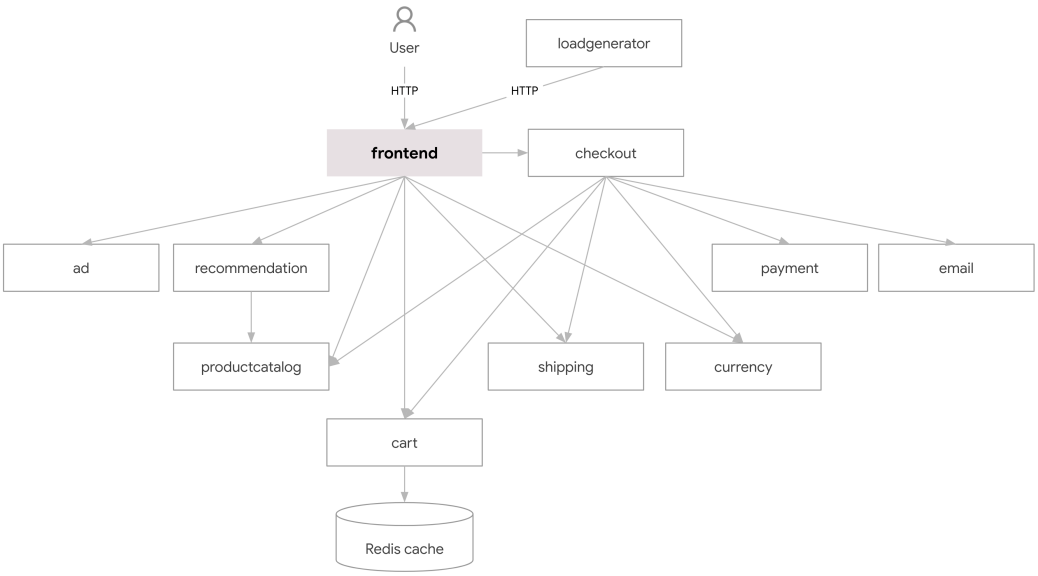


Figure 5.3: Service dependencies of Online Boutique (cited from [11]).

5.3.2 Edge Definition

Based on the Online Boutique architecture, 14 edges are defined. Table 5.4 lists all edges.

Table 5.4: Service dependencies (14 edges)

No.	Caller	→	Callee
Calls from frontendservice (7 edges)			
1	frontendservice	→	adservice
2	frontendservice	→	cartservice
3	frontendservice	→	checkoutservice
4	frontendservice	→	currencyservice
5	frontendservice	→	productcatalogservice
6	frontendservice	→	recommendationservice
7	frontendservice	→	shippingservice
Calls from checkoutservice (6 edges)			
8	checkoutservice	→	cartservice
9	checkoutservice	→	currencyservice
10	checkoutservice	→	emailservice
11	checkoutservice	→	paymentservice
12	checkoutservice	→	productcatalogservice
13	checkoutservice	→	shippingservice
Calls from recommendationservice (1 edge)			
14	recommendationservice	→	productcatalogservice
Total: 14 edges			

Edge Direction:
Edge direction follows "Caller → Callee." For example, when frontendservice calls checkoutservice, the edge is frontendservice → checkoutservice.

5.3.3 Graph Representation

Graph $G = (V, E)$ consists of the node set V (10 services, indexed alphabetically) and edge set E (14 directed edges). Features of all services at time t are represented as the node feature matrix $\mathbf{X}^{(t)} \in \mathbb{R}^{10\times 12}$.

5.4 Time Series Feature Extraction

This section details the methods for extracting time series features from metrics.csv. Appropriate Snapshot Intervals must be set considering the characteristics of Prometheus metric collection.

5.4.1 Snapshot Interval Configuration

In time series feature extraction, continuous time is divided into discrete snapshots (windows), and metric values within each window are aggregated

to generate a single feature vector. This study adopts a **10-second granularity**.

5.4.2 Extraction Methods by Feature Group

For each feature group, extraction methods are applied according to metric type.

Latency/Memory Group These are Prometheus gauge types, and the within-window average is computed:

$$x^{(t)} = \frac{1}{|\mathcal{T}_t|} \sum_{\tau \in \mathcal{T}_t} \text{metrics}(\tau) \tag{5.1}$$

CPU Group These are Prometheus counter types (cumulative values), so the following processing is applied:

1. Difference calculation (rate)
2. Outlier clipping $[0, 10^6]$
3. Gaussian Smoothing ($\sigma = 2.0$)
4. Within-window average computation

Gaussian Smoothing removes sampling noise while preserving abrupt changes at anomaly onset.

Network Group These are counter types, but the within-window average of cumulative values is used directly as features.

5.4.3 Normal/Anomaly Period Separation

In the RE2-OB dataset, anomaly injection timestamps are recorded in inject_time.txt. Using this timestamp as the boundary, the normal period ($t <$ inject_time) and anomaly period ($t \geq$ inject_time) are separated.

5.5 Preprocessing and Standardization

The time series features extracted in Section 5.4 are normalized through scaling. This study adopts different scaling methods depending on the feature group.

5.5.1 Scaling Method Selection

This study adopts the following two scaling methods according to feature characteristics:

1. **CPU features**: Fixed-value division ($z = x/0.5$)
2. **Other features**: P99 scaling + Asinh transformation

Conventional standardization methods (Z-Score, IQR, etc.) shift the baseline by subtracting the mean or median, causing anomaly period data to become extreme values. The scaling methods in this study preserve time series patterns while ensuring robustness to outliers.

5.5.2 P99 Scaling

P99 scaling is applied to features in the Latency, Memory, and Network Groups. For each combination of service v and feature f, the 99th percentile value $P99_v^f$ is computed from normal period data across all experiments, and scaling is performed by division only: $z_v^{f,(t)} = x_v^{f,(t)} / P99_v^f$.

The advantages of P99 scaling are:

1. Time series patterns are preserved through division only
2. Normal periods fall within $[0, 1.0]$ making threshold setting intuitive
3. Robustness to outliers through the use of P99 [7]

5.5.3 Asinh Transformation

After P99 scaling, the Asinh (inverse hyperbolic sine) transformation [17] is applied: $z_{\text{asinh}} = \text{arcsinh}(z/s) \cdot c$ ($s = 0.5$, $c = 2.0$).

The Asinh transformation is nearly linear for small values ($|x| < 0.5$) and compresses logarithmically for large values. This moderately compresses extreme anomaly values while preserving relative ordering, and is differentiable throughout, making it suitable for gradient-based learning.

Figure 5.4 shows the effect of each scaling method. P99 scaling + Asinh transformation prevents excessive amplification of anomaly values compared to IQR standardization.

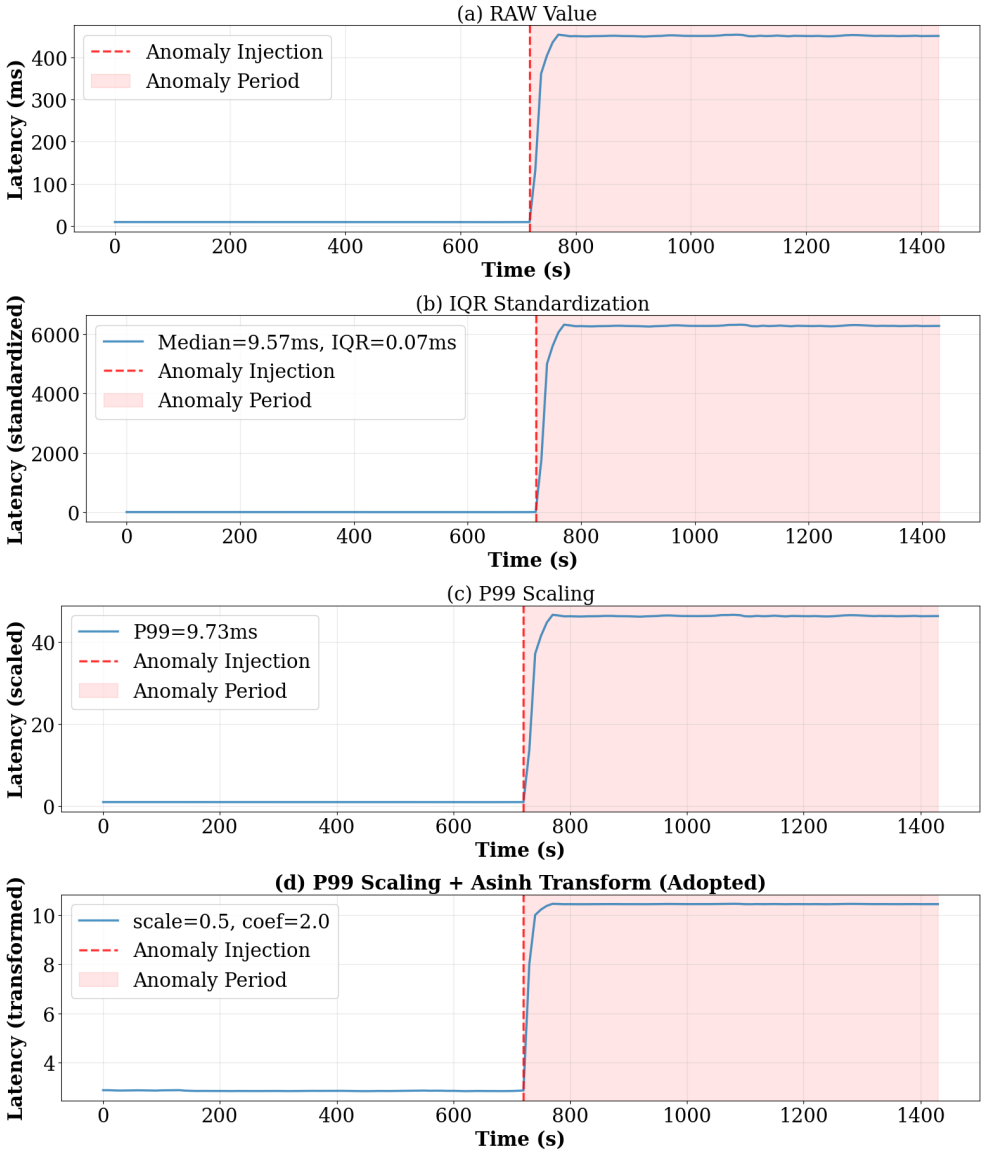


Figure 5.4: Comparison of scaling methods (Latency features).

5.5.4 Fixed-Value Scaling for CPU Features

CPU features require scaling after the Gaussian Smoothing ($\sigma = 2.0$) described in Section 5.4.2. Figure 5.5 shows the results of applying two scaling methods to CPU features after Gaussian Smoothing (a): (b) P99 scaling + Asinh transformation as with other features, and (c) fixed-value division ($z = x/0.5$).

This study adopts **(c) fixed-value division** for CPU features. The rationale is as follows:

Since the P99 value of CPU features is approximately 0.1, applying (b) P99 scaling excessively amplifies minor variations during normal periods,

raising the reconstruction error baseline. In contrast, (c) fixed-value division preserves the smooth waveform patterns that were smoothed by Gaussian Smoothing, enabling the model to learn stable normal patterns.

The scale inconsistency between CPU features and other features is absorbed by IQR standardization during anomaly score computation (Chapter 6), so this does not pose a problem.

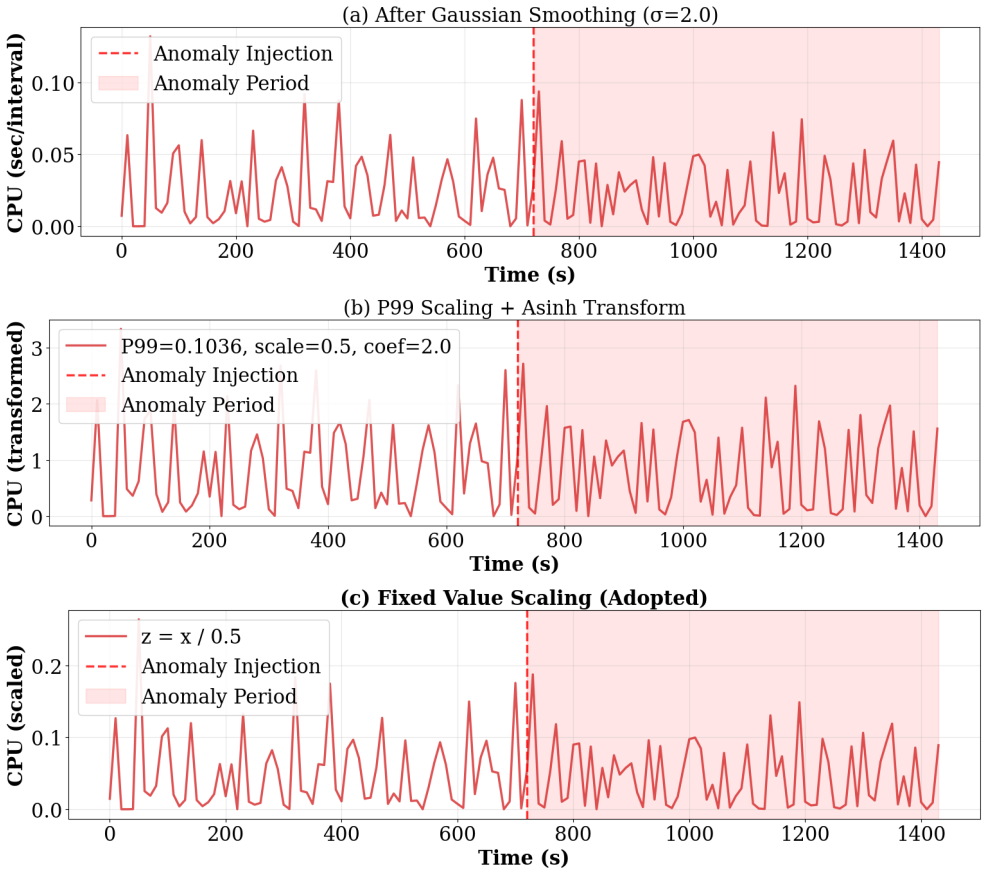


Figure 5.5: Comparison of CPU feature scaling.

5.5.5 Final Clipping

After all scaling processes, final values are clipped to the range $[-10, +10]$ to prevent extreme outliers from adversely affecting subsequent learning:

$$\tilde{z}^{f,(t)} = \text{clip}(z^{f,(t)}, -10, +10) \tag{5.2}$$

5.6 Dataset Construction

Preprocessed time series features and graph structures are converted to PyTorch Geometric (PyG) [10] format datasets. Data at each time step

t is converted to a PyG Data object. The main attributes are as follows:

- **x**: Node feature matrix $\in \mathbb{R}^{10\times 12}$. Used as input to the Graph Attention Encoder.
- **edge_index**: Inter-service call relationships $\in \mathbb{Z}^{2\times 14}$. A matrix representation of the 14 edges in Table 5.4, used as input to the Graph Attention Encoder.

$$\text{edge\_index} = \begin{bmatrix} \text{src}_0 & \text{src}_1 & \cdots & \text{src}_{13} \\ \text{dst}_0 & \text{dst}_1 & \cdots & \text{dst}_{13} \end{bmatrix} \tag{5.3}$$

  Here, src_i and dst_i are the node indices of the caller and callee for the i-th edge.

- **y**: Graph-level label (normal: 0, anomaly: 1). Used for calculating anomaly detection performance during evaluation.
- **node_y**: Node-level label (1 only for the root cause service). Used for calculating root cause identification performance during evaluation.

Chapter 6

Experiments

This chapter experimentally validates the effectiveness of the proposed method ADA-TGN. We first describe the experimental setup (Section 6.1), followed by the Soft-Gating mechanism ablation study (Section 6.2), anomaly detection performance evaluation (Section 6.3.2), and root cause analysis performance evaluation (Section 6.3.3).

6.1 Experimental Setup

6.1.1 Model Training

In this experiment, an independent model is trained for each experiment (service × anomaly type × replication). Since the RE2 dataset comprises 5 services × 4 anomaly types × 3 replications = 60 experiments, a total of 60 models are trained. This design stems from the data collection methodology of the RE2 dataset. In RCAEval [19], each experiment generates "random load between 10 and 200 requests per second." Due to this load randomness, time series patterns differ significantly across experiments. Attempting to train a single model on multiple experiments causes conflicting patterns under different load conditions, resulting in gradient conflict. Therefore, we adopted an approach where models are trained individually for each experiment.

Training data uses only the **normal period** of each experiment; anomaly period data is not included in training. This allows the model to learn only normal time series patterns, detecting anomalies through increased reconstruction errors during anomaly periods.

Training Strategy: ADA-TGN employs the **Raw Message Store** training strategy proposed in the original TGN paper [21]. In this strategy, memory is updated using the *previous* step's input before computing the current prediction, enabling gradient flow through the GRU. This differs from a naive implementation where memory update occurs after prediction, which disconnects the GRU from the loss function. Details are described in Section 4.5, and validation results are presented in Section 6.1.4.

Note that in this setup, the normal period used for training is also used for evaluation, resulting in smaller reconstruction errors during normal periods (data leakage). However, the evaluation objective of this study is **detection performance during anomaly periods**, and reconstruction accuracy during normal periods is not the primary evaluation target. This design is common in statistical anomaly detection methods, where referencing normal period data is inherently necessary to define the "normal" baseline.

Training settings are shown in Table 6.1.

6.1.2 Hyperparameters

Table 6.1: Hyperparameter settings

Category	Parameter	Value
Training	Epochs	500
	Optimizer	AdamW
	Learning rate	0.003
	Weight decay	0.01
	Gradient clipping	5.0
Soft-Gating	Threshold τ (Eq. 4.8)	1.5
	Sigmoid slope k (Eq. 4.8)	5.0
	Minimum IQR (Eq. 4.7)	0.05

The Soft-Gating threshold $\tau = 1.5$ corresponds to Tukey's inner fence in statistics. In outlier detection, $1.5 \times$ IQR is generally used as the threshold for mild outliers, and $3.0 \times$ IQR for extreme outliers [24]. This study adopts the more stringent $\tau = 1.5$ for a conservative design that suppresses memory impact even for mild anomalies. This considers the characteristic of reconstruction-based methods where detection capability is impaired once memory is contaminated. The sigmoid slope $k = 5.0$ provides a gradual α value transition near the threshold, preventing abrupt memory updates. The minimum IQR min_iqr $= 0.05$ prevents numerical instability when normal period variability is small, particularly important for well-trained models with low reconstruction error variance.

6.1.3 Evaluation Flow

All evaluation experiments in this study (ablation study, anomaly detection, root cause analysis) follow the unified two-pass evaluation protocol described below.

Pass 1: Normal Period Processing and IQR Statistics Computation

1. **Warmup Phase** (first 20 windows, 200 seconds): TGN memory is initialized with small random values drawn from a normal distribution:

$$\mathbf{m}_v^{(0)} \sim \mathcal{N}(0, 0.01^2\mathbf{I}), \quad \forall v \in V \tag{6.1}$$

   where $\mathbf{I}$ is the identity matrix. Reconstruction is unstable until sufficient data is incorporated. In this phase, memory is updated with $\alpha = 1.0$ (unconditional update) to stabilize memory state and ensure reconstruction accuracy.

2. **Statistics Collection Phase** (normal period after warmup): Inference is performed in normal evaluation mode while collecting reconstruction errors for each feature group. At the end of the normal period, IQR statistics (median, IQR) are computed from the collected errors.

Pass 2: Anomaly Score Computation

Using the IQR statistics determined in Pass 1, anomaly scores are computed for the entire period (normal and anomaly periods). For each node v and each feature group g, the reconstruction error $e_{v,g}^{(t)}$ is IQR-standardized using statistics computed from the normal period:

$$s_{v,g}^{(t)} = \frac{e_{v,g}^{(t)} - \tilde{\mu}_{v,g}}{\mathrm{IQR}_{v,g}} \tag{6.2}$$

Here, $\tilde{\mu}_{v,g}$ is the median of reconstruction errors during the normal period, and $\mathrm{IQR}_{v,g}$ is the interquartile range.

Rationale for IQR Standardization: This is for robustness to outliers and comparability across feature groups with different scales (see Section 4.3.4 for details).

At each window, Alpha values are computed from these IQR-standardized anomaly scores, and inference is performed while applying memory protection through the Soft-Gating mechanism.

This two-pass evaluation ensures that statistics are computed only from normal period data and the entire period is evaluated with consistent criteria.

6.1.4 Raw Message Store Training Validation

To verify that the Raw Message Store training strategy (Section 4.5) enables gradient flow through the GRU, we measured the gradient norm of GRU parameters after backpropagation.

We compare two training strategies:

- **Raw Message Store**: Updates memory *before* prediction using the previous step's stored input (Section 4.5.2).
- **Naive**: Updates memory *after* prediction, requiring detachment that disconnects the GRU from the computation graph (Section 4.5.1).

Measurement Method: After performing forward and backward passes on several training windows, we computed the L2 norm of gradients for all GRU parameters (`weight_ih`, `weight_hh`, `bias_ih`, `bias_hh`)[1].

Table 6.2 shows the results.

Table 6.2: GRU gradient norm by training strategy

Training Strategy	Gradient Norm
Raw Message Store	1.53 ± 0.17
Naive (post-prediction update)	0.00

Interpretation: Models trained with the Raw Message Store strategy exhibited gradient norms of approximately 1.5, indicating proper gradient flow (healthy range: 0.01–10). In contrast, the naive approach (post-prediction memory update with detachment) resulted in zero gradients, confirming that the GRU was disconnected from the computation graph and received no training signal. This validates that Raw Message Store training enables end-to-end learning of the memory update mechanism.

6.2 Memory Protection: Dual-Stage Ablation Study

This section experimentally verifies that the proposed Dual-Stage Soft-Gating mechanism effectively addresses the memory contamination problem formalized in Chapter 3. By systematically enabling and disabling each stage, we analyze the individual contribution of Stage 1 (GRU Input Soft-Gating) and Stage 2 (Memory Soft-Gating).

6.2.1 Objective and Comparison Targets

This ablation study analyzes the contribution of each gating stage by evaluating four configurations shown in Table 6.3. Vanilla-TGN serves as the baseline with both stages disabled.

Key insight: Only one model is trained, then evaluated with different stage configurations by toggling `enable_stage1` and `enable_stage2` flags during inference. This ensures fair comparison as all configurations use identical learned parameters.

Figure 6.1 shows the Vanilla-TGN architecture (baseline). Unlike ADA-TGN (Figure 4.1), observed values are directly input to the GRU without

[1]PyTorch GRU implementation concatenates parameters for the three gates (reset, update, candidate). `weight_ih` corresponds to input-to-hidden weights $[\mathbf{W}_{ir}; \mathbf{W}_{iz}; \mathbf{W}_{in}]$, and `weight_hh` to hidden-to-hidden weights $[\mathbf{W}_{hr}; \mathbf{W}_{hz}; \mathbf{W}_{hn}]$ in Eqs. (3.1)–(3.3). Biases are similarly concatenated.

Table 6.3: Ablation study configurations

Configuration	Stage 1	Stage 2
Vanilla-TGN (Baseline)	–	–
Stage 1 Only	✓	–
Stage 2 Only	–	✓
Dual-Stage (ADA-TGN)	✓	✓

GRU Input Soft-Gating, and memory is updated unconditionally without Memory Soft-Gating ($\mathbf{m}_v^{(t)} = \mathbf{m}_v^{\text{upd},(t)}$).

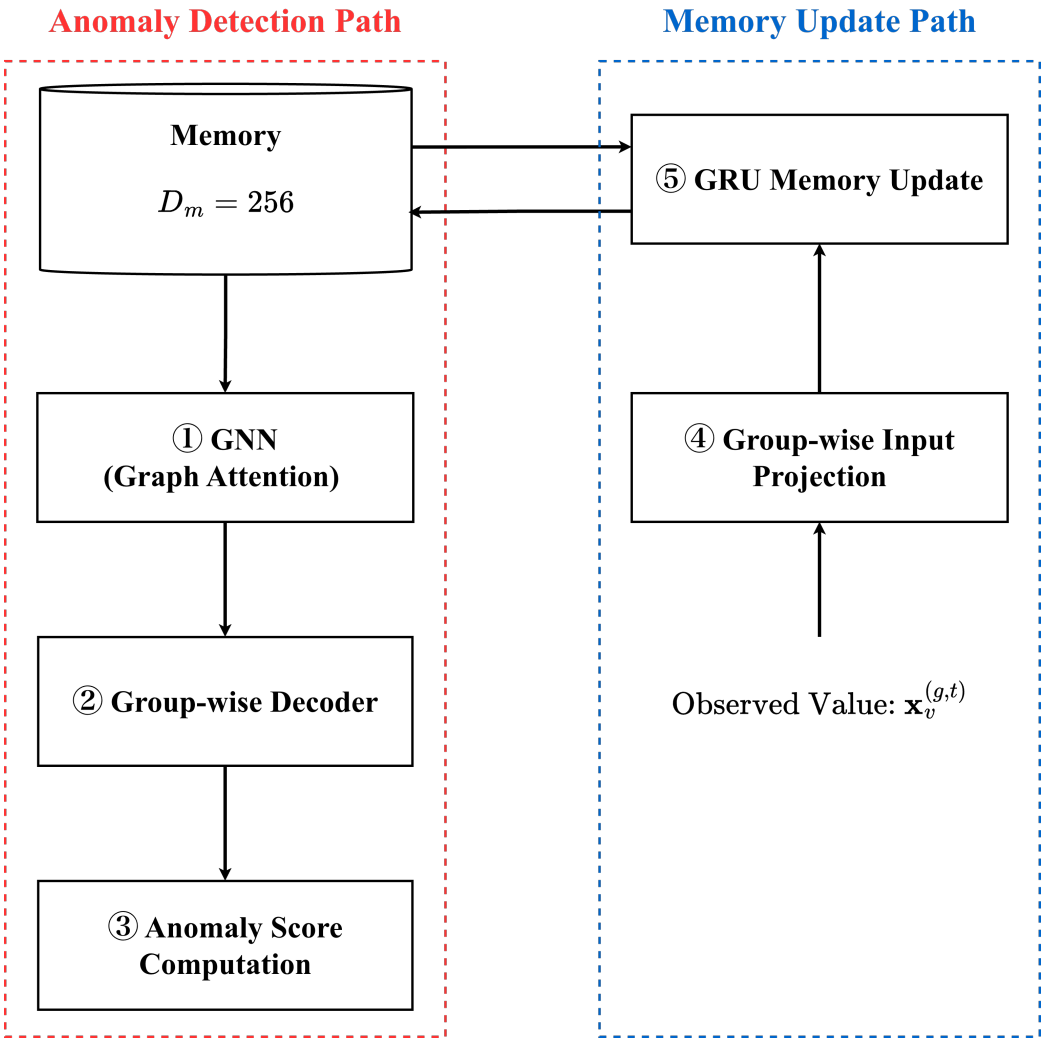


Figure 6.1: Vanilla-TGN architecture (baseline, both stages disabled). Observed values are directly input to the GRU, and memory is updated unconditionally.

6.2.2 Evaluation Metrics

To evaluate the impact of memory contamination from multiple perspectives, analysis is conducted from the following three viewpoints:

1. **Reconstruction Stability**: Observe whether model reconstruction values maintain normal patterns or become unstable during anomaly periods. When memory is contaminated, the Decoder performs reconstruction from contaminated memory state, causing unstable output.

2. **Alpha Value Changes**: Verify whether Alpha values change appropriately between normal and anomaly periods in ADA-TGN. During normal periods, $\alpha \approx 1.0$ (trust observed values) is expected; during anomaly periods, $\alpha \approx 0.0$ (trust predicted values) is expected.

3. **Memory Contamination Analysis**: Visualize memory contamination by projecting memory states to 2D using PCA (Principal Component Analysis). PCA is learned on normal period memory states only, and then applied to both normal and anomaly periods. When contamination occurs, anomaly period memory states deviate from the normal region.

6.2.3 Results: Reconstruction Stability

Figures 6.2–6.5 show reconstruction results for four representative features across all configurations in a representative experiment (currencyservice_cpu experiment, rep1). This section visualizes metrics from the **anomaly injection target service** (currencyservice in this experiment). One representative feature was selected from each group: istio_latency_50 (50th percentile latency) from the Latency group, container_memory_usage_bytes (memory usage) from the Memory group, container_cpu_user_seconds_total (user CPU time) from the CPU group, and container_network_transmit_bytes_total (transmitted bytes) from the Network group.

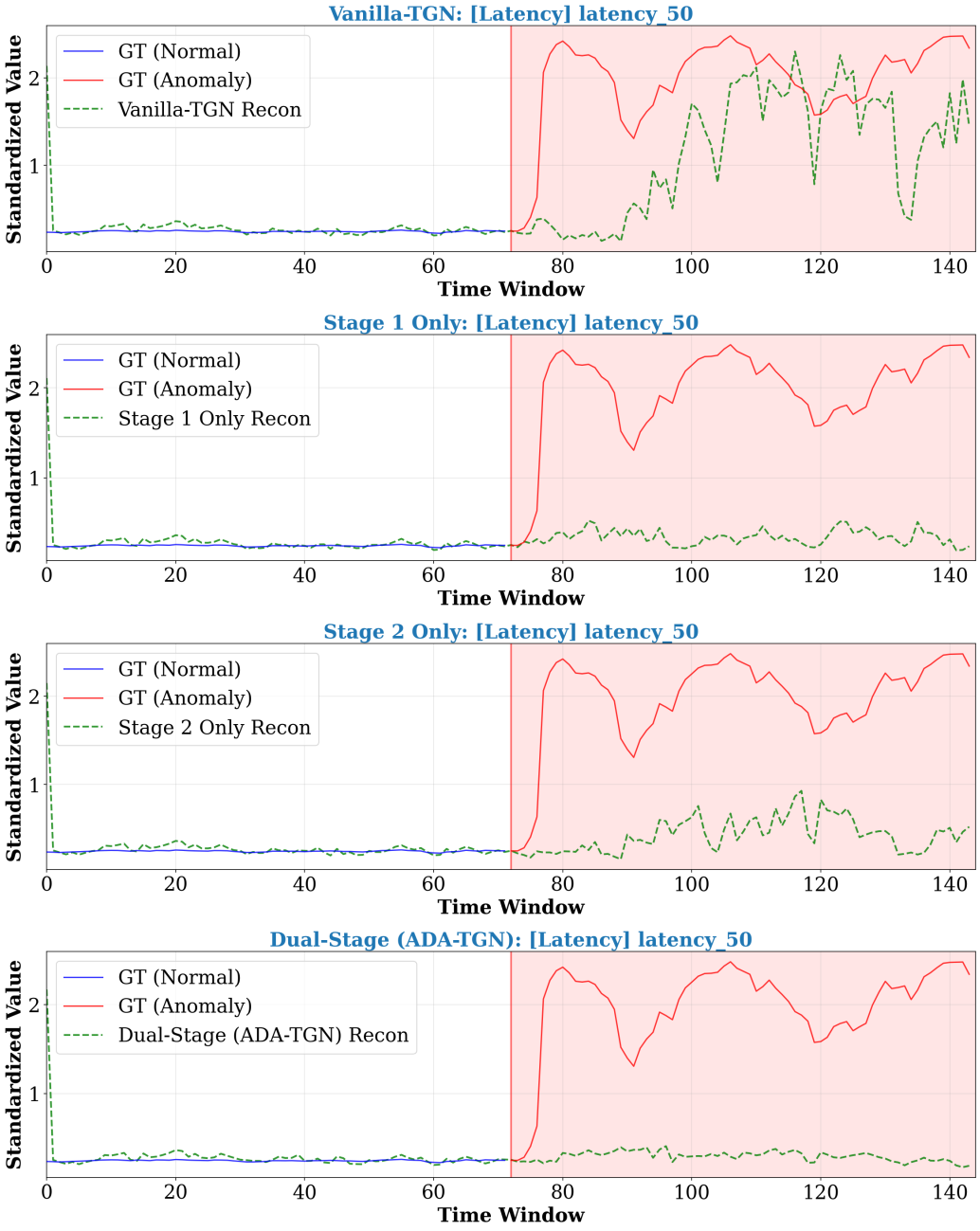


Figure 6.2: Reconstruction comparison for Latency (istio_latency_50). From top to bottom: Vanilla-TGN, Stage 1 Only, Stage 2 Only, Dual-Stage (ADA-TGN).

Figure 6.3: Reconstruction comparison for Memory (container_memory_usage_bytes). From top to bottom: Vanilla-TGN, Stage 1 Only, Stage 2 Only, Dual-Stage (ADA-TGN).

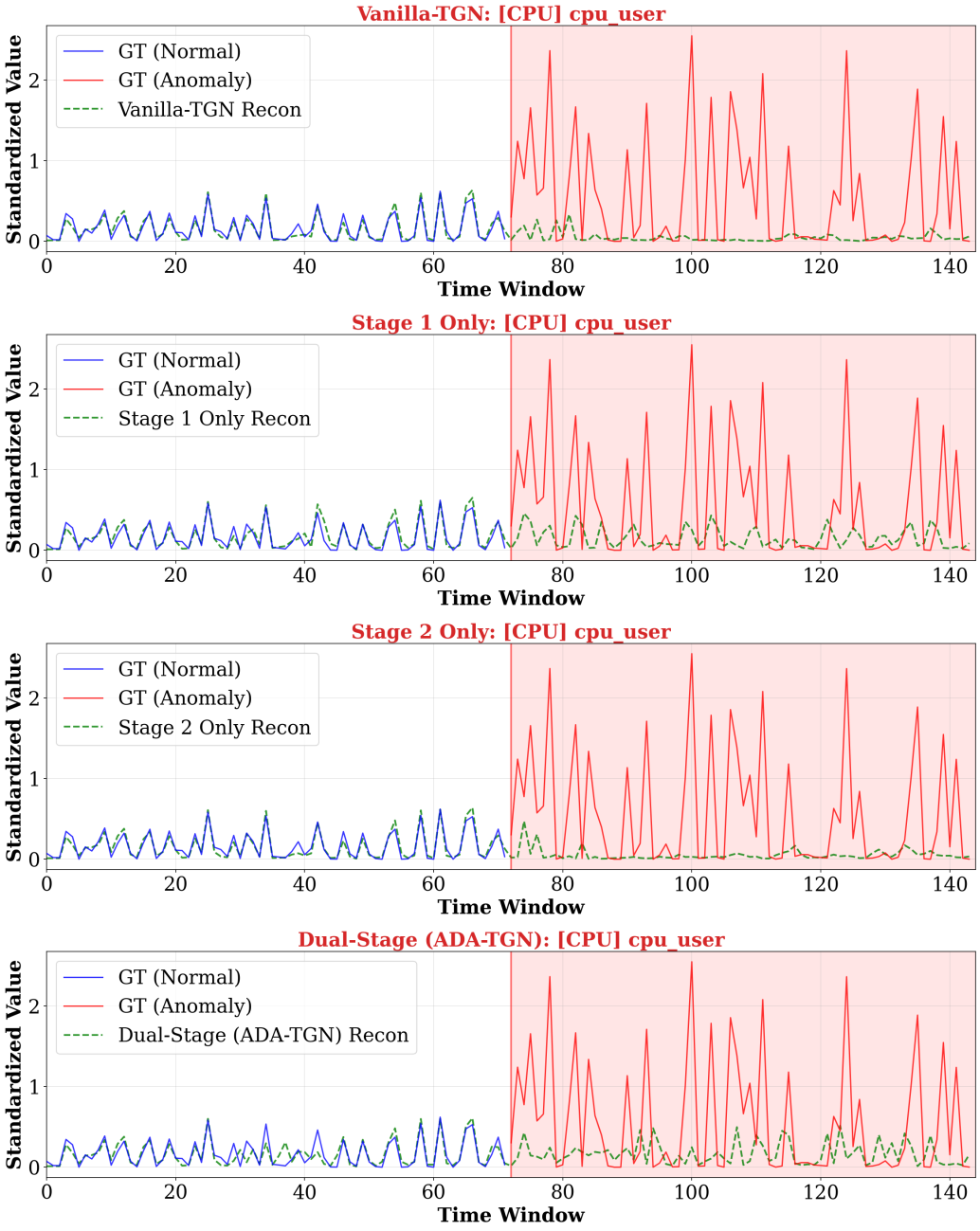


Figure 6.4: Reconstruction comparison for CPU (container_cpu_user_seconds_total). From top to bottom: Vanilla-TGN, Stage 1 Only, Stage 2 Only, Dual-Stage (ADA-TGN).

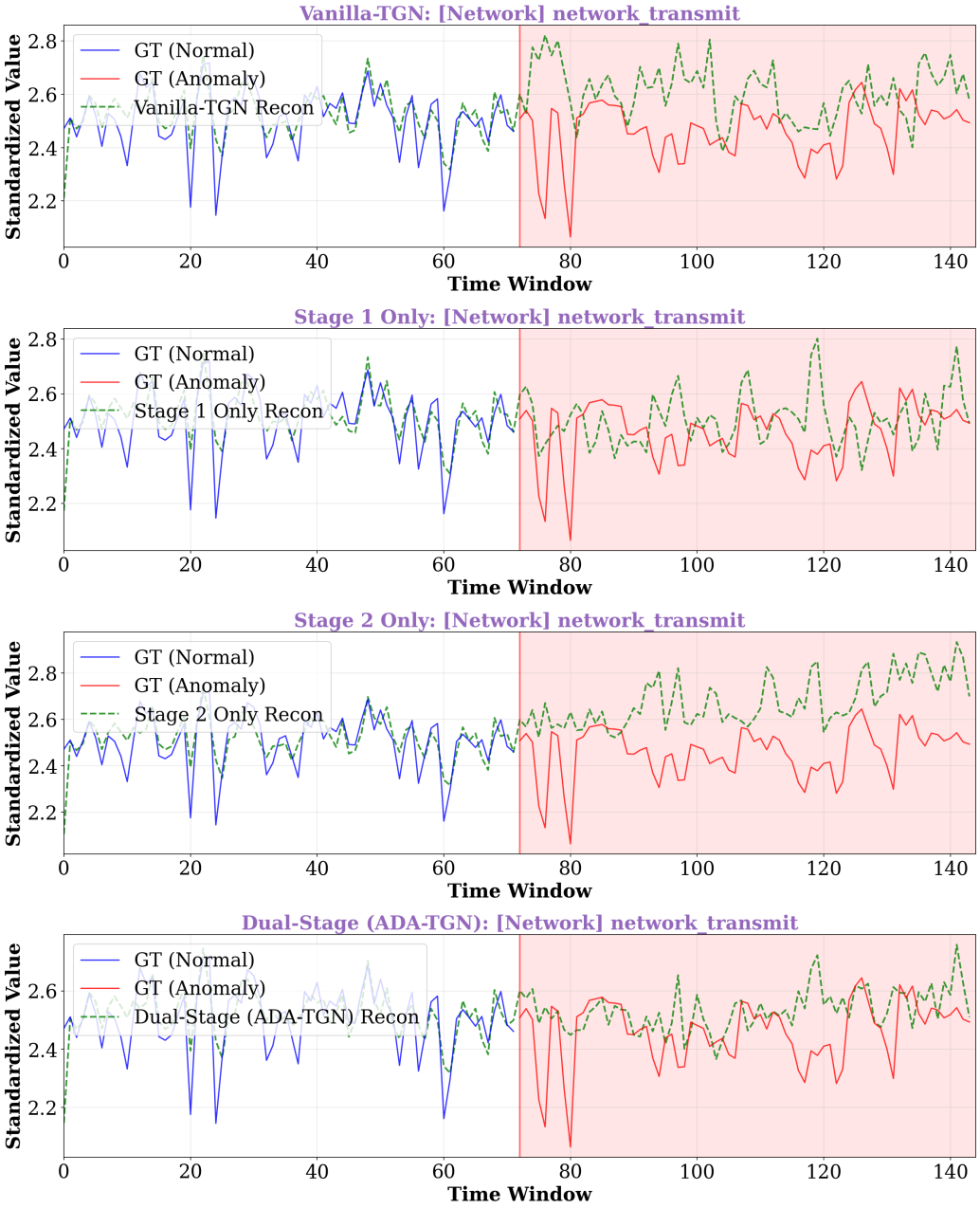


Figure 6.5: Reconstruction comparison for Network (container_network_transmit_bytes_total). From top to bottom: Vanilla-TGN, Stage 1 Only, Stage 2 Only, Dual-Stage (ADA-TGN).

6.2.3.1 Observations

Vanilla-TGN (Baseline): Memory contamination manifests in two distinct patterns during the anomaly period. The Latency group's reconstruction (Figure 6.2) exhibits **tracking behavior**, following anomalous ground truth and reducing reconstruction error, which degrades anomaly detectability.

In contrast, the Memory and CPU groups show **destabilization** caused by indirect contamination through the shared 256-dimensional memory representation. The Memory group (Figure 6.3) exhibits unstable reconstruction with unpredictable fluctuations, despite maintaining Alpha values at ≈ 1.0 (accepting all inputs). This destabilization is caused by contamination propagation from directly affected groups (CPU, Latency) through the shared memory representation (discussed in detail in Section 6.2.7). The CPU group (Figure 6.4) shows a different form of destabilization: **decoder output collapse** where reconstruction values drop to near-zero during the anomaly period. Notably, at the beginning of the anomaly period, the model can still output patterns similar to the normal period, as anomalous data has not yet accumulated significantly in memory. However, as time progresses and anomalous data accumulates in memory, the output gradually collapses toward zero. This occurs because the CPU group experiences direct anomaly impact (lower Alpha values), causing larger memory variations in that region, which amplifies the destabilization effect.

These distinct manifestations prevent stable detection of deviations from normal patterns.

Stage 1 Only (GRU Input Soft-Gating): This configuration achieves substantial improvement by filtering anomalous observations before they enter the GRU. As shown in Figures 6.2–6.5, reconstruction values remain consistently stable throughout the anomaly period, maintaining patterns close to the normal baseline. The key mechanism is that Stage 1 replaces anomalous observed values with predicted values when $\alpha \approx 0$, ensuring that GRU's internal computations are based on clean patterns. Since the GRU input is protected, the resulting updated memory $\mathbf{m}^{\text{upd}}$ remains uncontaminated. This clean updated memory is then directly adopted without Stage 2 filtering, but because the source itself is clean, **no contamination accumulates over time**, resulting in stable reconstruction throughout extended anomaly periods.

Stage 2 Only (Memory Soft-Gating): This configuration shows partial improvement over Vanilla-TGN, but reconstruction exhibits **gradual destabilization** during anomaly periods, particularly visible in the Latency and Memory groups. The fundamental limitation is that anomalous observations directly enter the GRU, causing the updated candidate memory $\mathbf{m}^{\text{upd}}$ to be contaminated. Stage 2 applies alpha-weighted blending to suppress contamination at each timestep:

$$\mathbf{m}^{(t)} = \alpha \cdot \mathbf{m}^{\text{upd}} + (1 - \alpha) \cdot \mathbf{m}^{(t-1)} \tag{6.3}$$

However, **as long as $\alpha > 0$, a portion of the contaminated candidate is adopted**, causing contamination to accumulate over time. While the contamination rate is slower than Vanilla-TGN, complete blockage is not achieved, and reconstruction gradually deviates from normal patterns.

Dual-Stage (ADA-TGN): The combination of both stages achieves the

best performance through synergistic protection at two critical points. Stage 1 blocks anomalous data at the input, ensuring clean GRU computations and preventing contamination at its source. Stage 2 provides an additional safety layer by filtering the memory update itself, catching any residual anomalies that may pass through Stage 1 due to threshold calibration or gradual drift. As shown in Figures 6.2–6.5, reconstruction maintains normal-period patterns throughout the anomaly period across all feature groups, providing consistently clear anomaly signals. This dual-layer architecture prevents both immediate contamination (Stage 1) and temporal accumulation (Stage 2), resulting in comprehensive memory protection.

6.2.4 Results: Alpha Value Dynamics

Figure 6.6 shows Alpha value dynamics for Dual-Stage (ADA-TGN) across all four feature groups.

6.2.4.1 Observations

The alpha dynamics reveal how different feature groups respond to anomalies, demonstrating the group-wise adaptivity of the Soft-Gating mechanism.

Latency group: Alpha values drop to approximately 0 immediately after anomaly injection and remain consistently suppressed throughout the anomaly period. This sharp and sustained suppression indicates that latency metrics deviate significantly from normal patterns. The consistently low alpha values ensure that GRU input is dominated by predicted values rather than anomalous observations, protecting memory from contamination at the source.

Memory group: Alpha values remain near 1.0 throughout both normal and anomaly periods, indicating that **memory metrics are not significantly affected** in this experiment. This demonstrates the benefit of group-wise gating: unaffected feature groups continue to update memory normally, preserving their learning capability even during partial anomalies. If global (node-level) gating were used instead, all groups would be suppressed simultaneously, unnecessarily blocking updates to unaffected metrics.

CPU group: Alpha values decrease during the anomaly period compared to the normal baseline, but exhibit **adaptive fluctuations** rather than consistent suppression like the Latency group. This fluctuation stems from two factors. First, CPU metrics are inherently variable, and as workload changes, reconstruction errors cross the threshold up and down, causing alpha to dynamically adjust. Second, **reconstruction accuracy limitations** play a role. While evaluation on the normal period benefits from data leakage with the training set (high reconstruction accuracy), the anomaly period requires predicting previously unseen values, resulting in relatively degraded reconstruction accuracy. Although the GRU learns

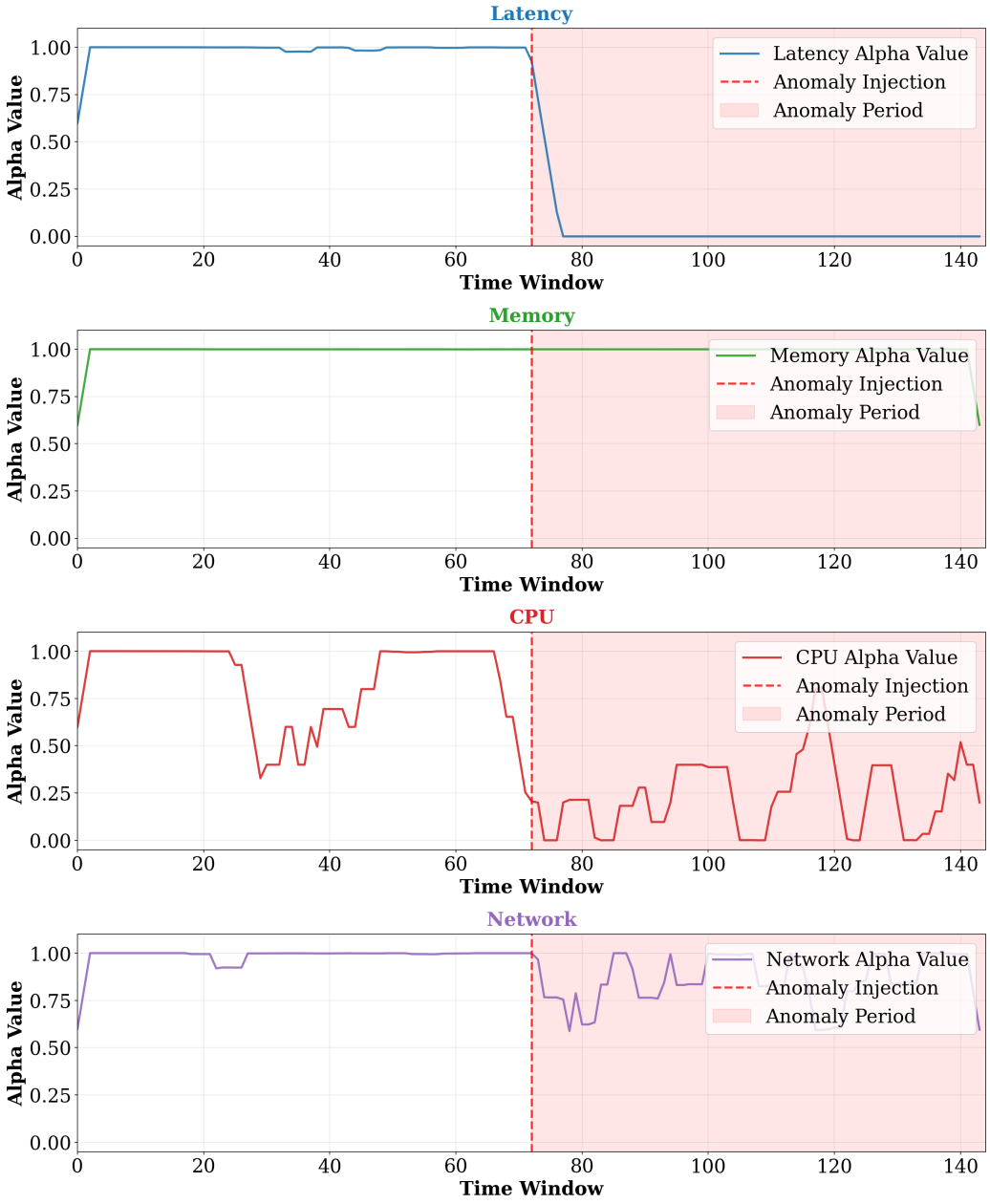


Figure 6.6: Alpha value dynamics for Dual-Stage (ADA-TGN) (currencyservice_cpu experiment, rep1). From top to bottom: Latency, Memory, CPU, Network groups.

temporal patterns (variable spike shapes), **when the phase shifts, peaks and valleys misalign**, and unexpected anomaly score increases can occur even when observed values are within the normal range. These two factors cause adaptive fluctuations in the CPU group's alpha.

Network group: Alpha values show adaptive fluctuations during both normal and anomaly periods, though the anomaly period exhibits somewhat lower values compared to the normal period. However, network metrics are not severely affected, and alpha values remain relatively higher than the CPU group. This fluctuation pattern is also attributable to the same two factors as the CPU group. First, network metrics are inherently variable, and as workload changes, reconstruction errors cross the threshold, causing alpha to dynamically adjust. Second, reconstruction accuracy limitations affect the results, and phase shifts can cause anomaly score increases even for normal observed values.

6.2.5 Qualitative Evaluation: Memory Contamination Analysis

Memory State Collection and PCA Projection Methodology:

To visualize memory contamination, we performed PCA projection following these steps:

1. **Memory state collection**: At each evaluation timestep, we recorded the model's internal memory states. For this analysis, we extracted the memory state $\mathbf{m}_v^{(t)} \in \mathbb{R}^{256}$ of the **root cause node** (currencyservice in this experiment), where anomaly impact is most pronounced, constructing a $T \times 256$ matrix across the entire period.

2. **Group-wise memory extraction**: The 256-dimensional memory vector is partitioned into four feature group subspaces: Latency group (dimensions 0–47, 48-dim), Memory group (dimensions 48–111, 64-dim), CPU group (dimensions 112–207, 96-dim), and Network group (dimensions 208–255, 48-dim).

3. **PCA learning**: For each feature group, we learned PCA using **only normal period memory states**:

$$\text{PCA} : \mathbb{R}^{d_g} \to \mathbb{R}^2 \tag{6.4}$$

   where d_g is the memory dimension of group g. Principal components are selected to maximize variance during the normal period.

4. **Projection to 2D**: Using the learned PCA, we projected memory states from both normal and anomaly periods into 2D space.

5. **95% confidence ellipse calculation**: For the projected 2D data, we computed separate 95% confidence ellipses for normal and anomaly periods. From each period's data, we calculated the covariance matrix $\mathbf{\Sigma}$ and determined principal axes and lengths via eigenvalue decomposition. The ellipse boundary is defined as:

$$(\mathbf{x} - \boldsymbol{\mu})^T \mathbf{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) = \chi^2_{0.95}(2) \tag{6.5}$$

where $\boldsymbol{\mu}$ is the mean vector and $\chi^2_{0.95}(2) = 5.991$ is the 95th percentile of the chi-squared distribution with 2 degrees of freedom.

Visualization interpretation: The spatial relationship between the blue ellipse (normal period) and red ellipse (anomaly period) provides visual assessment of memory contamination severity. Separated ellipses indicate formation of a new anomalous distribution, while overlapping ellipses demonstrate that memory remains within the normal region.

Latency Group Visualization:

Figure 6.7 visualizes memory states for the Latency group through PCA projection. We focus on this group in the main text for three reasons: (1) Alpha values drop to approximately 0 during the anomaly period (Figure 6.6), indicating the strongest gating response, (2) the deviation ratio shows the most pronounced contamination effect (Table 6.4), and (3) the direct anomaly impact results in the clearest ellipse separation in 2D projection. This makes the contamination-protection contrast across configurations clearly visible in PCA space.

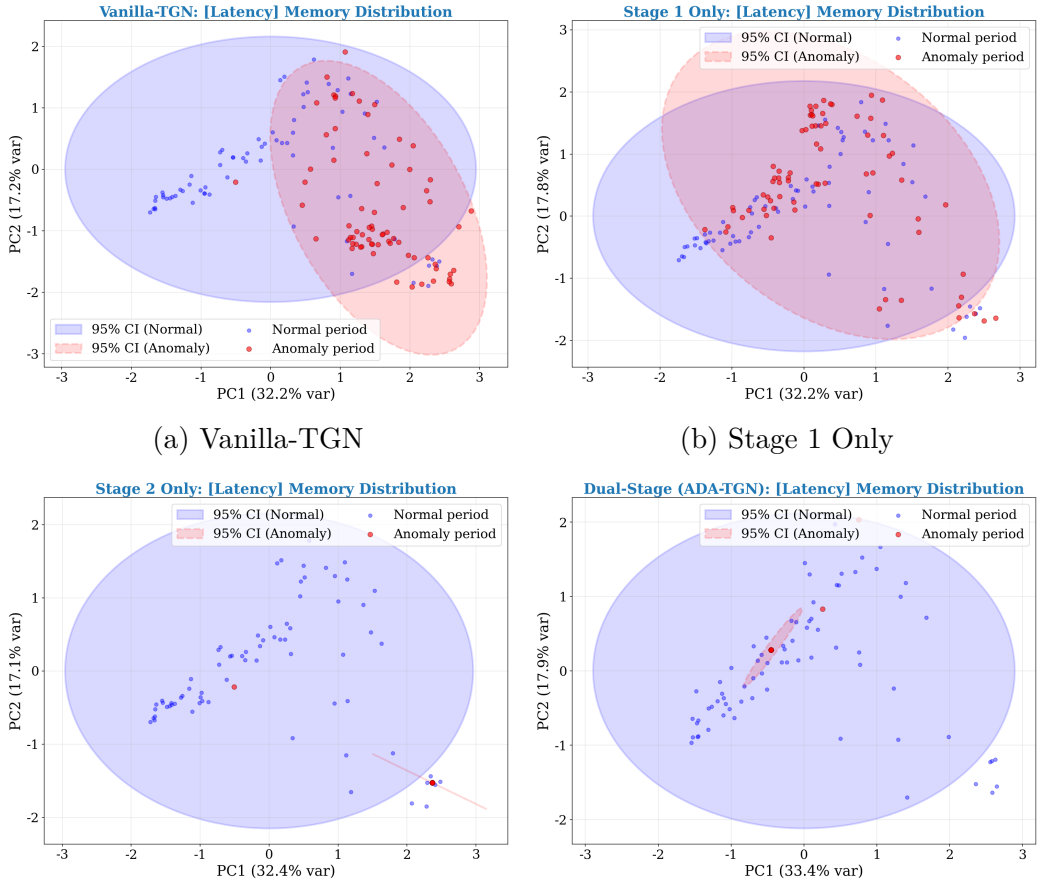
6.2.5.1 Observations

PCA visualization reveals how each configuration's memory states distribute in 2D space. The spatial relationship between the blue ellipse (normal period) and red ellipse (anomaly period) indicates the degree of memory contamination.

We focus on the Latency group in the main text for the following reasons. GRU parameters learned through RMS (Raw Message Store) training enable memory states to gradually incorporate anomalous data, rather than exhibiting abrupt distributional shifts. Combined with the less prominent anomaly characteristics in CPU, Memory, and Network groups compared to Latency (where alpha values drop to $\alpha \approx 0$), this results in PCA visualizations with substantial ellipse overlap for these groups. In contrast, the Latency group exhibits the clearest contamination-protection contrast in 2D projection due to the direct anomaly impact.

Below, we present observations for each configuration.

Vanilla-TGN (Baseline): The blue and red ellipses are clearly separated toward the bottom-right, indicating a significant distributional shift during the anomaly period. The normal period ellipse (blue) is located near



(a) Vanilla-TGN

(b) Stage 1 Only

(c) Stage 2 Only

(d) Dual-Stage (ADA-TGN)

Figure 6.7: Memory contamination analysis: Latency group. Blue ellipse: normal period 95% CI; Red ellipse: anomaly period 95% CI. Separated ellipses indicate memory contamination.

the center of the 2D space, while the anomaly period ellipse (red) has shifted substantially to the bottom-right, showing relatively clear separation. This demonstrates that unconditional memory updates cause memory states to completely deviate from the normal distribution and form a new anomalous distribution. Data points move abruptly at the anomaly onset and remain in a region far from the normal baseline throughout the anomaly period.

Stage 1 Only (GRU Input Soft-Gating): Ellipse overlap increases substantially, demonstrating that input filtering effectively reduces memory contamination. The anomaly period ellipse (red) still shows slight separation from the normal period ellipse (blue), but has moved significantly closer to the normal region compared to Vanilla-TGN. This is because Stage 1 blocks anomalous data at the GRU input stage, keeping the updated candidate memory clean. Data points move smoothly and remain near the normal region, visually confirming that contamination accumulation is suppressed.

Stage 2 Only (Memory Soft-Gating): The red ellipse is highly collapsed in the bottom-right region, with anomaly period data points being limited in number. This occurs because low alpha values ($\alpha \approx 0$) at many timesteps suppress memory updates, limiting memory state variation. However, critically, **when memory updates do occur (at timesteps where $\alpha > 0$), the updated candidate memory $\mathbf{m^{upd}}$ is already contaminated** because anomalous data has entered the GRU. As a result, even though updates are infrequent, they occur at positions deviated from the normal region (blue ellipse) toward the bottom-right. This observation is consistent with the mechanism explained in Section 6.2.3 (Reconstruction Stability): "as long as $\alpha > 0$, a portion of the contaminated candidate is adopted." The partial ellipse separation visually demonstrates that while update frequency is suppressed, the quality of memory when updates occur is not guaranteed.

Dual-Stage (ADA-TGN): The red ellipse is highly collapsed and nearly coincides with the blue ellipse, indicating that memory states remain within the normal region throughout the anomaly period. Through dual-stage protection combining Stage 1 (input protection) and Stage 2 (memory protection), the system achieves comprehensive defense: (1) blocking anomalous data at the GRU input stage, and (2) catching any residual anomalies at the memory update stage. Data points distribute in nearly the same region during both normal and anomaly periods, consistent with the low deviation ratio (0.66×) for the Latency group in Table 6.4. This visual evidence clearly demonstrates the synergistic effect of Dual-Stage protection.

6.2.6 Quantitative Evaluation: Memory Deviation Ratio

Table 6.4 quantifies memory contamination using std-normalized deviation ratio. The deviation metric measures how far memory states deviate from

the normal period centroid, normalized by the standard deviation of normal period memory states for cross-group comparability.

The specific calculation procedure is as follows:

1. **Calculate normal centroid**: Compute the mean $\boldsymbol{\mu}$ of normal period memory states
2. **Calculate normal std**: Compute the standard deviation σ of normal period memory states
3. **Calculate distances**: For each time t, compute the Euclidean distance $\|\mathbf{m}_t - \boldsymbol{\mu}\|_2$
4. **Compute normalized deviation**: Divide mean distance by σ for cross-group comparability
5. **Compute ratio**: Anomaly period deviation divided by normal period deviation

Table 6.4: Memory deviation ratio by configuration (std-normalized, currencyservice_cpu experiment, rep1)

Configuration	Stage 1	Stage 2	Latency	Memory	CPU	Network
Vanilla-TGN	–	–	2.67×	3.00×	2.23×	2.05×
Stage 1 Only	✓	–	1.31×	1.50×	**1.16**×	1.16×
Stage 2 Only	–	✓	2.06×	2.39×	1.66×	1.70×
Dual-Stage (ADA-TGN)	✓	✓	**0.66**×	**1.34**×	1.27×	**1.09**×

A deviation ratio near 1.0 indicates that memory states during the anomaly period remain within the normal region, which is consistent with stable reconstruction accuracy observed in Section 6.2.3. Conversely, ratios significantly above 1.0 indicate memory contamination. The results demonstrate that Dual-Stage (ADA-TGN) achieves the lowest deviation ratios across all feature groups, with the Latency group achieving a ratio below 1.0 (0.66×), confirming effective memory protection.

6.2.7 Discussion

The ablation study confirms that Dual-Stage Soft-Gating effectively addresses memory contamination. Stage 1 (GRU Input Soft-Gating) contributes more than Stage 2 (Memory Soft-Gating), reducing the deviation ratios in Table 6.4 by 43–51% and 17–26% compared to Vanilla-TGN, respectively. This is because filtering anomalous data before GRU entry prevents internal computation contamination. As shown in Sections 6.2.3 and 6.2.5, Stage 2 Only's limitation is that when $\alpha > 0$, the updated candidate memory

is already contaminated. Combining both stages provides defense in depth, achieving 43–75% contamination reduction across all groups.

Contamination Propagation Prevention. Notably, the Memory group analysis (Section 6.2.3, Figure 6.3) reveals a hidden benefit of the Dual-Stage mechanism. In the currencyservice_cpu anomaly experiment, the Memory group itself experienced no direct anomaly impact (Alpha values remained at $\approx$ 1.0 throughout), and GT values were stable. However, in Vanilla-TGN and Stage 2 Only, Memory reconstruction became unstable—despite all configurations accepting all Memory inputs (Alpha $\approx$ 1.0). This demonstrates contamination propagation: CPU/Latency groups experience direct anomaly impact (Latency Alpha values drop to ≈ 0; CPU Alpha shows adaptive fluctuations with lower values), causing memory contamination in those groups. Since TGN's memory is a shared 256-dimensional representation across all groups, contamination in CPU/Latency regions affects the entire memory vector processed by Graph Attention, indirectly destabilizing Memory reconstruction through contaminated embeddings.

ADA-TGN prevents this by protecting memory for directly affected groups (CPU, Latency) through Stage 1 gating (Alpha $\approx$ 0), keeping the shared memory representation clean. As a result, reconstruction for unaffected groups (Memory) is also indirectly stabilized. This demonstrates a key advantage of the Dual-Stage mechanism: group-specific Alpha gating identifies contamination sources and blocks them, preventing contamination propagation and improving overall system stability.

The following section evaluates how this effective memory protection translates into improved anomaly detection and root cause analysis performance.

6.3 Anomaly Detection and Root Cause Analysis Performance Evaluation

This section evaluates ADA-TGN's anomaly detection (AD) and root cause analysis (RCA) performance. BARO is used as the baseline for both tasks, with comparison under the same evaluation protocol.

6.3.1 Baseline Method: BARO

BARO [18] is used as the baseline method. BARO is an end-to-end approach that integrates anomaly detection and root cause analysis in microservices. It models dependencies in multivariate time series metrics using Bayesian Online Change Point Detection (BOCPD) to detect anomalies and identifies root causes using a non-parametric hypothesis testing method called RobustScorer. While BARO's original paper was evaluated on the fse-ob dataset (Online Boutique), which has different collection conditions from

RE2-OB used in this study, reference comparison is possible because both target the same application and fault types.

6.3.2 Anomaly Detection Performance

6.3.2.1 Detection Method

ADA-TGN outputs reconstruction errors for each feature group at each time step. Anomaly scores are computed using IQR standardization (Equation (6.2)) defined in Section 6.1.3, and dSPOT [23] is applied for anomaly detection.

Rationale for Choosing dSPOT: dSPOT is an adaptive threshold method based on Extreme Value Theory (EVT), with the following characteristics suitable for microservice anomaly detection:

- **Streaming Support**: Thresholds can be updated online, applicable to real-time monitoring.
- **Distribution-Free**: EVT is based on asymptotic distributions of extreme values, making no assumptions about data distribution. Microservice metrics often do not follow normal distributions, making this characteristic important.
- **Principled Threshold Setting**: Provides theoretically-grounded threshold determination, unlike arbitrary fixed thresholds.

dSPOT Parameter Settings:

- **Detection level** $q = 10^{-4}$: q represents the exceedance probability in EVT, controlling the upper bound of the probability of exceeding the threshold. Specifically, the threshold is set so that the probability of normal data exceeding it is at most q. We adopted the default value $q = 10^{-4}$ from the dSPOT implementation [1].
- **Initialization data size**: 50% of the normal period is used.
- **n_points, depth**: Dynamically computed from initialization data size N_{init}:

$$\text{n\_points} = \max(10, \lfloor 0.1 \times N_{\text{init}} \rfloor) \tag{6.6}$$

$$\text{depth} = \max(10, \lfloor 0.6 \times N_{\text{init}} \rfloor) \tag{6.7}$$

  n_points is the number of samples used for Generalized Pareto Distribution (GPD) parameter estimation, and depth is the window size for the moving average used for drift correction. In this experiment, the normal period is approximately 60 windows with $N_{\text{init}} = 30$, yielding n_points = 10 and depth = 18.

6.3.2.2 Evaluation Protocol

The evaluation protocol is as follows:

1. Apply dSPOT to anomaly scores computed during the normal period to adaptively compute thresholds
2. During the anomaly period, stop threshold updates and use the final threshold computed from the normal period for anomaly determination (to prevent threshold drift toward anomalous values)
3. Record the first time point of anomaly detection during the anomaly period
4. Following BARO [18], detection within the first 300 seconds of the anomaly period is True Positive (TP), detection during the normal period is False Positive (FP), and failure to detect within the first 300 seconds of the anomaly period is False Negative (FN)

6.3.2.3 Evaluation Metrics

- **Precision**:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{6.8}$$

- **Recall**:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{6.9}$$

- **F1 Score**:

$$\text{F1} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{6.10}$$

6.3.2.4 Results

Table 6.5 shows the comparison of anomaly detection performance.

Table 6.5: Comparison of anomaly detection performance (Online Boutique)

Method	Dataset	#Exp.	Precision	Recall	F1
BARO [18]	fse-ob†	100	0.760	1.000	0.870
ADA-TGN + dSPOT (Ours)	**RE2-OB**	**60**	**0.923**	**1.000**	**0.960**

† Dataset used in BARO's original paper (FSE'24). Presented as reference values due to different collection conditions from RE2-OB.

ADA-TGN + dSPOT achieved Precision of 0.923, Recall of 1.000, and F1 score of 0.960. Anomalies were detected in all 60 experiments (FN=0), with false positives (FP) occurring during the normal period in 5 experiments.

Compared to BARO, this shows improvements of +21.5% in Precision (0.760 → 0.923) and +10.3% in F1 score (0.870 → 0.960). Both methods achieved Recall of 1.000, detecting all anomalies.

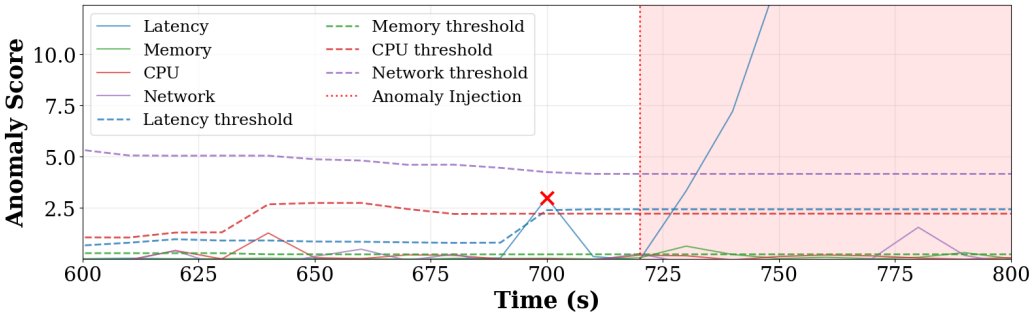
Note on Datasets: BARO results are from reproduction experiments on the fse-ob dataset using BARO's public repository[2]. fse-ob is the Online Boutique dataset collected and used in BARO's original paper (FSE'24), with different collection timing, load patterns, and experimental design from RE2-OB (collected by RCAEval [19]) used in this study. Therefore, this comparison should be interpreted as reference values indicating relative performance trends between the two methods, not as a strict direct comparison. Both datasets target the Google Online Boutique application and include the same fault types (CPU, MEM, DELAY, LOSS), ensuring a certain degree of comparability.

6.3.2.5 Analysis of False Positives

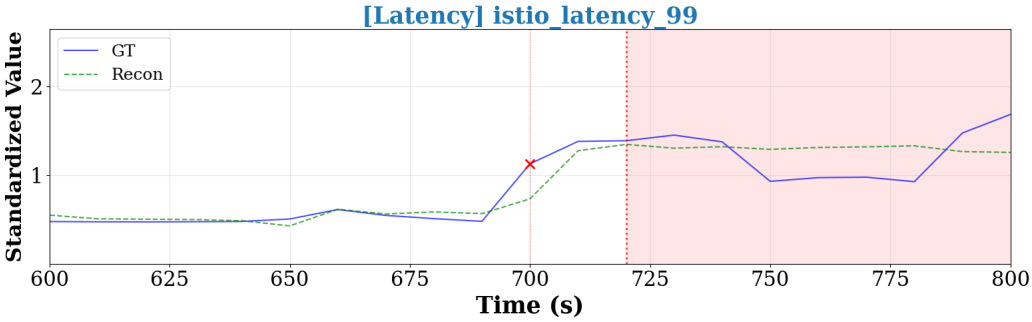
We analyze cases where false positives (FP) occur during the normal period in anomaly detection using dSPOT. As defined in Equation (4.6), the anomaly score for each feature group adopts the maximum reconstruction error among features within the group (MAX aggregation). Therefore, if any single feature within a group shows large discrepancy between GT and reconstruction value, that group's anomaly score increases, causing threshold exceedance.

Figure 6.8 shows a representative FP case. In the Latency group, outliers occur in istio_latency_99, and the reconstruction cannot follow these sudden spikes, resulting in large reconstruction errors.

[2] https://github.com/phamquiluan/baro



(a) Anomaly score and threshold time series



(b) GT (blue) vs reconstruction (green) for istio_latency_99

Figure 6.8: FP case in Latency group. Anomaly score and threshold time series for frontend service during the productcatalogservice_cpu experiment (rep3), where FP was detected on this service. Red X: FP detection point; Pink background: anomaly period. (a) Shows the anomaly score exceeding the threshold. (b) Shows GT and reconstruction for istio_latency_99, where the reconstruction cannot track sudden spikes in the normal period.

These FPs are attributed to dSPOT's threshold update mechanism. As shown in Figure 6.8(a), when sudden outliers (temporary increases in reconstruction error) occur during the normal period, dSPOT's threshold updates cannot keep pace, causing false detections. This demonstrates the following limitations of dSPOT:

- **Dependence on Historical Data**: dSPOT estimates thresholds based on extreme values observed in the past, following Extreme Value Theory (EVT). Therefore, it cannot immediately respond to sudden anomaly score fluctuations unprecedented in the past.
- **Threshold Update Speed Constraints**: The threshold update speed may not keep pace with the rate of reconstruction error fluctuation. In particular, instantaneous spikes during the normal period (traffic surges, load fluctuations, etc.) may be detected before the threshold adapts.

While ADA-TGN's reconstruction values themselves maintain normal patterns, discrepancies from instantaneous GT fluctuations exceed dSPOT's

threshold, generating FPs. To reduce such FPs, threshold adjustment or introduction of consecutive detection conditions could be considered, but this would involve trade-offs with Recall. This study adopted a Recall-focused design, accepting a certain number of FPs.

6.3.2.6 Discussion

ADA-TGN + dSPOT achieved Precision of 0.923, Recall of 1.000, and F1 score of 0.960, showing improvements of +21.5% in Precision and +10.3% in F1 compared to BARO.

The main cause of False Positives is cases where dSPOT's threshold updates cannot keep pace with sudden load fluctuations during the normal period. This is due to dSPOT's characteristic of dependence on historical data. While threshold adjustment or introduction of consecutive detection conditions could reduce FPs, this would involve trade-offs with Recall, so this study adopted a Recall-focused design.

6.3.3 Root Cause Analysis Performance

6.3.3.1 RCA Method

RCA by ADA-TGN is based on reconstruction errors of each service (node). After anomaly detection, reconstruction errors of each service are IQR-standardized using Equation (6.2), and services are ranked in descending order of standardized scores. The service with the highest score is considered the first root cause candidate.

6.3.3.2 Evaluation Protocol

After anomaly detection, the ranking of the true root cause (anomaly injection target service) is evaluated based on the service ranking output by each method.

6.3.3.3 Evaluation Metrics

- **AC@k** (Accuracy at k): The proportion of cases where the true root cause is included in the Top-k ranking.

$$\mathrm{AC@}k = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}[\mathrm{rank}_i \leq k] \tag{6.11}$$

  where $\mathbf{1}[\cdot]$ is the indicator function that returns 1 if the condition is true and 0 otherwise.

- **Avg@k** (Average at k): The cumulative average from AC@1 to AC@k.

$$\text{Avg@}k = \frac{1}{k}\sum_{j=1}^{k}\text{AC@}j \tag{6.12}$$

6.3.3.4 Results

Table 6.6 shows Avg@5 by fault type. Evaluation follows BARO's evaluation protocol, using only the first 300 seconds (30 windows at 10-second granularity) of the anomaly period. BARO's performance values are results re-evaluated on the RE2-OB dataset using the RCAEval [19] framework[3].

Table 6.6: Root cause analysis performance by fault type (Avg@5)

Method	CPU	MEM	DELAY	LOSS	Overall
BARO [18]	0.640	0.870	0.670	0.800	0.745
ADA-TGN (Ours)	**0.920**	**0.960**	**0.787**	**0.933**	**0.900**

ADA-TGN achieved Avg@5 of 0.900 overall, exceeding BARO's 0.745 by 20.8%. ADA-TGN outperformed BARO across all fault types. Particularly notable improvements were shown for CPU anomalies at 43.8% (0.640 → 0.920) and LOSS anomalies at 16.6% (0.800 → 0.933).

6.3.3.5 Discussion

ADA-TGN achieved Avg@5 of 0.900 overall, exceeding BARO's 0.745 by 20.8%. ADA-TGN outperformed BARO across all fault types.

Extremely high performance was achieved for CPU anomalies (Avg@5: 0.920), MEM anomalies (Avg@5: 0.960), and LOSS anomalies (Avg@5: 0.933). These anomalies are directly observable as resource consumption or packet loss, showing the most prominent anomaly scores at the root cause service.

On the other hand, while ADA-TGN outperforms BARO for DELAY anomalies (0.787 vs 0.670), performance remains lower compared to other fault types. This is due to the **cascade effect**, where latency anomalies in downstream services propagate and amplify upstream along service dependencies, causing upstream services to show higher anomaly scores than the root cause service. Figure 6.9 shows a concrete example of this phenomenon.

[3]The default evaluation command (`python main.py --method baro --dataset re2-ob`) targets all 90 experiments (5 services × 6 anomaly types × 3 replications) in RE2-OB. For fair comparison with ADA-TGN, we added the `--faults` option to specify anomaly types in the evaluation script and evaluated only the same 60 experiments (5 services × 4 anomaly types × 3 replications).

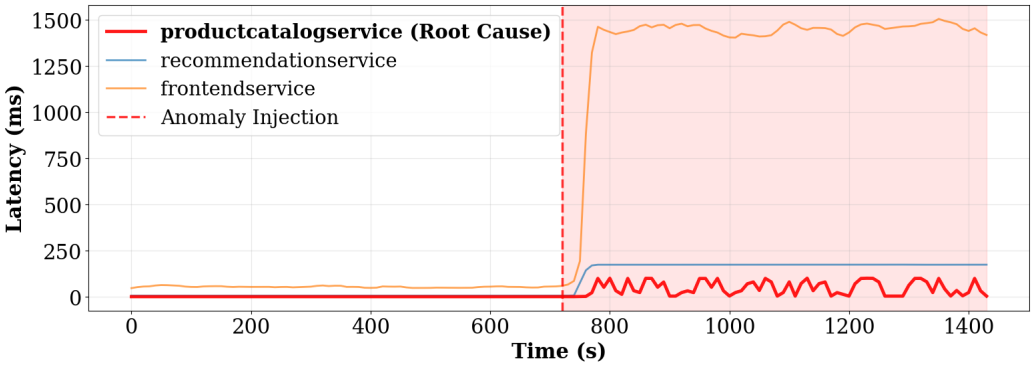


Figure 6.9: Latency cascade phenomenon (productcatalogservice_delay experiment). Upstream services show larger latency increases than the root cause service (productcatalogservice). See Figure 5.3 for service dependencies.

The limitations of this score-based RCA and improvement directions utilizing graph structure are detailed in Section 7.2.1.

Chapter 7

Conclusion

This chapter summarizes the achievements of this research and discusses limitations and future work.

7.1 Summary

This study proposed ADA-TGN (Temporal Graph Networks with Adaptive Memory Protection), a novel method for anomaly detection and root cause analysis in microservice environments.

Temporal Graph Networks (TGN) are well-suited for modeling microservices due to their ability to capture temporal patterns in spatio-temporal graph-structured data. However, when applying TGN to anomaly detection, a fundamental problem arises: anomalous data incorporated into memory contaminates the memory state, causing the model to learn anomalous patterns as "normal" and degrading detection capability.

This study formalizes the **memory contamination problem for GRU-based TGN** and proposes the **Soft-Gating mechanism** as a solution. The Soft-Gating mechanism dynamically controls memory update intensity based on reconstruction errors, suppressing anomalous data incorporation while enabling continuous learning from normal data. Dual-stage protection—applied at both GRU input and memory output—provides comprehensive defense against contamination.

Ablation studies confirmed that input-stage gating (Stage 1) provides effective protection (43–51% reduction). The Dual-Stage combination provides more stringent protection, particularly evident in scenarios requiring strong suppression (e.g., Latency group with 75% reduction), ensuring memory remains within the normal region.

Evaluation on the RE2-OB dataset (Online Boutique, 60 experiments) demonstrated ADA-TGN's effectiveness, achieving an F1 score of 0.960 for anomaly detection and Avg@5 of 0.900 for root cause analysis. Compared to the baseline method BARO, ADA-TGN showed substantial improvement in RCA performance (+20.8%).

7.2 Limitations and Future Work

This section discusses the limitations of this study and future research directions to overcome them.

7.2.1 Limitations of Score-Based RCA and Extension to Graph-Based RCA

7.2.1.1 Limitations

ADA-TGN's RCA is a ranking method based on anomaly scores of each service. This method identifies the service with the highest anomaly score as the root cause, but in microservice environments, due to the **cascade effect**, upstream services may show higher anomaly scores than the root cause service. As shown in Section 6.3.3, this reversal phenomenon is prominent for DELAY anomalies, and ADA-TGN's Avg@5 remained at 0.787.

7.2.1.2 Future Directions

To overcome this limitation, integration with causal inference approaches utilizing graph structure is effective. For example, MicroRCA [25] combines attributed graphs with Personalized PageRank to achieve ranking that considers anomaly propagation patterns. By combining anomaly scores computed by ADA-TGN for each service as node attributes with such graph-based methods, it may be possible to address the reversal phenomenon caused by cascade effects.

7.2.2 Evaluation on a Single Dataset and Verification of Generalization Performance

7.2.2.1 Limitations

The evaluation of this study was conducted only on the RE2-OB (Online Boutique) dataset. While Online Boutique is a representative microservice benchmark, it does not fully reflect the complexity of production environments.

Additionally, in RE2-OB, the service topology is static (fixed 10 services with invariant dependencies), and only node features change over time. TGN is originally designed to handle dynamic topology changes including edge and node additions/deletions [21], but this capability was not fully utilized in this experiment. In production environments, topology changes dynamically due to service scale-out/in, deployments, and service outages due to failures. The effectiveness of ADA-TGN in such environments remains unverified.

7.2.2.2 Future Directions

To verify the generalizability of the method, additional evaluation on the following datasets is necessary:

- **Sock-Shop**: Another microservice benchmark
- **Train-Ticket**: Large-scale microservice system
- **Industrial Datasets**: Production data from real-world cloud environments

In particular, it is important to verify the scalability and generalization performance of ADA-TGN through evaluation on datasets with different numbers of services and topology structures.

7.2.3 Use of Metrics Only and Multi-Modal Data Fusion

7.2.3.1 Limitations

ADA-TGN uses only metrics (CPU, memory, latency, etc.) as input. However, in microservice environments, logs and traces also contain important diagnostic information. Especially for anomalies with sparse metric features such as SOCKET failures, utilizing logs and traces is essential for improving detection accuracy.

7.2.3.2 Future Directions

Multi-modal anomaly detection integrating metrics, logs, and traces has emerged as a rapidly advancing research direction in software engineering. The following studies, published at top-tier venues, demonstrate a progressive path toward comprehensive multi-modal integration that is directly applicable to the future development of ADA-TGN.

Dynamic Graph Construction from Traces MicroHECL [15], presented at ICSE 2021, constructs dynamic service call graphs incrementally from distributed traces and maps metric time series (response time, error count, QPS) onto the graph edges. This on-demand approach achieved a Top-3 hit rate of 68% for root cause localization in Alibaba's production environment with over 30,000 services. ADA-TGN currently relies on a static graph derived from predefined service dependencies; replacing this with trace-based dynamic graph construction would enable adaptation to runtime topology changes.

Log Semantics as Node Features DeepTraLog [29], presented at ICSE 2022, transforms unstructured log messages into event vectors (300-dimensional embeddings) through log parsing and event embedding, and integrates them as node features within a Gated Graph Neural Network (GGNN). By unifying trace-derived span events and log events into a single Trace Event Graph, DeepTraLog achieves combined anomaly detection within a unified GNN framework. Augmenting ADA-TGN's 12-dimensional metric features with log-derived semantic features would enable capturing code-level anomaly context that metrics alone cannot express.

Unified Multi-Modal Integration Nezha [27], presented at ESEC/FSE 2023, converts metrics, logs, and traces—three fundamentally heterogeneous data sources—into a homogeneous event representation. By mining event patterns from this unified event graph and comparing patterns between fault-free and fault-suffering phases, Nezha achieves fine-grained root cause analysis at the code region and resource type level, with an average Top-1 accuracy of 89.77%. Ablation studies confirmed that removing any single modality degrades performance, quantitatively demonstrating the critical contribution of integrating all three modalities.

Emerging Trends More recently, advanced data fusion mechanisms are being explored. FAMOS [9] (ICSE 2025) employs cross-attention mechanisms to dynamically capture inter-modality relationships, improving F1 scores by 20.33% over feature concatenation baselines. Furthermore, LLM-based approaches such as CloudAnoAgent [30] (FSE 2024) process metrics and logs as multi-modal inputs through autonomous reasoning agents. Graph-based methods and large language models possess complementary strengths, and their effective fusion remains an open challenge and an important research direction in multi-modal root cause analysis.

7.2.4 Lack of Concept Drift Handling and Integration of Adaptation Mechanisms

7.2.4.1 Limitations

This study assumed that data distribution remains constant between the training and test phases. However, in production environments, **concept drift** occurs where the statistical properties of data change over time due to system updates, changes in load patterns, and service configuration changes. The current ADA-TGN lacks adaptation mechanisms for such environmental changes.

7.2.4.2 Future Directions

For application to production environments, integration of adaptation mechanisms for concept drift is essential. Recent GNN-based drift adaptation methods are rapidly advancing, and the following approaches are candidates for integration with ADA-TGN.

Statistical Drift Detection ADWIN (Adaptive Windowing) [4] is a method that statistically monitors the difference in mean values between two sliding windows based on Hoeffding bounds. In FedST-GNN, ADWIN triggers a meta-learning loop the moment drift is detected, achieving fast recovery. This combination of statistical detection and meta-learning can be naturally integrated into ADA-TGN's anomaly score monitoring.

Dynamic Graph Regularization MSGR (Multi-stream Self-adaptation Graph Regularization) [32] uses Gumbel sampling and adaptive matrices to dynamically construct correlation graphs from newly arriving samples. Upon drift detection, the regularization coefficient λ_{drift} is adjusted to prioritize adaptation to the new data distribution:

$$\mathcal{L}_{total} = \mathcal{L}_{task} + \lambda_{drift} \cdot \Omega(G_{adj}, \Theta) \tag{7.1}$$

This dynamic graph generation is effective for adaptation to microservice topology changes.

Continuous Graph Learning CGLM (Continuous Graph Learning-based self-adaptation for Multistream) [31] clearly separates lightweight fine-tuning in "non-drift states" where drift has not occurred from "reconstruction" upon drift detection. ADGAT (Adaptive Diffusion Graph Attention module) adaptively updates correlation graph weights based on the dynamic graph. This two-pronged adaptation mechanism achieves both computational cost reduction and resilience to environmental changes.

When integrating these methods, important implementation considerations include adjusting drift detection sensitivity to account for noise characteristics in microservice environments, and managing the computational overhead of adaptation processes to avoid impacting service performance.

7.3 Concluding Remarks

This study proposed ADA-TGN for anomaly detection and root cause analysis in microservice environments and experimentally verified its effectiveness. Memory protection through the Soft-Gating mechanism presents one solution to an essential challenge in reconstruction-based anomaly detection methods.

With the proliferation of microservice architectures, ensuring distributed system reliability has become an increasingly important challenge. The integration of graph neural networks and time series modeling demonstrated in this study is an effective approach for monitoring and diagnosing systems with complex dependencies.

In the future, it is expected that applicability to production environments will be enhanced through the integration of graph-based RCA, multi-modal data fusion, and concept drift adaptation.

Appendix A

Understanding Memory Contamination Dynamics in RMS-Trained Models

A.1 Research Question and Main Claim

Section 6.2.5 presented PCA visualizations only for the Latency group. During our ablation study, we observed an interesting phenomenon regarding GRU parameter training. While this is not the main focus of our research, we report these observations as supplementary findings that provide insights into memory contamination dynamics.

The purpose of this appendix is to **observe how the presence or absence of GRU parameter training affects contamination progression speed**. Specifically, we compare RMS-trained (with GRU learning) and Non-RMS (without GRU learning) Vanilla-TGN. Contamination occurs in both cases, but the progression dynamics differ: when GRU is not trained, the fixed parameters rapidly incorporate new anomalous patterns, leading to abrupt contamination progression; when GRU is trained, adaptive incorporation results in gradual contamination progression.

Key observation: In *Vanilla-TGN*, contamination occurs regardless of training strategy, but the contamination progression speed differs depending on whether GRU parameters are trained:

	Non-RMS Vanilla	**RMS Vanilla**
GRU Parameters	Not trained	Trained
Contamination Progression	Abrupt	Gradual
2D PCA Separation	Clear	Unclear

While total contamination levels are similar (Table A.1), adaptive memory dynamics in RMS training cause contamination to progress gradually (Figures A.1–A.4).

We observe this phenomenon through three perspectives. First, **time-series analysis of reconstruction behavior** clearly distinguishes abrupt degradation (Non-RMS) from gradual degradation (RMS). Second, **PCA visualization** shows clear separation in 2D space for Non-RMS but overlap for RMS. This is because RMS's gradual contamination disperses across multiple dimensions. Third, **quantitative metrics in 256-dimensional space** (Table 6.4, Section 6.2.6) confirm similar contamination levels across both training strategies, despite unclear separation in 2D PCA.

A.2 Reconstruction Behavior Comparison

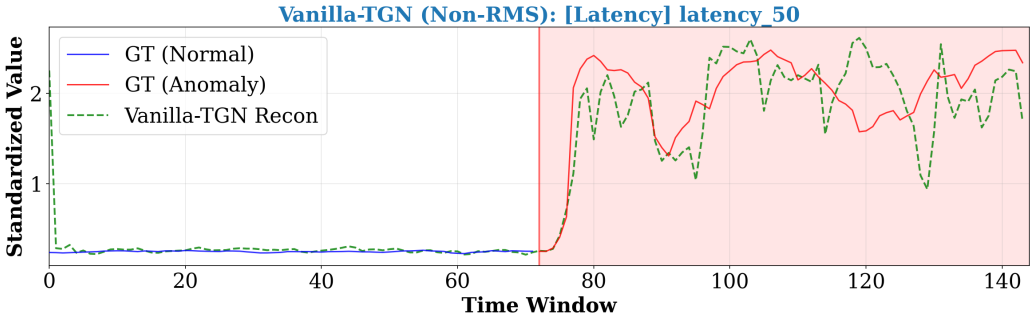
We compare reconstruction behavior between two configurations to visualize the difference in contamination progression speed: **Non-RMS Vanilla-TGN** exhibits abrupt reconstruction instability, while **RMS Vanilla-TGN** shows gradual reconstruction instability. All comparisons use the same `currencyservice_cpu` experiment (rep1) analyzed in Section 6.2.

Latency Group

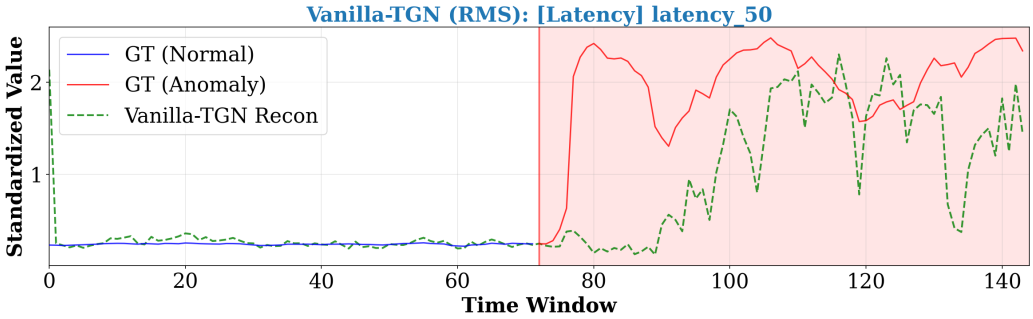
Figure A.1 compares reconstruction behavior for the Latency group between Non-RMS and RMS Vanilla-TGN.

Non-RMS Vanilla-TGN (Figure A.1a): At the anomaly onset (vertical dashed line), reconstruction quality degrades abruptly and severely. The error immediately jumps to a high level and remains unstable.

RMS Vanilla-TGN (Figure A.1b): Reconstruction quality degrades gradually after the anomaly onset. The error increases progressively rather than jumping immediately.



(a) Non-RMS Vanilla-TGN



(b) RMS Vanilla-TGN

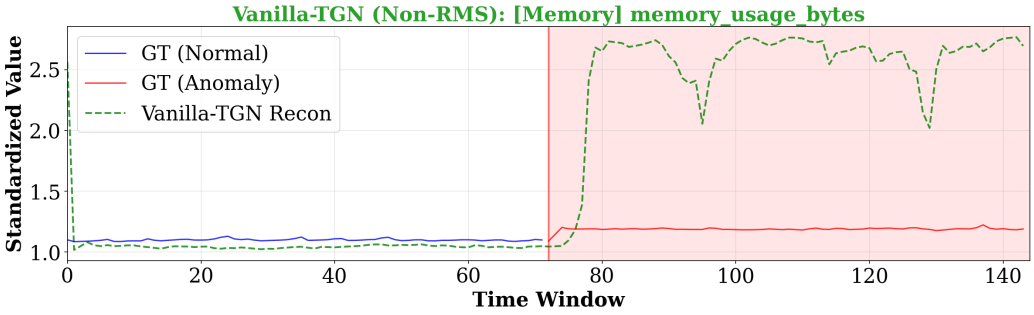
Figure A.1: Latency group reconstruction comparison (istio_latency_50)

Memory Group

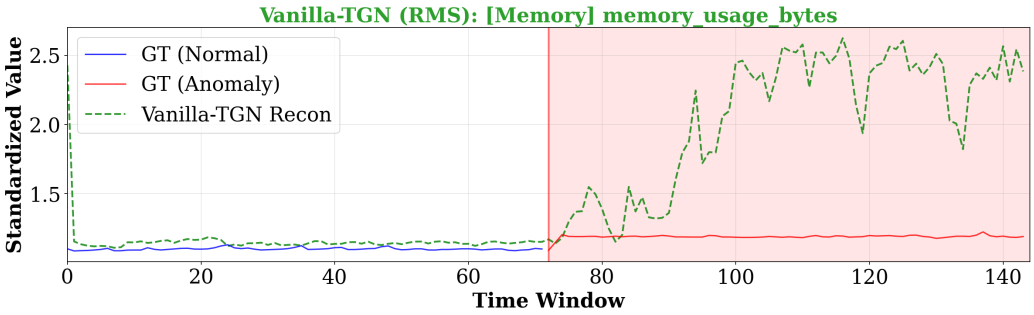
Figure A.2 shows reconstruction behavior for the Memory group. Despite Memory features themselves not being directly anomalous, reconstruction quality is affected:

Non-RMS Vanilla-TGN (Figure A.2a): Abrupt destabilization occurs even though Memory features are not directly anomalous. This demonstrates that CPU/Latency memory contamination indirectly destabilizes Memory reconstruction through the shared 256-dimensional memory representation (detailed mechanism discussed in Section 6.2.3).

RMS Vanilla-TGN (Figure A.2b): Gradual destabilization with smoother transitions.



(a) Non-RMS Vanilla-TGN



(b) RMS Vanilla-TGN

Figure A.2: Memory group reconstruction comparison (container_memory_usage_bytes)

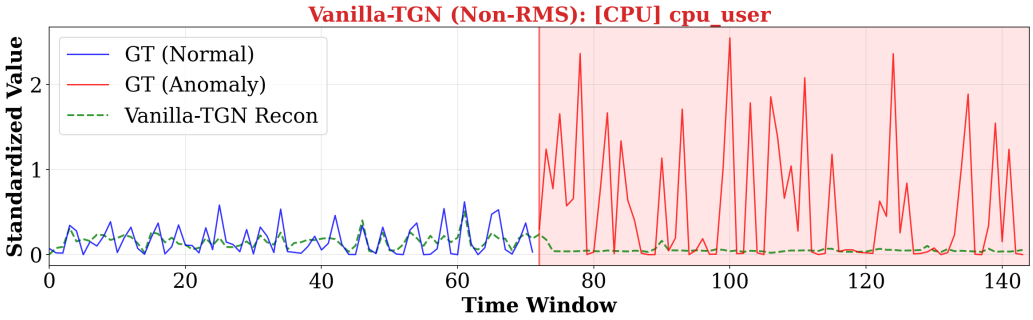
CPU Group

Figure A.3 shows reconstruction behavior for the CPU group. As explained in Section 6.2.3, the CPU group exhibits **decoder output collapse**: reconstruction values drop to near-zero during the anomaly period.

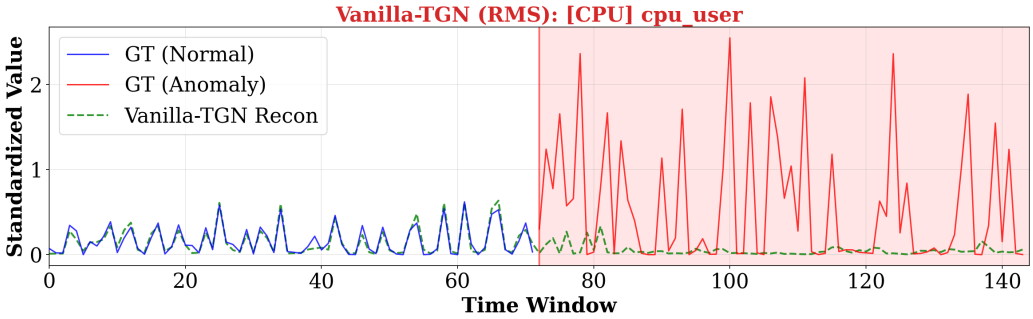
Non-RMS Vanilla-TGN: Collapse occurs immediately at anomaly onset.

RMS Vanilla-TGN: Gradual degradation leading to progressive output collapse.

This pattern demonstrates that GRU training affects contamination progression speed across different manifestations of instability.



(a) Non-RMS Vanilla-TGN

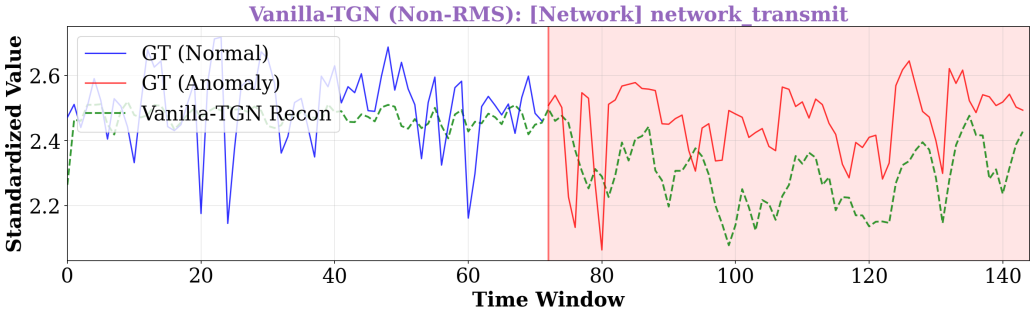


(b) RMS Vanilla-TGN

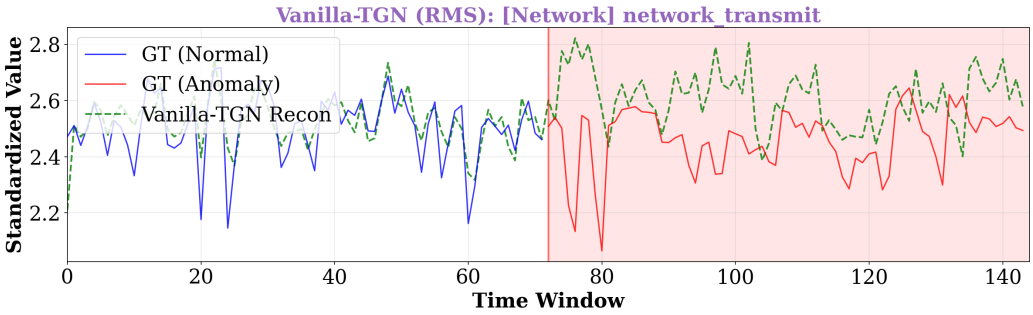
Figure A.3: CPU group reconstruction comparison (container_cpu_user_seconds_total)

Network Group

Figure A.4 shows reconstruction behavior for the Network group. It exhibits tracking patterns similar to the Latency group, with Non-RMS showing abrupt degradation and RMS showing gradual degradation.



(a) Non-RMS Vanilla-TGN



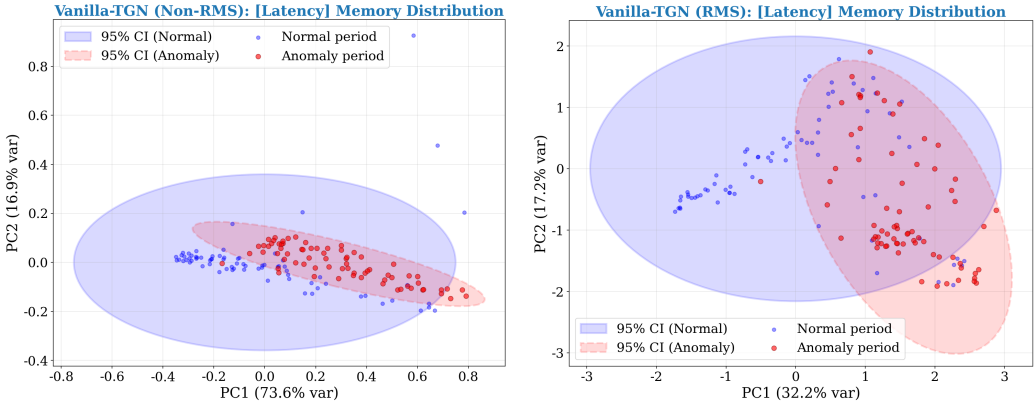
(b) RMS Vanilla-TGN

Figure A.4: Network group reconstruction comparison (container_network_transmit_bytes_total)

These observations show consistent patterns across four different feature groups: Non-RMS Vanilla-TGN exhibits abrupt degradation, while RMS Vanilla-TGN shows gradual degradation. This consistency provides direct evidence of the impact of GRU training on contamination progression dynamics.

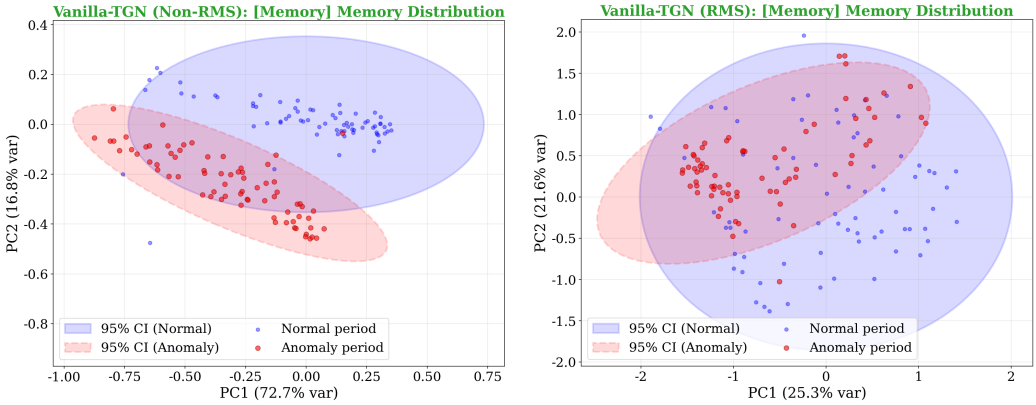
A.3 PCA Visualization Analysis

Figures A.5–A.8 compare PCA visualizations between Non-RMS and RMS Vanilla-TGN across all four feature groups.



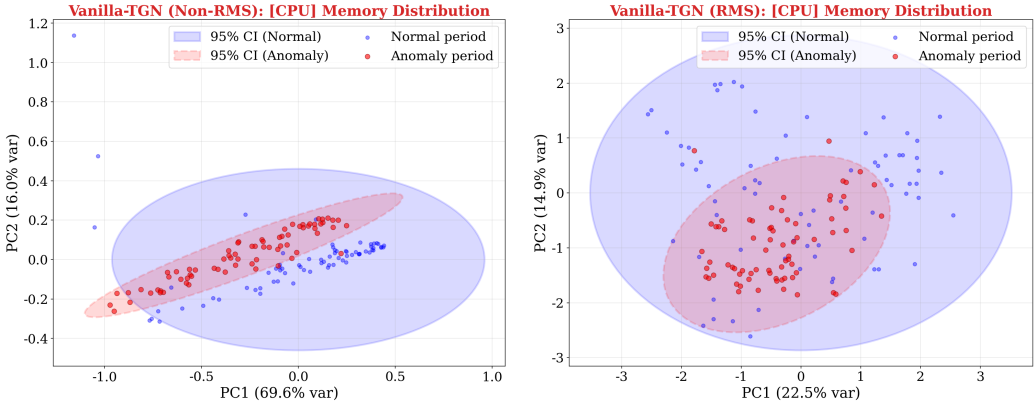
(a) Non-RMS Vanilla-TGN (b) RMS Vanilla-TGN

Figure A.5: Latency group PCA comparison



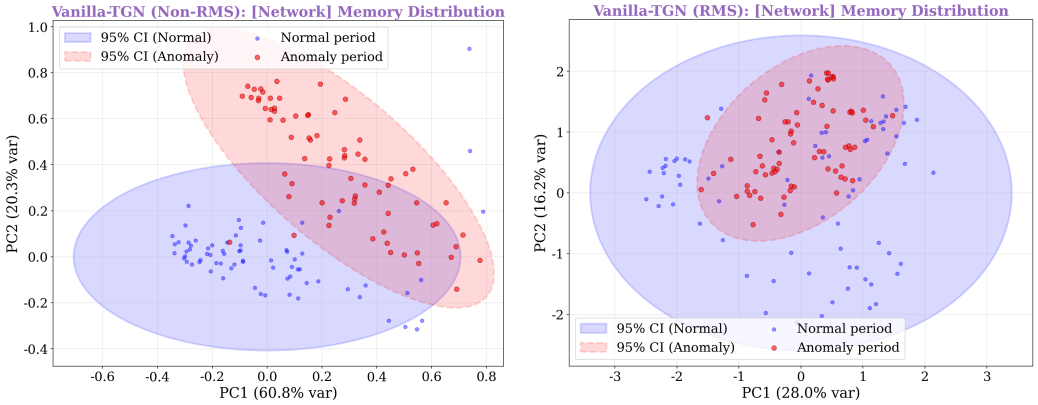
(a) Non-RMS Vanilla-TGN (b) RMS Vanilla-TGN

Figure A.6: Memory group PCA comparison



(a) Non-RMS Vanilla-TGN (b) RMS Vanilla-TGN

Figure A.7: CPU group PCA comparison



(a) Non-RMS Vanilla-TGN (b) RMS Vanilla-TGN

Figure A.8: Network group PCA comparison

Non-RMS Vanilla-TGN (Figures A.5a, A.6a, A.7a, A.8a): The ellipses for normal (blue) and anomaly (red) periods are clearly separated across all groups. PC1+PC2 explain approximately 90% of variance, indicating that memory state variations follow a low-dimensional structure. This demonstrates that abrupt contamination progression (Section A.2) with fixed GRU parameters produces distinctly different memory states between normal and anomaly periods, and this distributional difference is clearly observable even in 2D projection.

RMS Vanilla-TGN (Figures A.5b, A.6b, A.7b, A.8b): The ellipses for normal and anomaly periods substantially overlap across all groups. PC1+PC2 explain only approximately 50% of variance, indicating that memory state variations are high-dimensional. This demonstrates that gradual contamination progression (Section A.2) through GRU learning produces complex memory state changes, where most variations cannot be captured by PC1-PC2.

Interpretation: The limited separation in 2D PCA for RMS Vanilla-TGN does not indicate absence of contamination. Rather, it reflects that contamination-induced variations are high-dimensional and complex, not fully representable in 2D projection. Indeed, quantitative evaluation in 256-dimensional space (Table A.1) confirms similar contamination levels between Non-RMS and RMS.

This consistent pattern—abrupt degradation with clear 2D separation for Non-RMS, gradual degradation with unclear 2D separation for RMS—is observed across all four feature groups. This provides direct evidence of GRU learning's impact on contamination progression dynamics.

A.4 Quantitative Validation

A.4.1 Deviation Ratio Validation

Table A.1: Deviation ratio comparison: Non-RMS vs RMS Vanilla-TGN

Configuration	Latency	Memory	CPU	Network
Non-RMS Vanilla-TGN	2.76×	2.73×	2.84×	3.29×
RMS Vanilla-TGN	2.67×	3.00×	2.23×	2.05×

Table A.1 compares deviation ratios between Non-RMS and RMS Vanilla-TGN. The deviation ratio is calculated using the same std-normalized methodology as described in Section 6.2.6 (see Table 6.4): for each period, we compute the mean Euclidean distance of memory states from the normal period centroid, normalize by the standard deviation of normal period memory states, and take the ratio of anomaly period deviation to normal period deviation.

Both Non-RMS (2.73–3.29×) and RMS (2.05–3.00×) Vanilla-TGN models exhibit substantial memory contamination, with comparable deviation ratios across all feature groups. This confirms that contamination occurs regardless of training strategy.

A.5 Discussion and Conclusion

Key Implications

This appendix provides three key insights:

1. **GRU training fundamentally changes contamination progression dynamics**: Reconstruction behavior analysis (Section A.2) provides the most direct evidence that GRU parameter training affects how contamination progresses. When GRU parameters are trained (RMS), contamination progresses gradually as the adaptive GRU learns to smoothly incorporate new patterns. When GRU parameters are fixed (Non-RMS), contamination progresses abruptly as fixed parameters rapidly incorporate anomalous patterns. This difference in progression dynamics is consistently observed across all four feature groups (Latency, Memory, CPU, Network).

2. **Total contamination levels are similar despite different progression dynamics**: Quantitative evaluation through deviation ratios (Table A.1) confirms that both Non-RMS (2.73–3.29×) and RMS (2.05–3.00×) configurations exhibit substantial and comparable memory contamination. This validates that contamination occurs regardless

of training strategy, and at least within the observed experimental period, the key difference lies in how contamination progresses (abruptly vs. gradually) rather than in the total amount of contamination.

3. **Different progression dynamics are reflected in PCA patterns**: As noted in Section 6.2.5, RMS training enables memory states to gradually incorporate anomalous data rather than exhibiting abrupt distributional shifts. This appendix provides deeper analysis of this phenomenon through explained variance. Abrupt contamination progression (Non-RMS) creates simple change patterns through fixed GRU parameters, concentrating memory state variations in a few principal directions. Consequently, PC1+PC2 explain approximately 90% of variance, resulting in clear separation in 2D PCA. In contrast, gradual contamination progression (RMS) through adaptive GRU learning creates complex, high-dimensional memory state changes. Variations are distributed across many principal components, with PC1+PC2 explaining only approximately 50% of variance. The remaining approximately 50% of variations exist in higher-dimensional components (PC3 and beyond). Therefore, differences between normal and anomaly periods do not clearly appear in 2D projection (PC1-PC2 plane), causing them to overlap. This overlap does not indicate absence of contamination, but rather means that contamination-induced changes are high-dimensional and complex, and cannot be fully captured by 2D projection.

Conclusion

This appendix confirmed through multiple complementary analyses that GRU parameter training affects how contamination progresses (abruptly vs. gradually), not whether it occurs. This progression difference—rapid incorporation with fixed parameters versus adaptive incorporation through learning—is consistently observed across all four feature groups, demonstrating the fundamental impact of GRU training on memory contamination dynamics.

References

[1] Amossys. SPOT: Streaming peaks-over-threshold. https://github.com/Amossys-team/SPOT, 2017. Accessed: 2025-01-29.

[2] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly Media, Sebastopol, CA, 2016.

[3] Siddharth Bhatia, Arjit Jain, Shivin Srivastava, Kenji Kawaguchi, and Bryan Hooi. MemStream: Memory-based streaming anomaly detection. In *Proceedings of the ACM Web Conference (WWW)*, pages 610–621, 2022.

[4] Albert Bifet and Ricard Gavaldà. Learning from time-changing data with adaptive windowing. *Proceedings of the 2007 SIAM International Conference on Data Mining*, pages 443–448, 2007.

[5] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[6] Yingnong Dang, Qingwei Lin, and Peng Huang. AIOps: Real-world challenges and research innovations. In *IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 4–5. IEEE, 2019.

[7] Jeffrey Dean and Luiz André Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013.

[8] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. Microservices: Yesterday, today, and tomorrow. In *Present and Ulterior Software Engineering*, pages 195–216. Springer, 2017.

[9] Yifan Duan, Xinyu Yang, Dong Li, Haojia Zhang, and Yiming Zhang. FAMOS: Fault diagnosis for microservice systems through effective multi-modal data fusion. In *Proceedings of the 47th International Conference on Software Engineering (ICSE)*. IEEE/ACM, 2025.

[10] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[11] Google Cloud Platform. Online boutique: Cloud-native microservices demo application. https://github.com/GoogleCloudPlatform/microservices-demo, 2024. Accessed: 2025-01-24.

[12] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[14] Dongjin Lee, Junghyun Lee, and Kijung Shin. Spear and shield: Adversarial attacks and defense methods for model-based link prediction on continuous-time dynamic graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 13566–13574, 2024.

[15] Dewei Liu, Chuan He, Xin Peng, Fan Lin, Chenxi Zhang, Shengfang Gong, Zai Li, Jiayu Ou, and Zheshun Wu. MicroHECL: High-efficient root cause localization in large-scale microservice systems. In *Proceedings of the 43rd International Conference on Software Engineering (ICSE)*, pages 338–349. IEEE/ACM, 2021.

[16] Aleksei Liuliakov, Alexander Schulz, Luca Hermes, and Barbara Hammer. One-class intrusion detection with dynamic graphs. In *International Conference on Artificial Neural Networks (ICANN)*, pages 537–549. Springer, 2023.

[17] Robert H. Lupton, James E. Gunn, and Alexander S. Szalay. A modified magnitude system that produces well-behaved magnitudes, colors, and errors even for low signal-to-noise ratio measurements. *The Astronomical Journal*, 118(3):1406–1410, 1999.

[18] Luan Pham, Huong Ha, and Hongyu Zhang. BARO: Robust root cause analysis for microservices via multivariate Bayesian online change point detection. *Proceedings of the ACM on Software Engineering*, 1(FSE):Article 98, 2024.

[19] Luan Pham, Hongyu Zhang, Huong Ha, Flora Salim, and Xiuzhen Zhang. RCAEval: A benchmark for root cause analysis of microservice systems with telemetry data. In *Companion Proceedings of the ACM Web Conference (WWW)*, 2025.

[20] Twitter Research. TGN: Temporal graph networks. https://github.com/twitter-research/tgn, 2020. Accessed: 2025-01-29.

[21] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. In *ICML Workshop on Graph Representation Learning*, 2020.

[22] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjing Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1548–1554, 2021.

[23] Alban Siffer, Pierre-Alain Fouque, Alexandre Termier, and Christine Largouet. Anomaly detection in streams with extreme value theory. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1067–1075. ACM, 2017.

[24] John W. Tukey. *Exploratory Data Analysis*. Addison-Wesley, Reading, MA, 1977.

[25] Li Wu, Johan Tordsson, Erik Elmroth, and Odej Kao. MicroRCA: Root cause localization of performance issues in microservices. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 1–9. IEEE, 2020.

[26] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2020.

[27] Guangba Yu, Pengfei Chen, Yufeng Li, Hongyang Chen, Xiaoyun Li, and Zibin Zheng. Nezha: Interpretable fine-grained root causes analysis for microservices on multi-modal observability data. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 553–565. ACM, 2023.

[28] Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019.

[29] Chenxi Zhang, Xin Peng, Chaofeng Sha, Ke Zhang, Zhenqing Fu, Xiya Wu, Qingwei Lin, and Dongmei Zhang. DeepTraLog: Trace-log combined microservice anomaly detection through graph-based deep learning. In *Proceedings of the 44th International Conference on Software Engineering (ICSE)*, pages 623–634. ACM, 2022.

[30] Zhiming Zhang, Zhenhe Li, Zhiwei Wu, and Jianhua Li. CloudAnoAgent: Anomaly detection for cloud sites via LLM agent with neuro-symbolic mechanism. In *Proceedings of the 32nd ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 2024.

[31] Ming Zhou and Jie Lu. Continuous graph learning-based self-adaptation for multi-stream concept drift. *IEEE Transactions on Cybernetics*, 2025.

[32] Ming Zhou, Jie Lu, and Guangquan Zhang. Dynamic graph regularization for multi-stream concept drift self-adaptation. *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[33] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chenjie Ji, Wenhai Li, and Dan Ding. Enjoy your observability: an industrial survey of microservice tracing and analysis. *Empirical Software Engineering*, 27(1):1–40, 2022.

List of Presentations

Domestic Conference Presentations (Non-Refereed)

1. Tomoya Ueno, "Microservice Anomaly Detection and Root Cause Analysis Using Temporal Graph Networks with Adaptive Memory Protection," *The 43rd Symposium on Cryptography and Information Security (SCIS 2026)*, Hakodate, Japan, January 26–30, 2026.

Note: This presentation is based on the research described in this thesis.