# TopoGAT: Topology-Aware Unsupervised Root-Cause Localization in Distributed Training via Spatio-Temporal Graphs

**Anonymous authors**
Paper under double-blind review

## Abstract

Distributed large language model (LLM) training relies on parallelism for efficiency, yet synchronization allows a single faulty rank to slow an entire job, wasting substantial compute resources. Timely root-cause localization is therefore essential. Heterogeneous metric baselines make fixed thresholds difficult to apply, while learning-based alternatives often depend on scarce fault labels. Beyond these limitations, synchronization propagation causes root-cause and affected ranks to exhibit similar slowdowns, making anomaly severity alone insufficient for localization. We propose *TopoGAT* , an unsupervised, parallelism-aware spatio-temporal graph attention framework trained exclusively on normal data. Using collective communication events, GPU metrics, and network counters, *TopoGAT* learns normal spatio-temporal dependencies through a heterogeneous graph encoding parallel relationships, temporal continuity, and cluster-level context. A two-stage calibration suppresses group-wise common-mode noise and normalizes heterogeneous baselines, exposing local deviations informative for root-cause localization. Trained once on a single benchmark topology, *TopoGAT* achieves 94.6% offline and 88.9% online Hit@3 in zero-shot evaluation on 11 real-world datasets spanning three model families, ten parallelism configurations, and computation- and communication-side faults. It substantially outperforms state-of-the-art baselines while using approximately 0.16 million parameters and processing each window in milliseconds.

## 1 Introduction

Distributed training is the standard approach for training large language models (LLMs), combining tensor parallelism (TP), pipeline parallelism (PP), and data parallelism (DP) into a three-dimensional parallelism paradigm (Shoeybi et al., 2019; Rasley et al., 2020). At each iteration, collective communication (CCL) operations synchronize gradients and activations across ranks (Dai et al., 2026). These operations enable efficient scaling but require participating ranks to wait for the slowest member. A single faulty rank can therefore slow the entire job through synchronization propagation (Tang et al., 2020; Gandhi et al., 2024; Wu et al., 2025). Such slowdowns incur substantial costs: a 40-minute slowdown in a 128-machine training job can waste over $1,700 in GPU compute resources (Deng et al., 2025). Timely root-cause rank localization is thus essential for targeted remediation and reduced resource waste.

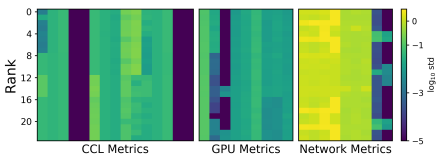



Figure 1: Under fault-free conditions, the fluctuation magnitude of the same metric varies across ranks.

Existing approaches face several limitations. Threshold-based methods (Pazho et al., 2023) struggle with substantial variation in normal baselines across ranks and metrics (Figure 1). Learning-based alternatives include supervised methods such as Minder (Deng et al., 2025) and unsupervised methods such as TSLoc (Zhou et al., 2026). However, two challenges remain: (1) **Dependence on scarce labels.** Root-cause annotation requires substantial engineering effort, limiting the availability of labeled data for supervised methods. Even otherwise unsupervised approaches may retain label dependence: TSLoc uses labeled data to calibrate metric-specific anomaly-score weights (Zhou et al., 2026).



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

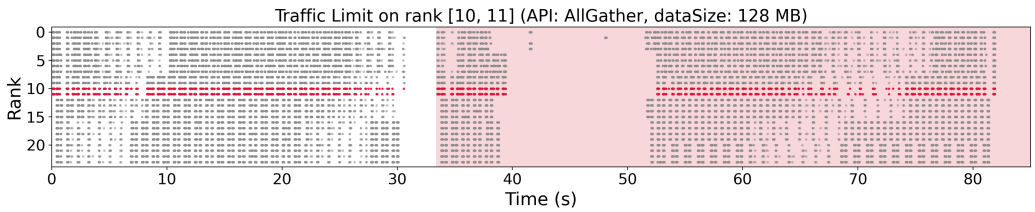



Figure 2: CCL event streams (AllGather, 128 MB) across 24 ranks in a real training job, comparing a normal iteration (left) with the first fault-affected iteration (shaded). A traffic-limiting fault on two ranks (red) extends the iteration duration to 2.6× its normal value. Near-lockstep slowdown within parallel groups obscures root-cause ranks, while irregular communication events on healthy ranks 0–3 further mask their signals.

(2) **Mismatch between observed anomalies and causal origins.** Synchronization propagates local fault effects through heterogeneous parallel topologies, obscuring the distinction between root-cause ranks and affected ranks. The most anomalous rank in raw telemetry may therefore be a victim rather than the root cause (Figure 2). Reliable localization must distinguish local fault signals from synchronization-induced shared slowdown.

Our approach draws on normal training dynamics: each rank exhibits spatial consistency with its parallel-group neighbors and temporal continuity with its own history. These regularities enable learning without fault labels, but deviations alone cannot distinguish root causes from affected ranks. We therefore model normal spatio-temporal dependencies and suppress slowdown components shared within parallel groups to expose informative local deviations. Building on this insight, we present *TopoGAT* , an unsupervised, parallelism-aware spatio-temporal graph attention framework for root-cause rank localization. *TopoGAT* first infers iteration durations from CCL events and detects anomalous segments using a robust CUSUM statistic (Page, 1954). It then constructs a spatio-temporal heterogeneous graph for each anomalous window. Ranks form nodes, TP/PP/DP relationships define typed spatial edges, directed temporal edges connect the same rank across consecutive iterations, and global nodes provide cluster-level context. A self-supervised graph attention network (GAT) predicts each rank's deviation from its neighbors. Two-stage calibration (Decommon + Baseline-scale) suppresses group-wise common-mode noise and normalizes heterogeneous baselines. Calibrated prediction errors are aggregated into window-level scores, and online and offline readouts produce Top-$K$ root-cause candidates. Trained exclusively on normal data, *TopoGAT* supports zero-shot generalization across workloads.

Our main contributions are as follows:

- We formalize root-cause rank localization under synchronization propagation using only normal training data, identifying group-wise common-mode slowdown and heterogeneous statistical baselines as key obstacles to root-cause identification.
- We propose *TopoGAT* , which encodes parallel topology through typed spatial edges, directed temporal edges, and global nodes. Its two-stage calibration suppresses common-mode noise and normalizes heterogeneous baselines, improving the root-cause signal-to-noise ratio (SNR) and supporting cross-task generalization.
- Trained once on normal data from a single benchmark topology, *TopoGAT* achieves 94.6% offline and 88.9% online Hit@3 in zero-shot evaluation across 11 diverse real-world datasets, substantially outperforming state-of-the-art baselines. It uses only 0.16M parameters, processes windows in milliseconds, and is validated on production traces.

## 2 Related Work

### 2.1 Distributed Training Fault Handling

Slow-node mitigation methods (e.g., EDiT (Cheng et al., 2025), AntDT (Xiao et al., 2024), and sequential gradient coding (Lee et al., 2017)) bypass stragglers to preserve throughput but do not identify their root causes (Rahman, 2025). In contrast, root-cause diagnosis methods aim to identify

faulty components explicitly: Minder (Deng et al., 2025) uses supervised LSTM-VAEs but depends on labeled faults, while TSLoc (Zhou et al., 2026) combines model scores with hand-crafted rules but require per-task tuning.

### 2.2 Anomaly Detection

Time-series methods (e.g., Anomaly Transformer (Xu et al., 2021)) operate per-node independently, ignoring structural dependencies (Bayani, 2026). Graph anomaly detection approaches (e.g., GDN (Deng & Hooi, 2021), MTAD-GAT (Zhao et al., 2020)) use graph structure as auxiliary context (Jiang et al., 2025). However, during synchronization, fault propagation is governed by the propagation topology, thereby violating the independence assumptions.

In terms of model architecture, spatio-temporal GNNs typically stack spatial and temporal layers sequentially (Jin et al., 2024; Sahili & Awad, 2023), while heterogeneous GATs handle multiple edge types without integrating time into graph structure (Wang et al., 2019). We unify spatial and temporal edges in a single heterogeneous graph for joint reasoning and parameter efficiency. In terms of learning paradigm, self-supervised anomaly detection uses reconstruction or prediction error as anomaly signals (Nguyen et al., 2024; Wang et al., 2022), but does not account for synchronization propagation: raw prediction errors rise across the entire group, masking the root cause. Our two-stage calibration addresses this gap.

To our knowledge, no prior work encodes parallelism topology as inductive bias for unsupervised root-cause localization under synchronization propagation.

## 3 Problem Formulation

Consider a distributed training job with $N$ ranks $\mathcal{R} = \{r_1, \ldots, r_N\}$ organized in standard 3D parallelism (TP/PP/DP). Each rank belongs to exactly one group of each parallelism type. For each rank $r$ at iteration $t$, we observe a $D$-dimensional telemetry vector $x(r,t) \in \mathbb{R}^D$ covering CCL events, GPU metrics, and network counters.

**Task:** Given topology group assignments and telemetry over a detection window, output an anomalous iteration segment $[t_s, t_e]$ and a ranked Top-K list of candidate root causes.

Collective communication barriers force all ranks in a parallelism group to wait for the slowest member (Hu et al., 2025; Dalmia et al., 2022). For any group $g$, the iteration end time is:

$$T_{\text{end}}(g,t) = \max_{r \in g} \left[ T_{\text{start}}(r,t) + T_{\text{comp}}(r,t) + T_{\text{comm}}(r,t) \right] \tag{1}$$

A fault on rank $r$ with extra delay $\Delta$ propagates to all group members, yielding two consequences:

- **Group-wide common-mode slowdown**: Observed iteration duration is identical across the affected group, so absolute slowdown cannot identify the root cause.
- **Victim anomaly dominance**: Waiting ranks may exhibit more volatile metric fluctuations than the faulty rank itself. The most visibly anomalous rank is often a victim, not the root cause.

This creates a fundamental observation-causality mismatch—the core challenge we address.

## 4 TopoGAT: Unsupervised Spatio-Temporal Graph Attention

Figure 3 illustrates the overall architecture of *TopoGAT* . Both normal and anomalous sliding windows are processed through the spatio-temporal graph. The prediction errors of normal windows dynamically maintain the per-(rank, metric) baseline error statistics, while anomalous windows detected by the scanner activate the root-cause localization workflow.

### 4.1 Data Preprocessing

Monitoring training throughput over fixed intervals (e.g., 30s or 5min) has low SNR because different workloads fluctuate differently, making slowdown detection unreliable. To achieve reliable temporal alignment, we infer iteration boundaries from CCL event streams as stable anchors. CCL

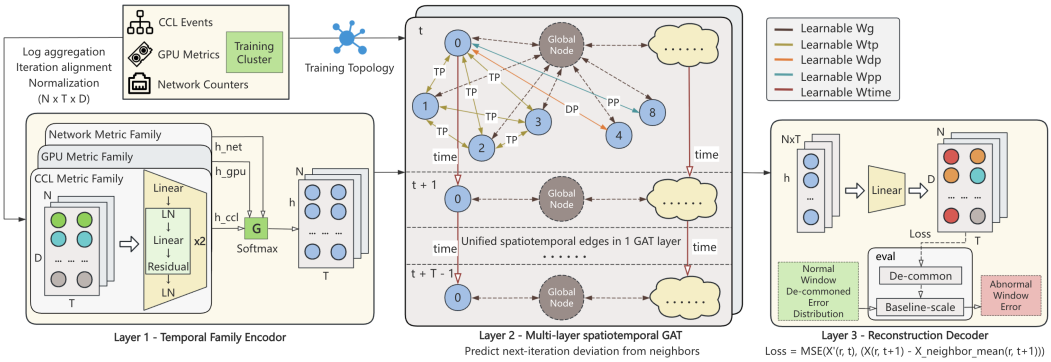



Figure 3: Overall pipeline of *TopoGAT* . The first stage infers iteration durations from CCL events and detects anomalous segments via CUSUM and MAD (the detection results are provided in Appendix C). All sliding windows are fed into the spatio-temporal graph: normal windows estimate the baseline error distribution for calibration, while anomalous windows trigger root-cause localization and output ranked Top-K candidate ranks.

events contain only `timestamp`, `api`, `data_size`, and `duration`, with no explicit iteration indices. Over the first 8 ranks, we use multi-rank voting to identify the most stable, widest-ranging, and most frequent (`api`, `data_size`) pair as the periodic anchor: each rank independently detects candidate anchors, and the same pair must receive at least 50% of the votes. The final iteration period is the median of per-rank periods, and boundaries are then aligned across all ranks. This approach is robust to missing data and clock drift, and assumes no fixed anchor type across training tasks. Full algorithm details and anchor detection results are provided in Appendix B.

We employ two normalization strategies to accommodate different metric distributions (Khanmohammadi & Azmi, 2024): (1) Min-Max normalization to CCL and GPU metrics, which have well-defined physical ranges: $x_{\text{norm}} = (\text{clip}(x, s_{\min}, s_{\max}) - s_{\min})/(s_{\max} - s_{\min} + \epsilon)$; (2) Robust normalization to Network metrics, which exhibit heavy-tailed distributions: $x_{\text{norm}} = (x - \text{median})/(\text{p90} - \text{median})$, with statistics estimated only on normal iterations.

Finally, we construct a heterogeneous spatio-temporal graph with three categories of edges:

- **Spatial parallelism edges**: Full connectivity within each TP, PP, and DP group (no self-loops) to model synchronization constraints. Edges are replicated at every iteration within the window.
- **Global context edges**: One global node per iteration, whose feature is the mean of rank features at that iteration. This node connects bidirectionally to all ranks, providing cluster-level context.
- **Directed temporal edges**: Directed edges $(r, t) \to (r, t+1)$ connecting consecutive iterations of the same rank, modeling temporal continuity and causal dependencies across iterations.

The entire graph consists of $(N + 1) \times T$ nodes (with $T$ Global Nodes) and 5 edge types.

### 4.2 Temporal Encoding

Telemetry metrics form three semantically distinct families: GPU metrics reflect local hardware state, CCL metrics reflect collective synchronization, and Network metrics lie between them. Mixing them in one encoder causes mutual interference and dilutes fault signals when anomalies only occur in one family. Therefore, we encode each family independently and fuse adaptively. Each family is processed by an independent temporal encoder with shared structure:

$$\text{Linear}(D_f \to d) \to \text{GELU} \to \text{Dropout} \to 2 \times \text{ResBlock}(\text{LN} \to \text{Linear} \to \text{Dropout}) \to \text{LN}$$

where $D_f$ is the dimensionality of family $f$ and $d$ is the model dimension.

We fuse the three family features using a lightweight scalar gating mechanism. For each family $f$, we compute the per-node L2 norm of its feature tensor $e_f = \|h_f\|_2$, stack them into a vector of shape $(B, N, T, F)$, and pass through a linear layer followed by softmax to obtain per-family

weights $w_f$. The fused node feature is:

$$h = \sum_{f=1}^{F} w_f \cdot h_f \tag{2}$$

With $F = 3$ families, the gating mechanism introduces only 12 learnable parameters.

### 4.3 Spatio-Temporal Heterogeneous GAT

Fused node features feed a heterogeneous GAT that jointly models spatial and temporal dependencies in one unified architecture. We stack 2 GAT layers, each followed by ResBlock and LN.

For each of the five edge types (TP, PP, DP, Global, Time), we apply an independent linear projection followed by standard GAT attention scoring (Veličković et al., 2017):

$$\alpha_{ij}^{k} = \frac{\exp\left(\text{LeakyReLU}\left(a_k^\top \left[W_k h_i \parallel W_k h_j\right]\right)\right)}{\sum_{l \in \mathcal{N}_k(i)} \exp\left(\text{LeakyReLU}\left(a_k^\top \left[W_k h_i \parallel W_k h_l\right]\right)\right)} \tag{3}$$

where $W_k$ and $a_k$ are the projection matrix and attention vector for edge type $k$, $\mathcal{N}_k(i)$ denotes neighbors of node $i$ via edge type $k$, and $\parallel$ denotes concatenation. Aggregated features from each edge type are combined via learnable scalar weights:

$$h_i' = \sum_{k=1}^{K} w_k \cdot \sum_{j \in \mathcal{N}_k(i)} \alpha_{ij}^{k} \cdot W_k h_j \tag{4}$$

Modeling inter-iteration dependencies as directed temporal edges instead of separate temporal and spatial layers lets spatial and temporal information interact within each attention layer for joint patterns, while encoding time as edges avoids duplicating graph layers and keeps model small.

Global nodes provide a cluster-level reference point, allowing each rank to gauge its state relative to the entire job rather than only its local parallelism group. This prevents localization bias when faults affect entire groups and helps calibrate across groups with different baseline workloads.

### 4.4 Spatio-Temporal Prediction Objective

We train the model with a self-supervised prediction objective designed to exploit both spatial synchronization and temporal continuity (Zhang et al., 2024). To explicitly couple each rank's prediction target with its spatial neighbors, we predict its *deviation from the neighbor mean*:

$$x_d(r,t) = x(r,t) - \text{neighbor_mean}(x(r,t)) \tag{5}$$

where neighbor_mean averages over all spatial neighbors of rank $r$, acting as a simple, parameter-free smoother. The model output $\hat{x}(r,t)$ is trained to predict $x_d(r,t+1)$. The training loss is MSE over all $T-1$ prediction positions:

$$\mathcal{L} = \frac{1}{(T-1)\cdot N \cdot D} \sum_{t=1}^{T-1} \sum_{r=1}^{N} \sum_{m=1}^{D} \left(\hat{x}(r,t,m) - x_d(r,t+1,m)\right)^2 \tag{6}$$

This objective enforces both spatial and temporal constraints: predicting a rank's deviation from its neighbors encourages learning within-group synchronization patterns rather than copying the input via ResBlock, while one-step-ahead prediction captures normal temporal evolution, so abrupt fault-induced relative deviations yield large prediction errors.

### 4.5 Error Calibration

Raw prediction errors alone cannot reliably identify root causes due to common-mode propagation and heterogeneous baseline noise. We first average raw per-(rank, metric) errors over time into an $(N, D)$ error field, then introduce a two-stage calibration to recover the root-cause signal.

#### 4.5.1 De-common: Suppresses Common-Mode Noise

Delay propagation causes prediction errors to rise across all members of an affected group. Absolute error thus cannot distinguish the root cause from victims waiting at the synchronization barrier. For each parallelism group type $g \in \{\text{TP}, \text{PP}, \text{DP}\}$, we subtract the group-wise median error from each rank's error (Ranks with no peers use the global median):

$$e_g(r, m) = \text{err}(r, m) - \underset{r' \in g(r)}{\text{median}}\ \text{err}(r', m) \tag{7}$$

where $g(r)$ denotes the group containing rank $r$. $e_g(r, m)$ measures a rank's deviation from group consensus. We use median rather than mean for robustness against multiple victim ranks.

#### 4.5.2 Baseline-scale: Normalizes Heterogeneous Fluctuations

Normal errors vary across (rank, metric) pairs due to differences in hardware, workload, and topology. Directly comparing De-commoned errors across ranks produces false positives from inherently noisy ranks. We maintain a running estimate of the 99th percentile of De-commoned errors from normal windows, denoted $\sigma_g(r, m)$ (*it is not part of model training*). The calibrated error is:

$$x_g(r, m) = \frac{e_g(r, m)}{\sigma_g(r, m) + \epsilon} \tag{8}$$

Reversing the calibration order reduces offline Hit@3 by 3.5 points on average while leaving online comparable, so we adopt De-common before Baseline-scale (Appendix I.5).

### 4.6 Scoring Pipeline

First, we apply per-metric z-scoring across ranks (**Cluster-z**) to make scores comparable across metrics, then pool over the metric dimension by taking the maximum value and average over the three parallelism views (TP, PP, DP) to obtain a per-rank window score:

$$s_{\text{win}}(r) = \text{mean}_{g \in \{\text{TP,PP,DP}\}} \left( \max_m\ x_g(r, m)^2 \right) \tag{9}$$

#### 4.6.1 Online Readout: Real-Time Continuous Localization

For real-time monitoring, we maintain an EMA of window scores (Ross et al., 2012), gated by detected anomaly segments (EMA state reset upon entering a new anomaly segment):

$$s_{\text{online}}(r, t) = \gamma \cdot s_{\text{online}}(r, t-1) + (1 - \gamma) \cdot s_{\text{win}}(r, t) \tag{10}$$

#### 4.6.2 Offline Readout: Event-Level Attribution

For post-hoc diagnosis with access to the full anomaly segment, we average window scores over the entire event and subtract a normal baseline:

$$s_{\text{offline}}(r) = \frac{1}{|\mathcal{T}_a|} \sum_{t \in \mathcal{T}_a} \left( s_{\text{win}}(r, t) - \underset{t \in \mathcal{T}_n}{\text{median}}\ s_{\text{win}}(r, t) \right) \tag{11}$$

where $\mathcal{T}_a$ is the set of anomalous iterations and $\mathcal{T}_n$ is a set of preceding normal iterations.

## 5 Experiments

We conduct a comprehensive set of experiments to systematically evaluate *TopoGAT* .

### 5.1 Experimental Setup

**Training:** The model is trained *once* on a single benchmark topology (64 ranks, TP=8, PP=4, DP=2) with only normal data. (All hyperparameters are provided in Appendix D).

**Datasets:** We evaluate on 11 real-world distributed training datasets collected from production clusters. The datasets span 3 model families (LLaMA2-70B, Qwen2.5-7B, and Qwen3-8B), 2 training

tasks (pre-training and fine-tuning), 10 parallelism configurations with 16 to 128 ranks, 12 fault types (both communication and computation faults), and 2 injection methods (hardware and software faults). Detailed fault injection specifications are provided in Appendix E.

**Baselines:** We compare *TopoGAT* against 8 representative backbones: (1) Temporal-free: MLP; (2) Generic time-series: GRU (Cho et al., 2014), LSTM (Hochreiter & Schmidhuber, 1997), 1D-CNN (Wang et al., 2017), STConv (Yu et al., 2017) and Anomaly Transformer (Xu et al., 2021); (3) Distributed training diagnosis: TSLoc (Zhou et al., 2026), and Minder LSTM-VAE (Deng et al., 2025). All baselines use the same data, scoring chain, and evaluation protocol.

**Evaluation Metrics:** We report standard ranking evaluation metrics: *Hit@K* (whether the true root cause appears in the top-K candidates), *Recall@K*, *MRR@K* (Mean Reciprocal Rank), and *NDCG@K* (Normalized Discounted Cumulative Gain) (Järvelin & Kekäläinen, 2002).

### 5.2 Main Results

Since the two-stage De-common and Baseline-scale calibration is one of our contributions, applying it only to *TopoGAT* would confound architecture comparison with calibration. We therefore evaluate all methods under two unified protocols: the **bare protocol** applies no calibration, isolating each backbone's localization capability; the **full protocol** applies the same calibration to every backbone, evaluating the complete pipeline under equal treatment. All models are trained only on the benchmark dataset (D1, Table 2) and evaluated across all 11 datasets. Table 1 reports Hit@3 on the benchmark and averaged over all datasets; per-dataset breakdowns are provided in Appendix F.

Table 1: Comparison with baseline methods under both protocols (Hit@3, *online / offline*). *Bench*: the benchmark dataset (D1) on which all models are trained; *Avg*: average over all 11 evaluation datasets.

| | Bare protocol | | Full protocol | |
|---|---|---|---|---|
| **Backbone** | **Bench** | **Avg** | **Bench** | **Avg** |
| TopoGAT (ours) | 0.745 / **0.867** | **0.781 / 0.835** | 0.804 / 0.900 | **0.889 / 0.946** |
| MLP | 0.773 / **0.867** | 0.760 / 0.812 | 0.810 / 0.867 | 0.833 / 0.860 |
| GRU | 0.745 / 0.800 | 0.757 / 0.795 | 0.779 / 0.867 | 0.827 / 0.856 |
| LSTM | **0.801** / 0.800 | 0.730 / 0.794 | 0.804 / 0.867 | 0.841 / 0.878 |
| 1D-CNN | 0.776 / 0.833 | 0.763 / 0.807 | 0.838 / 0.900 | 0.835 / 0.868 |
| STConv | 0.791 / 0.833 | 0.741 / 0.803 | 0.838 / 0.900 | 0.841 / 0.905 |
| Anomaly Transformer | 0.782 / 0.833 | 0.746 / 0.832 | 0.826 / 0.833 | 0.846 / 0.899 |
| TSLoc | 0.670 / 0.800 | 0.573 / 0.673 | 0.763 / 0.900 | 0.688 / 0.769 |
| VAE (Minder) | 0.723 / **0.867** | 0.744 / 0.794 | **0.875 / 0.933** | 0.862 / 0.884 |

Three key findings: (1) On the benchmark dataset where all models are trained, several baselines are competitive and some even surpass *TopoGAT* under the full protocol: VAE reaches 0.933 offline Hit@3 and 0.875 online, above *TopoGAT* 's 0.900 and 0.804. However, its zero-shot averages collapse—the strongest baseline average are VAE at 0.862 online and STConv at 0.905 offline, and TSLoc falls to 0.688 / 0.769—whereas *TopoGAT* attains 0.889 / 0.946. This shows that the bottleneck is not fitting the training topology but transferring to unseen ones, which the topology inductive bias of *TopoGAT* directly addresses. (2) Even without calibration, the *TopoGAT* backbone attains the best bare-protocol average (0.781 / 0.835), confirming that the spatio-temporal heterogeneous graph provides a localization advantage independent of calibration. (3) Under the full protocol, all backbones significantly improve over their bare-protocol scores, confirming that the two-stage calibration itself contributes. Meanwhile, *TopoGAT* still achieves the best average zero-shot Hit@3, outperforming the strongest baseline by 2.7 / 4.1 points, indicating that its architecture provides a complementary contribution in cross-task transfer and common-mode noise suppression.

### 5.3 Top-K Localization Performance

We evaluate ranking quality on the benchmark dataset across $K \in \{1, 2, 3, 5, 10\}$ (Figure 4). *TopoGAT* achieves an offline Hit@3 of 90.0% and an online Hit@3 of 80.4%: that means, given only three candidate ranks to inspect, operators find the faulty GPU in 9 out of 10 cases post-hoc, and 8 out of 10 under real-time monitoring. Offline readouts consistently outperform online ones because event-level averaging suppresses single-window noise, and offline Hit@10 reaches 93.3%. MRR@K and NDCG@K exhibit the same saturating trend, confirming that the ranked list places the true root cause near the top rather than merely within it.

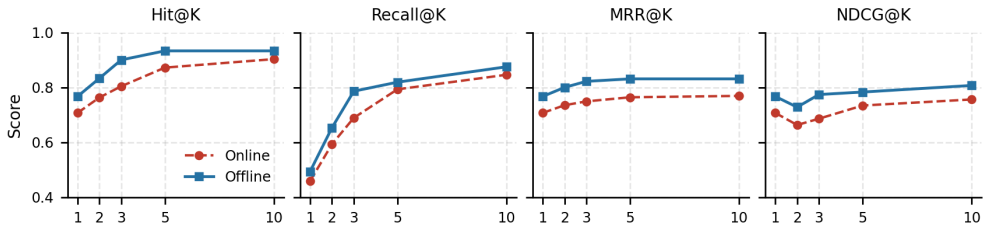


Figure 4: Top-K localization performance on the benchmark dataset.

### 5.4 Generalization

The most critical property of *TopoGAT* is transferring across unseen training tasks and topologies. Table 2 summarizes the 11 evaluation datasets. *TopoGAT* is trained only on D1 (LLaMA2-70B pre-training, 64 ranks) and evaluated zero-shot on D2–D11, which differ from the training topology in model family (Qwen2.5-7B, Qwen3-8B), training stage (supervised fine-tuning), scale (16–128 ranks), parallelism structure (9 additional TP/PP/DP configurations), and fault-injection method (framework-side Megatron injection).

Table 2: Generalization across 11 evaluation datasets. *Injection*: *Hardware* = physical anomalies on GPU/NIC; *Megatron* = framework-side software slowdowns via a modified Megatron-LM framework (Shoeybi et al., 2019; Yu et al., 2026).

| ID | Model | Stage | Ranks | TP/PP/DP | Injection | Fault count | Hit@3 |
|---|---|---|---|---|---|---|---|
| D1 (dataset for training) | LLaMA2-70B | Pre-training | 64 | 8/4/2 | Hardware | 30 | 0.804 / 0.900 |
| D2 | LLaMA2-70B | Pre-training | 64 | 8/4/2 | Megatron | 86 | 0.946 / 0.981 |
| D3 | LLaMA2-70B | Pre-training | 32 | 4/8/1 | Hardware | 20 | 0.764 / 0.895 |
| D4 | LLaMA2-70B | Pre-training | 64 | 4/8/2 | Hardware | 20 | 0.900 / 0.900 |
| D5 | LLaMA2-70B | Pre-training | 32 | 8/4/1 | Hardware | 19 | 0.882 / 0.944 |
| D6 | LLaMA2-70B | Pre-training | 64 | 8/8/1 | Hardware | 20 | 0.864 / 0.950 |
| D7 | LLaMA2-70B | Pre-training | 128 | 8/8/2 | Hardware | 30 | 0.884 / 0.933 |
| D8 | Qwen2.5-7B | Pre-training | 16 | 4/2/2 | Hardware | 10 | 0.992 / 1.000 |
| D9 | Qwen2.5-7B | Pre-training | 32 | 4/2/4 | Hardware | 20 | 0.927 / 1.000 |
| D10 | Qwen3-8B | SFT | 24 | 4/2/3 | Hardware | 12 | 0.878 / 0.900 |
| D11 | Qwen3-8B | SFT | 64 | 4/4/4 | Hardware | 16 | 0.943 / 1.000 |
| **Average** | | | | | | **0.889 / 0.946** | |

Across all datasets, *TopoGAT* achieves an average offline Hit@3 of 94.6% and online Hit@3 of 88.9%, and reaching 1.00 on 3 unseen workloads. These results demonstrate that *TopoGAT* learns general principles of topological relationships, rather than overfitting to benchmark topology.

### 5.5 Ablation Studies

To verify the contribution of each core component, we conduct ablation studies by removing one component at a time and evaluating across all datasets. Table 3 summarizes the results.

Table 3: Ablation study results.

| Variant | Hit@3 | Δ |
|---|---|---|
| Default (Full model) | 0.889 / 0.946 | – / – |
| w/o Cluster-z | 0.800 / 0.818 | -0.089 / -0.127 |
| w/o De-common | 0.832 / 0.882 | -0.058 / -0.063 |
| w/o Baseline-scale | 0.825 / 0.915 | -0.064 / -0.031 |
| w/o Global Node | 0.835 / 0.870 | -0.054 / -0.076 |
| w/o Time Edge | 0.858 / 0.918 | -0.032 / -0.028 |
| w/o Family & Gate | 0.846 / 0.901 | -0.043 / -0.044 |
| w/o Gate | 0.842 / 0.897 | -0.047 / -0.048 |
| w/o Edge Typing<br># Anonymous edges (no TP/PP/DP typing) | 0.862 / 0.906 | -0.027 / -0.040 |
| w/o Spatio-Target<br># Loss = MSE(x'(r,t) - x(r,t+1)) | 0.870 / 0.918 | -0.019 / -0.028 |

Several key observations are: (1) Cluster-z is the single most critical component. Removing it reduces offline Hit@3 by 12.7%, because Cluster-z enables error scores to be comparable across metrics for unified ranking; (2) the Global Node and calibration are the next most important, costing 7.6% (Global Node), 6.3% (De-common), and 3.1% (Baseline-scale) offline, respectively, confirming that group-consensus calibration and cluster-level context jointly carry the root-cause signal; (3) family-wise encoding and gating contribute 4.4%, validating the design of encoder; (4) typed edge semantics and time edges add consistent gains of 4.0% and 2.8% offline, respectively, confirming the design of graph edges; (5) the neighbor-relative prediction objective yields a small but consistent 2.8% improvement.

### 5.6 Error Field Visualization

To illustrate the effect of *TopoGAT* backbone and the two-stage calibration, we visualize the raw and calibrated error field in Figure 5. The raw error field already shows a clear and strong anomaly signal, but also contains noise signals (e.g., rank1 under GPU underclocking). Calibration removes such noise, making the root-cause slow rank stand out in the final score. More detailed visualizations (three separate views, with De-common and Baseline-scale separated) are provided in Appendix H.

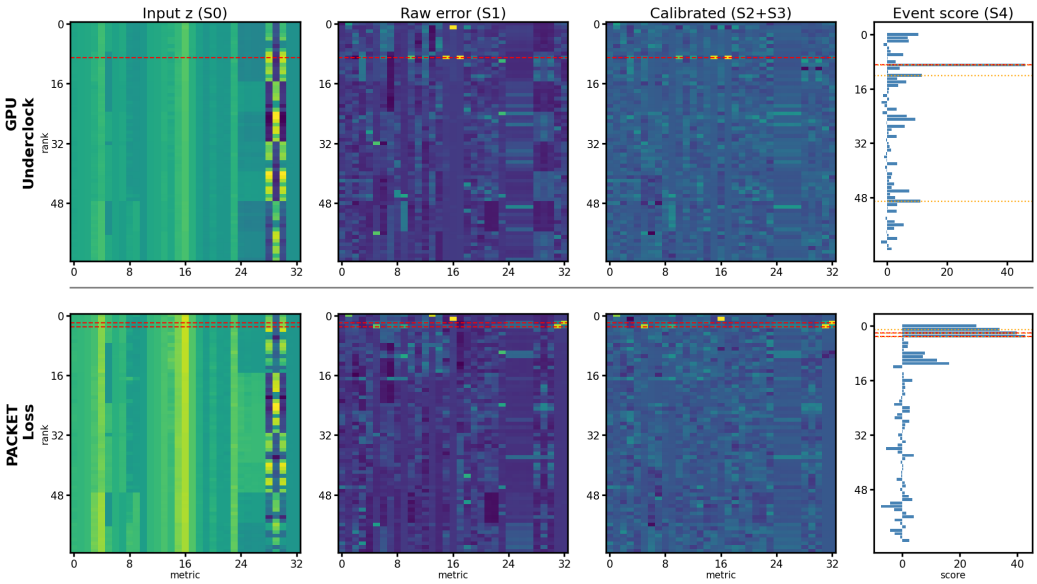



Figure 5: From left to right: input z-score, raw error, calibrated error, and final score. Brighter color indicates stronger signal. Red lines mark root-cause ranks; orange lines mark predicted Top-3 ranks.

### 5.7 Metric Family Robustness

To validate family-wise encoding and gated fusion, we ablate metric families during training and evaluation (Table 4). Removing GPU reduces gpu_underclock from 1.00 to 0.22; removing Network reduces packet_loss from 1.00 to 0.22 and traffic_limit from 0.87 to 0.22. CCL is redundant alone but critical in combination: removing both CCL and Network pushes communication faults below 0.10, and removing any two families is nearly catastrophic. The framework-side fault results are provided in Appendix G;

Table 4: Offline Hit@3 when a metric family is removed during both training and evaluation.

| **Fault type** | **full** | **−GPU** | **−CCL** | **−Net** | **−GPU-CCL** | **−GPU-Net** | **−CCL-Net** |
|---|---|---|---|---|---|---|---|
| gpu_stress | 0.78 | 0.22 | 0.17 | 0.31 | 0.06 | 0.25 | 0.30 |
| gpu_underclock | 1.00 | 0.22 | 0.50 | 0.50 | 0.09 | 0.14 | 0.50 |
| packet_loss | 1.00 | 0.50 | 0.50 | 0.22 | 0.50 | 0.27 | 0.09 |
| traffic_limit | 0.87 | 0.45 | 0.41 | 0.22 | 0.42 | 0.21 | 0.08 |
| mixed | 0.99 | 0.49 | 0.50 | 0.41 | 0.47 | 0.27 | 0.37 |

## 6 Conclusion

In this paper, we present *TopoGAT* , an unsupervised spatio-temporal graph framework for root-cause rank localization under synchronization propagation. It encodes TP/PP/DP topology as typed spatial edges and inter-iteration dependencies as directed temporal edges in a heterogeneous graph attention architecture, paired with a two-stage De-common and Baseline-scale calibration to suppress common-mode noise and normalize heterogeneous baselines. Trained once on a benchmark topology, *TopoGAT* generalizes zero-shot to 11 real-world datasets spanning diverse model families, parallelism configurations, and fault types, achieving state-of-the-art accuracy with only 0.16M parameters and millisecond inference. This shows that embedding synchronization structure as inductive bias enables robust, practical unsupervised diagnosis.

## REFERENCES


Samir Vinayak Bayani. Anomaly detection in large-scale distributed systems using graph-based deep learning. In *2026 2nd International Conference on Artificial Intelligence Systems (AIS)*, pp. 135–140. IEEE, 2026.

Jialiang Cheng, Ning Gao, Yun Yue, Zhiling Ye, Jiadi Jiang, and Jian Sha. Edit: A local-sgd-based efficient distributed training method for large language models. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 65944–65965, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/a4f419c398382295e1e350d56ecb65f2-Paper-Conference.pdf.

Kyunghyun Cho, Bart Van Merriënboer, Çağlar Gulçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1724–1734, 2014.

Liuyao Dai, Adam Weingram, Weicong Chen, and Xiaoyi Lu. Cclinsight: Unveiling insights in gpu collective communication libraries via primitive-centric analysis. In *Proceedings of the 2026 IEEE/ACM 48th International Conference on Software Engineering*, pp. 1674–1686, 2026.

Preyesh Dalmia, Rohan Mahapatra, Jeremy Intan, Dan Negrut, and Matthew D Sinclair. Improving the scalability of gpu synchronization primitives. *IEEE Transactions on Parallel and Distributed Systems*, 34(1):275–290, 2022.

Ailin Deng and Bryan Hooi. Graph neural network-based anomaly detection in multivariate time series. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 4027–4035, 2021.

Yangtao Deng, Xiang Shi, Zhuo Jiang, Xingjian Zhang, Lei Zhang, Zhang Zhang, Bo Li, Zuquan Song, Hang Zhu, Gaohong Liu, et al. Minder: Faulty machine detection for large-scale distributed model training. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pp. 505–521, 2025.

Swapnil Gandhi, Mark Zhao, Athinagoras Skiadopoulos, and Christos Kozyrakis. Recycle: Resilient training of large dnns using pipeline adaptation. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, pp. 211–228, 2024.

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

Zhiyi Hu, Siyuan Shen, Tommaso Bonato, Sylvain Jeaugey, Cedell Alexander, Eric Spada, James Dinan, Jeff Hammond, and Torsten Hoefler. Demystifying nccl: An in-depth analysis of gpu communication protocols and algorithms. In *2025 IEEE Symposium on High-Performance Interconnects (HOTI)*, pp. 48–59. IEEE, 2025.

Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002.

Laura Jiang, Reza Ryan, Qian Li, and Nasim Ferdosian. A survey of heterogeneous graph neural networks for cybersecurity anomaly detection. *Journal of Computer Security*, pp. 0926227X261461764, 2025.

Guangyin Jin, Yuxuan Liang, Yuchen Fang, Zezhi Shao, Jincai Huang, Junbo Zhang, and Yu Zheng. Spatio-temporal graph neural networks for predictive learning in urban computing: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 36(10):5388–5408, 2024. doi: 10.1109/TKDE.2023.3333824.

Fatemeh Khanmohammadi and Reza Azmi. Time-series anomaly detection in automated vehicles using d-cnn-lstm autoencoder. *IEEE Transactions on Intelligent Transportation Systems*, 25(8): 9296–9307, 2024.

Kangwook Lee, Maximilian Lam, Ramtin Pedarsani, Dimitris Papailiopoulos, and Kannan Ramchandran. Speeding up distributed machine learning using codes. *IEEE Transactions on Information Theory*, 64(3):1514–1529, 2017.

Huy Hoang Nguyen, Cuong Nhat Nguyen, Xuan Tung Dao, Quoc Trung Duong, Dzung Pham Thi Kim, and Minh-Tan Pham. Variational autoencoder for anomaly detection: A comparative study. *arXiv preprint arXiv:2408.13561*, 2024.

E. S. Page. Continuous inspection schemes. *Biometrika*, 41(1/2):100–115, 1954. ISSN 00063444. URL http://www.jstor.org/stable/2333009.

Armin Danesh Pazho, Ghazal Alinezhad Noghre, Arnab A Purkayastha, Jagannadh Vempati, Otto Martin, and Hamed Tabkhi. A survey of graph-based deep learning for anomaly detection in distributed systems. *IEEE Transactions on Knowledge and Data Engineering*, 36(1):1–20, 2023.

Ratun Rahman. Federated learning: A survey on privacy-preserving collaborative intelligence. *arXiv preprint arXiv:2504.17703*, 2025.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 3505–3506, 2020.

Gordon J Ross, Niall M Adams, Dimitris K Tasoulis, and David J Hand. Exponentially weighted moving average charts for detecting concept drift. *Pattern recognition letters*, 33(2):191–198, 2012.

Zahraa Al Sahili and Mariette Awad. Spatio-temporal graph neural networks: A survey, 2023. URL https://arxiv.org/abs/2301.10569.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Zhenheng Tang, Shaohuai Shi, Wei Wang, Bo Li, and Xiaowen Chu. Communication-efficient distributed deep learning: A comprehensive survey. *arXiv preprint arXiv:2003.06307*, 2020.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Hongchao Wang, Weiting Zhang, Dong Yang, and Yuhong Xiang. Deep-learning-enabled predictive maintenance in industrial internet of things: methods, applications, and challenges. *IEEE systems journal*, 17(2):2602–2615, 2022.

Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *The world wide web conference*, pp. 2022–2032, 2019.

Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International joint conference on neural networks (IJCNN)*, pp. 1578–1585. IEEE, 2017.

Tianyuan Wu, Lunxi Cao, Hanfeng Lu, Xiaoxiao Jiang, Yinghao Yu, Siran Yang, Guodong Yang, Jiamang Wang, Lin Qu, Liping Zhang, et al. Adaptra: Straggler-resilient hybrid-parallel training with pipeline adaptation. *arXiv preprint arXiv:2504.19232*, 2025.

Youshao Xiao, Lin Ju, Zhenglei Zhou, Siyuan Li, Zhaoxin Huan, Dalong Zhang, Rujie Jiang, Lin Wang, Xiaolu Zhang, Lei Liang, et al. Antdt: A self-adaptive distributed training framework for leader and straggler nodes. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pp. 5238–5251. IEEE, 2024.

Jiehui Xu, Haixu Wu, Jianmin Wang, and Mingsheng Long. Anomaly transformer: Time series anomaly detection with association discrepancy. *arXiv preprint arXiv:2110.02642*, 2021.

Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875*, 2017.

Guangba Yu, Gou Tan, Haojia Huang, Zhenyu Zhang, Pengfei Chen, Roberto Natella, Zibin Zheng, and Michael R Lyu. A survey on failure analysis and fault injection in ai systems. *ACM Transactions on Software Engineering and Methodology*, 35(1):1–42, 2026.

Kexin Zhang, Qingsong Wen, Chaoli Zhang, Rongyao Cai, Ming Jin, Yong Liu, James Y Zhang, Yuxuan Liang, Guansong Pang, Dongjin Song, et al. Self-supervised learning for time series analysis: Taxonomy, progress, and prospects. *IEEE transactions on pattern analysis and machine intelligence*, 46(10):6775–6794, 2024.

Hang Zhao, Yujing Wang, Juanyong Duan, Congrui Huang, Defu Cao, Yunhai Tong, Bixiong Xu, Jing Bai, Jie Tong, and Qi Zhang. Multivariate time-series anomaly detection via graph attention network. In *2020 IEEE International Conference on Data Mining (ICDM)*, pp. 841–850, 2020. doi: 10.1109/ICDM50108.2020.00093.

Quan Zhou, Changhua Pei, Yuanwei Lu, Difeng Ma, Zexin Wang, Jianhui Li, Yibo Zhu, Daxin Jiang, Dan Pei, Jingjing Li, et al. Tsloc: Self-supervised faulty node localization framework in large-scale training clusters. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 8575–8584, 2026.

# A Telemetry Metrics

Telemetry data has three semantically distinct families: collective communication event logs, GPU hardware monitors, and NIC hardware counters. All metrics are aggregated to per-iteration features: CCL metrics are computed separately for three communication APIs (15 dimensions total); GPU and network metrics each have 9 dimensions, reflecting synchronization state, local computation health, and network transmission status. Table 5 lists all metrics used in *TopoGAT* .

Table 5: Detailed definitions of all telemetry metrics (CCL metrics are computed for each of the three APIs (AllGather, AllReduce, ReduceScatter), totaling 15 dimensions).

| Family | Category | Metric | Description | Unit |
|---|---|---|---|---|
| CCL | Timing offset | `first_off` | Time from iteration start to the first communication event | seconds |
| | | `last_off` | Time from iteration start to the last communication event | seconds |
| | | `off_std` | Standard deviation of event timestamps within one iteration | seconds |
| | Duration | `p50_dur` | 50th percentile of communication event durations | microseconds |
| | | `p99_dur` | 99th percentile of communication event durations | microseconds |
| GPU | Compute state | `gpu_util_pct` | GPU utilization | % |
| | | `spc_clock_mhz` | Shader clock frequency | MHz |
| | | `gpu_temp_c` | GPU core temperature | °C |
| | Power | `power_usage_w` | Total board power consumption | W |
| | | `memory_power_w` | Memory module power consumption | W |
| | | `asic_power_w` | ASIC core power consumption | W |
| | Memory | `mem_util_pct` | GPU memory utilization | % |
| | PCIe | `pcie_rx_mbs` | PCIe receive bandwidth | MB/s |
| | | `pcie_tx_mbs` | PCIe transmit bandwidth | MB/s |
| Network | Packet/byte rate | `rxpck_s` | Received packets per second | pkt/s |
| | | `txpck_s` | Transmitted packets per second | pkt/s |
| | | `rxMB_s` | Receive throughput | MB/s |
| | | `txMB_s` | Transmit throughput | MB/s |
| | Congestion marker | `rxcnp` | Received CNP packets per second | pkt/s |
| | | `txcnp` | Transmitted CNP packets per second | pkt/s |
| | | `rxecn` | Received ECN-marked packets per second | pkt/s |
| | Reliability | `rnr_nak` | RNR NAK count per second | count/s |
| | | `dup_req` | Duplicate request count per second | count/s |

## B ITERATION ANCHOR DETECTION

The anchor detection algorithm identifies a stable periodic (`api`, `data_size`) combination from CCL event streams to infer iteration boundaries without explicit iteration indices. Algorithm 1 provides the pseudo-code. Table 6 reports anchor detection results across all 11 datasets. All datasets meet the $\geq 50\%$ voting threshold, and detected periods align with actual iteration durations. We emphasize that **a detected anchor is not equivalent to a true iteration**: for an actual training iteration spanning $[T, T+60]$, the anchor may span $[T+10, T+70]$. Although not exactly aligned, their durations remain nearly identical (e.g., both around 60s), and extensive validation confirms their sufficiency for reliable detection and localization.

**Algorithm 1** Multi-Rank Voting-Based Iteration Anchor Detection

**Require:** Raw CCL event logs for ranks $0, 1, \ldots, N-1$

**Step 1: Per-rank candidate detection**
**for** each rank $r$ in $\{0, \ldots, 7\}$ **do**
  Extract unique (api, data_size) pairs from event stream
  **for** each pair $(a, s)$ **do**
    Compute inter-arrival times between consecutive events
    Filter outliers using MAD: keep events within $median \pm 3 \times MAD$
    Compute coefficient of variation $CV = \sigma/\mu$ of inter-arrival times
    Compute inlier ratio = inlier events / total events
  **end for**
  $c_r^\star \leftarrow \arg\min_{(a,s)} CV_r(a, s)$, ties broken by max inlier ratio
  $T_r \leftarrow \text{median}$(filtered inter-arrival times of $c_r^\star$)
**end for**

**Step 2: Multi-rank voting**
$V(a, s) \leftarrow \sum_{r=0}^{7} \mathbf{1}[c_r^\star = (a, s)]$
$(a^\star, s^\star) \leftarrow \arg\max_{(a,s):\, V(a,s) \geq 4} V(a, s)$
$T \leftarrow \text{median}\{T_r \mid c_r^\star = (a^\star, s^\star)\}$

**Step 3: Cross-rank iteration boundary alignment**
Extract anchor event timestamps $A_r$ from ranks $r = 0, \ldots, 15$
$r_0 \leftarrow \arg\max_r |A_r|$
**for** each reference anchor time $t_0 \in A_{r_0}$ **do**
  $t_r \leftarrow \arg\min_{t \in A_r} |t - t_0|$ for each rank $r$
  $M \leftarrow \{t_r \mid |t_r - t_0| \leq 0.5T\}$
  Iteration boundary $\leftarrow \text{median}(M)$
**end for**
Return anchor type $(a^\star, s^\star)$, period $T$, and aligned iteration boundaries

Table 6: Anchor detection parameters across all datasets.

| Dataset | Anchor API | Data size (B) | Period (s) | Vote rate (n/8) |
|---|---|---|---|---|
| D1 | AllReduce | 256 | 60.44 | 8/8 (100%) |
| D2 | AllReduce | 256 | 61.57 | 8/8 (100%) |
| D3 | AllReduce | 256 | 102.84 | 7/8 (88%) |
| D4 | AllGather | 2 | 56.50 | 4/8 (50%) |
| D5 | AllReduce | 256 | 123.88 | 7/8 (88%) |
| D6 | AllGather | 4278845440 | 65.51 | 7/8 (88%) |
| D7 | AllGather | 2139422720 | 33.47 | 6/8 (75%) |
| D8 | AllGather | 8 | 50.12 | 8/8 (100%) |
| D9 | AllGather | 8 | 26.90 | 7/8 (88%) |
| D10 | AllGather | 8 | 43.71 | 8/8 (100%) |
| D11 | AllGather | 8 | 15.24 | 8/8 (100%) |

## C ANOMALY SEGMENT DETECTION

We validate the precision of anomaly segment scanner. Since injection labels are recorded at wall-clock granularity, exact segment boundaries may differ; we therefore use overlap-based matching

rather than exact boundary matching: a labeled fault window is counted as recalled if at least one detected segment overlaps it, and a detected segment is counted as a false positive if it overlaps no labeled window. Table 7 summarizes the results. On D1, all 30 injected faults are matched with zero onset delay, with one false positive caused by residual recovery after a fault. On D2, 82 of 86 labeled windows are matched; the four misses are short-interval injections already covered by adjacent matched windows. On the rest datasets, all detected segments match valid injected faults. Pooled across all groups, the scanner achieves a precision of $0.992$ and a recall of $0.970$, with zero onset delay on all matched segments.

Table 7: Segment-level quality of the CUSUM scanner.

| Dataset | Label windows | Matched | Missed | False positive | Onset delay |
|---|---|---|---|---|---|
| D1 | 30 | 30 | 0 | 1 | 0 |
| D2 | 86 | 82 | 4 | 0 | 0 |
| D3 - D11 | 167 | 167 | 0 | 0 | 0 |

## D Model Hyperparameters and Parameters

Table 8 lists the full training configuration and parameters for the default *TopoGAT* model. Total trainable parameters: **158,743** ($\approx$ 0.16M). Parameter count is **independent** of rank count $N$ and edge count $E$ (weights are shared, edges are input as indices only), ensuring consistent behavior across topologies and zero-shot transferability.

Table 8: Default training configuration and parameters.

| | Hyperparameter / parameter | Default value / size |
|---|---|---|
| Hyperparameters | Window size ($T$) | 6 iterations |
| | Input metrics ($D$) | 33 (CCL(15)/GPU(9)/Network(9)) |
| | Model dimension ($d$) | 96 |
| | Number of GAT layers | 2 |
| | Number of attention heads | 4 (head dim = 24) |
| | Edge types | 5 (TP/DP/PP/global/time) |
| | Encoder residual blocks | 2 |
| | Dropout (encoder / GAT) | 0.1 |
| | Learning rate | $10^{-3}$ |
| | Optimizer | Adam |
| | Training epochs | 8 |
| | Batch size | 32 |
| Parameters | Metric-family encoders (3) | 61,056 |
| | Family gate ($F \times F$) | 12 |
| | GAT (2 layers) | 94,090 |
| | Edge-type projections (per layer) | 46,080 |
| | Attention source / destination vectors (per layer) | 480 / 480 |
| | Edge-type aggregation weights (per layer) | 5 |
| | Post-GAT layer norms | 384 |
| | Decoder ($d \rightarrow D$) | 3,201 |
| | **Total trainable** | **158,743** |

## E Fault Injection

We inject two broad categories of faults on individual ranks to simulate real-world single-machine failures: hardware-side physical anomalies directly imposed on GPU and NIC, and framework-side computation and communication slowdowns introduced via a modified Megatron-LM framework. All fault types are summarized in Table 9.

## F Per-Dataset Backbone Comparison

Table 10 reports Hit@3 on each of the 11 evaluation datasets under the bare and full protocols.

Table 9: Fault types.

| Category | Fault type | Injection point | Description |
|---|---|---|---|
| Hardware-side | `gpu_stress` | 1 rank | GPU compute stress load, reducing available compute capacity |
| | `gpu_underclock` | 1 rank | GPU clock frequency reduction |
| | `packet_loss` | 1 rank | Network interface packet loss |
| | `traffic_limit` | 1 rank | NIC bandwidth throttling |
| | `mixed` | $\geq$ 2 ranks | Mixed computation and communication faults |
| Framework-side | `gqa_compute` | 1 rank | GQA computation kernel slowdown |
| | `rope` | 1 rank | RoPE positional embedding computation slowdown |
| | `self_attention` | 1 rank | Self-attention computation slowdown |
| | `swiglu_delay` | 1 rank | SwiGLU activation computation slowdown |
| | `tp_slow` | 1 rank | Peer-to-peer TP communication slowdown |
| | `pp_slow` | 1 rank | Peer-to-peer PP communication slowdown |
| | `mixed` | $\geq$ 2 ranks | Mixed computation and communication faults |

Table 10: Per-dataset backbone comparison: Hit@3 under the bare and full protocols. Each cell reports *online/offline* Hit@3. (AT means Anomaly Transformer, VAE means LSTM-VAE (Minder))

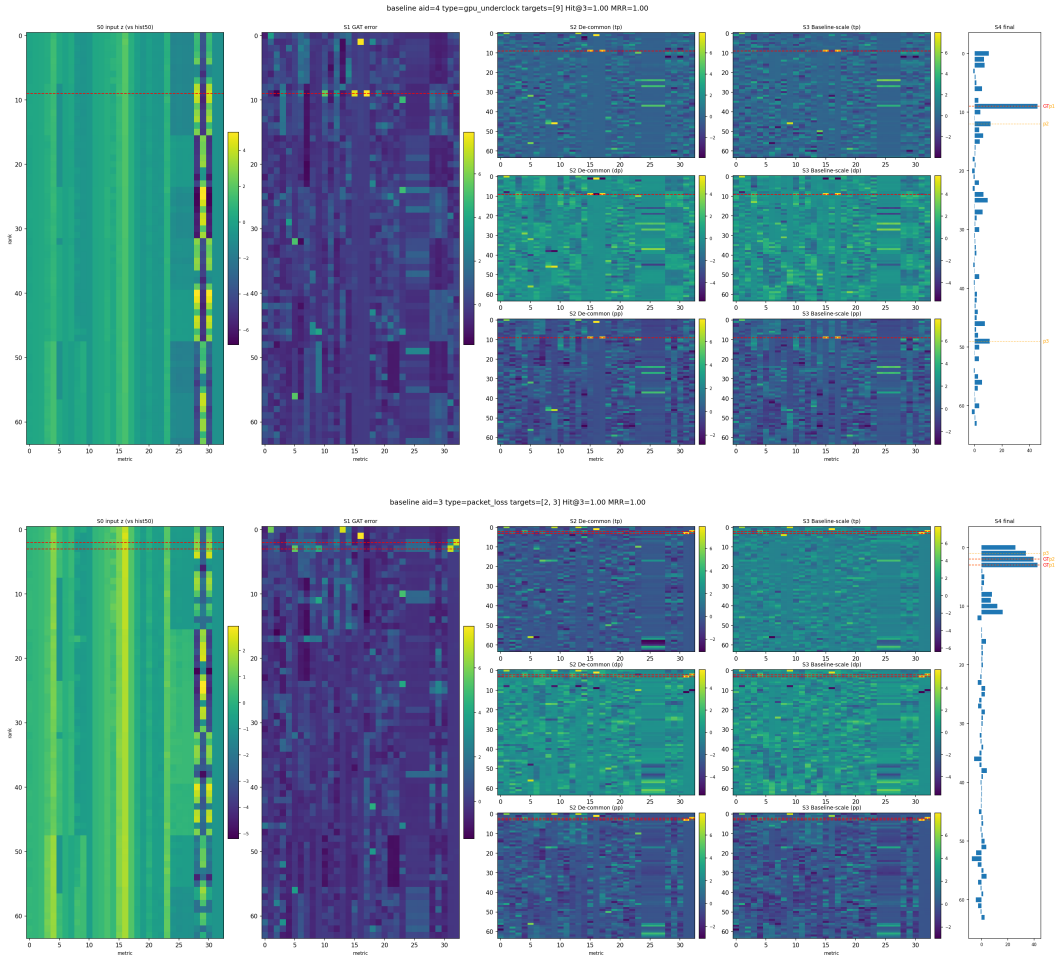
| Backbone | Hit@3 | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | D1 | D2 | D3 | D4 | D5 | D6 | D7 | D8 | D9 | D10 | D11 |
| ***Bare*** | | | | | | | | | | | |
| MLP | 0.773/**0.867** | 0.744/0.899 | 0.626/0.684 | 0.688/0.650 | 0.757/0.833 | 0.742/0.750 | 0.712/0.833 | 0.783/**0.900** | 0.744/0.800 | 0.946/0.900 | 0.848/0.812 |
| GRU | 0.745/0.800 | **0.810/0.926** | 0.580/0.579 | 0.692/0.700 | 0.736/0.778 | 0.751/0.800 | **0.737**/0.800 | 0.783/0.800 | 0.753/0.850 | 0.952/0.900 | 0.789/0.812 |
| LSTM | **0.801**/0.800 | 0.523/0.684 | 0.538/0.650 | 0.743/**0.778** | 0.656/0.750 | 0.730/0.800 | 0.721/0.857 | 0.806/0.800 | 0.776/0.850 | **0.973**/0.900 | 0.761/0.875 |
| 1D-CNN | 0.776/0.833 | 0.723/0.870 | 0.603/0.737 | 0.731/0.700 | 0.743/0.778 | 0.742/0.800 | 0.734/0.800 | 0.806/0.800 | 0.744/0.850 | 0.952/0.900 | 0.841/0.812 |
| STConv | 0.791/0.833 | 0.697/0.850 | 0.489/0.579 | 0.638/0.650 | 0.743/0.778 | 0.742/0.800 | 0.730/0.800 | 0.798/**0.900** | 0.781/0.800 | 0.946/0.900 | 0.792/0.938 |
| AT | 0.782/0.833 | 0.797/0.870 | 0.494/0.750 | 0.604/0.750 | 0.654/0.850 | 0.819/0.833 | 0.693/0.867 | 0.806/0.800 | 0.753/0.850 | 0.912/0.813 | 0.892/0.938 |
| TSLoc | 0.670/0.800 | 0.804/0.878 | 0.448/0.474 | 0.392/0.450 | 0.701/0.778 | 0.389/0.500 | 0.489/0.633 | 0.535/0.600 | 0.493/0.600 | 0.952/**1.000** | 0.427/0.688 |
| VAE | 0.723/**0.867** | 0.583/0.736 | 0.649/**0.789** | 0.738/0.600 | 0.743/0.833 | 0.729/0.700 | 0.624/0.833 | **0.814**/0.700 | **0.826/0.900** | 0.918/0.900 | 0.841/0.875 |
| TopoGAT | 0.745/**0.867** | 0.725/0.899 | **0.724/0.789** | **0.746**/0.650 | **0.854/0.944** | 0.738/**0.850** | 0.683/0.833 | 0.744/0.700 | 0.749/0.750 | **0.973**/0.900 | **0.907/1.000** |
| ***Full*** | | | | | | | | | | | |
| MLP | 0.810/0.867 | 0.879/0.926 | 0.805/0.842 | 0.800/0.850 | 0.778/0.722 | 0.869/**0.950** | 0.762/0.800 | 0.791/0.800 | 0.776/0.800 | 0.918/**0.900** | 0.974/**1.000** |
| GRU | 0.779/0.867 | 0.891/0.944 | 0.793/0.842 | 0.850/**0.950** | 0.701/0.722 | 0.846/0.900 | 0.777/0.800 | 0.806/0.800 | 0.749/0.750 | **0.959/0.900** | 0.941/0.938 |
| LSTM | 0.804/0.867 | 0.830/0.852 | **0.842/0.900** | 0.760/0.788 | 0.864/0.900 | 0.796/0.833 | 0.880/0.915 | 0.899/**1.000** | 0.740/0.750 | 0.878/**0.900** | 0.958/0.953 |
| 1D-CNN | 0.838/0.900 | 0.851/0.884 | 0.793/0.842 | 0.831/0.850 | 0.778/0.833 | 0.851/0.900 | 0.799/0.800 | 0.853/0.900 | 0.772/0.800 | 0.871/**0.900** | 0.951/0.938 |
| STConv | 0.838/0.900 | 0.900/0.947 | 0.747/0.842 | 0.742/0.800 | 0.785/0.833 | **0.873/0.950** | 0.834/**0.933** | 0.899/**1.000** | 0.772/0.850 | 0.884/**0.900** | **0.979/1.000** |
| AT | 0.826/0.833 | 0.867/0.918 | 0.780/0.878 | 0.837/0.900 | 0.826/**0.944** | 0.869/**0.950** | 0.737/0.831 | 0.946/**1.000** | 0.776/0.800 | 0.877/**0.900** | 0.964/0.938 |
| TSLoc | 0.763/0.900 | 0.868/0.912 | 0.563/0.632 | 0.696/0.800 | 0.778/0.889 | 0.588/0.650 | 0.470/0.533 | 0.512/0.600 | 0.607/0.700 | 0.803/**0.900** | 0.918/0.938 |
| VAE | **0.875/0.933** | 0.887/0.953 | 0.822/0.842 | 0.796/0.700 | 0.812/**0.944** | 0.855/0.900 | 0.843/0.867 | 0.884/0.900 | 0.904/0.950 | 0.891/0.800 | 0.910/0.938 |
| TopoGAT | 0.804/0.900 | **0.946/0.981** | 0.764/0.895 | **0.900**/0.900 | **0.882/0.944** | 0.864/**0.950** | **0.884/0.933** | **0.992/1.000** | **0.927/1.000** | 0.878/**0.900** | 0.943/**1.000** |

## G Extend Metric Family Robustness

Table 11 reports Hit@3 when family metrics are removed during training and evaluation.

Table 11: Offline Hit@3 when a metric family is removed during both training and evaluation.

| Fault type | full | −GPU | −CCL | −Net | −GPU-CCL | −GPU-Net | −CCL-Net |
|---|---|---|---|---|---|---|---|
| gpu_stress | 0.78 | 0.22 | 0.17 | 0.31 | 0.06 | 0.25 | 0.30 |
| gpu_underclock | 1.00 | 0.22 | 0.50 | 0.50 | 0.09 | 0.14 | 0.50 |
| packet_loss | 1.00 | 0.50 | 0.50 | 0.22 | 0.50 | 0.27 | 0.09 |
| traffic_limit | 0.87 | 0.45 | 0.41 | 0.22 | 0.42 | 0.21 | 0.08 |
| gqa_compute | 1.00 | 0.66 | 0.79 | 1.00 | 0.00 | 0.74 | 0.86 |
| rope | 0.80 | 0.11 | 0.50 | 0.61 | 0.00 | 0.21 | 1.00 |
| self_attention | 1.00 | 0.00 | 1.00 | 1.00 | 0.00 | 0.21 | 1.00 |
| swiglu_delay | 1.00 | 0.58 | 1.00 | 1.00 | 0.45 | 0.47 | 1.00 |
| pp_comm | 1.00 | 1.00 | 0.70 | 1.00 | 0.70 | 1.00 | 0.60 |
| tp_comm | 1.00 | 1.00 | 0.14 | 1.00 | 0.29 | 1.00 | 0.29 |




## H Extended Error Field Visualization

We visualize the error field evolution pipeline for 2 representative faults: GPU Underclock, and Packet Loss. Each has 5 stages:

1. Input z-score of raw telemetry
2. Raw prediction error field
3. Error field after De-common
4. Error field after Baseline-scale
5. Final score

Raw prediction errors already contain a detectable root-cause signal but are contaminated by group-wide common-mode noise. De-common removes the shared group offset, and Baseline-scale normalizes heterogeneous baseline fluctuations, making the root-cause rank clearly stand out. TP, PP, and DP views show consistent root-cause localization, the three-view averaging in the scoring pipeline combines these complementary signals for robust performance.

## I Supplementary Experiments

### I.1 Attention and Edge-Type Weight Analysis

This section investigates two core questions regarding the learned edge-type attention mechanism: (1) why it fails to differentiate edge types under the default training regime, and (2) what conditions

unlock such differentiation, and whether it improves fault localization. We demonstrate that uniform attention is not a design flaw but the *optimal* solution under normal-only self-supervised training. Furthermore, the same attention interface can deliver measurable localization gains under richer supervision, without requiring additional rule-based decision layers.

#### I.1.1 Why Uniform Attention Is Optimal Under Normal-Only Training

For edge-type attention to both *differentiate* across edge types and *benefit* localization, three necessary conditions must hold simultaneously:

- (C-a) Identifiability: The scoring function can distinguish between peers and identify which to attend to.
- (C-b) Usefulness: Attending to specific peers improves prediction quality.
- (C-c) Readout transmissibility: The benefits of differentiated attention are not erased by the downstream scoring pipeline.

Under normal-only training on a single benchmark dataset, conditions C-a and C-b are not satisfied, and C-c renders even an *oracle* attention indistinguishable from uniform attention. Table 12 summarizes the diagnostic experiments designed to verify each condition.

Table 12: Diagnostic experiments. Each probe targets one of the three conditions to isolate the bottleneck of attention differentiation.

| Condition | Probe | Observation |
|---|---|---|
| C-a | Within-group permutation test | Output change $\leq 2.98 \times 10^{-7}$ |
| C-a | Linear probe for group identity | $0.141$ (random: $0.125$) |
| C-a | Root-cause decoding AUC | Raw $0.943 >$ residual $0.847 >$ hidden $0.767$ |
| C-b | $\alpha$ deviation from uniform (L1) | $10^{-3}$ level, tracks group size, not fault semantics |
| C-b | Edge-type scalar weights $w_k$ | $0.95$–$1.01$ (near initialization) |
| C-c | Replace learned $\alpha$ with uniform | 0 flips over 3,232 windows / 283 events |
| C-c | Oracle $\alpha$ (pointed at root cause) | 1 of 30 events flipped ($p = 1.0$) |
| C-c | Typed-mean trained from scratch | $0.887/0.941$ vs. full GAT $0.889/0.946$ |
| Causal | Synthetic teacher, C-a and C-b present | $\alpha \to 0.69$; recovers $93\%$ oracle gain |
| Causal | Synthetic teacher, C-a absent | No learning despite oracle $+0.67$ |

Three key observations emerge from the diagnostic probes:

First, **condition C-a fails**: the model has no access to within-group identity information. The within-group permutation test shows that shuffling peer identities changes outputs by at most $2.98 \times 10^{-7}$, indicating the model is permutation-invariant within groups. A linear probe for group identity achieves only 0.141 accuracy, confirming that group membership cannot be recovered from learned representations. The root-cause decoding AUC further shows that information about root causes degrades progressively from raw features (0.943) to residual features (0.847) to hidden representations (0.767), leaving insufficient signal for peer discrimination.

Second, **conditions C-b fails**: uniform averaging is the MSE-optimal solution for the induced data distribution. Within-group full connectivity, permutation-invariant prediction targets, and highly similar peer features collectively leave no meaningful gradient direction for the attention scoring function to exploit. The L1 deviation of attention weights $\alpha$ from uniform is on the order of $10^{-3}$, and tracks group size rather than faults. Edge-type scalar weights $w_k$ remain in the range 0.95–1.01, close to their initialization values, confirming that no meaningful differentiation is learned.

Third, **condition C-c fails**: even perfectly routed oracle attention would not change the final readout. Replacing learned attention with uniform weights produces bit-identical outputs across all 283 events. An oracle attention pointed directly at the root cause flips only 1 out of 30 event rankings ($p = 1.0$, statistically insignificant). Furthermore, a typed-mean variant trained from scratch—preserving edge typing and type-wise projections but replacing node-level attention with uniform averaging—achieves 0.878/0.941 Hit@3, statistically indistinguishable from the full GAT.

Finally, synthetic teacher experiments confirm that the bottleneck is not model capacity. When identifiability and usefulness are both introduced via a synthetic teacher, the same architecture spontaneously shifts $\alpha$ to 0.69 and recovers 93% of the oracle gain. When the intervention signal is removed, no learning occurs despite the oracle still providing +0.67 potential gain. We therefore conclude that weak attention differentiation is not an architectural deficiency, but the optimal solution under the induced data distribution.

### I.1.2 What Unlocks Differentiation, and When It Helps

We next systematically probe which mechanisms can unlock attention differentiation. We train 13 variants spanning five categories: (1) unsupervised auxiliary tasks (identity prediction, role prediction, edge-type classification, peer-group classification, etc.); (2) regularizers (entropy minimization, score-gap regularization); (3) identity encodings (rank embedding, role embedding); (4) real-label supervision (output cross-entropy, attention cross-entropy), and (5) data-side modifications (training on anomalous windows). Table 13 reports the differentiation profile and zero-shot performance of representative variants.

Table 13: Mechanisms that unlock attention differentiation. The *shuffled* control trains attention cross-entropy with randomized root-cause labels to isolate label-specific effects. $\alpha$ **dev.**: L1 distance of Layer-1 attention weights from uniform distribution; **Held-out align.**: Alignment gain of $\alpha$ toward the root-cause source on held-out data, in percentage points; $\Delta$ values are relative to the same-pipeline unsupervised baseline.

| **Variant** | $\alpha$ **dev.** | **Held-out align.** | **AVG** $\Delta$ |
|---|---|---|---|
| Baseline (no supervision) | $4.4 \times 10^{-4}$ | $\approx 0$ | $0/0$ |
| Unsupervised auxiliaries (8 types) | $\leq 1.5 \times 10^{-3}$ | $\approx 0$ | $-2.3$ to $+0.3$ / $-2.5$ to $+0.8$ |
| Rank embedding | $2.7 \times 10^{-2}$ | $\approx 0$ | $-12.7/-10.0$ |
| Attention CE ($\lambda$=0.5) | $7.6 \times 10^{-3}$ | $+4.2$ pp | $-5.0/-3.3$ |
| Attention CE ($\lambda$=2.0) | $5.1 \times 10^{-2}$ | $+9.8$ pp | $-10.4/-10.4$ |
| Attention CE, shuffled | $4.1 \times 10^{-2}$ | $+1.5$ pp | $-11.1/-8.7$ |

Three key findings emerge:

First, **unsupervised mechanisms cannot induce meaningful differentiation**. All eight unsupervised variants yield $\alpha$ deviations at the same level as the baseline, with near-zero held-out alignment. None produce semantically meaningful attention differentiation.

Second, **only two mechanisms drive differentiation, but with different trade-offs**. Identity encoding via rank embedding pushes $\alpha$ far from uniform but produces zero held-out alignment and severely degrades zero-shot accuracy. Real-label supervision via attention cross-entropy exhibits a clear dose–response relationship: increasing the loss weight $\lambda$ from 0.5 to 2.0 raises both $\alpha$ deviation (from $7.6 \times 10^{-3}$ to $5.1 \times 10^{-2}$) and held-out alignment (from +4.2 pp to +9.8 pp). The shuffled-label control reproduces the magnitude of $\alpha$ deviation but not the alignment, confirming that alignment stems from genuine label-specific learning rather than generic attention perturbation.

Third, **differentiation alone does not transfer to zero-shot performance**. Every strongly differentiating configuration degrades zero-shot Hit@3, forcing attention to differentiate disrupts the error field that drives unsupervised localization.

Differentiation becomes useful for localization when supervision spans multiple datasets. Under a multi-dataset supervision protocol—Each dataset using its first 40% of anomalous events for unified training and holding out the remaining 60% plus leave-one-dataset-out (LODO) sets—attention CE combined with readout rank CE drives $\alpha$ to 25–40$\times$ the noise level, with +18.7 pp held-out root-cause alignment (positive on all datasets). When the shaped attention weights are plugged into an attention-aware (A-aware) readout, held-out event localization improves consistently across seeds. Table 14 reports the full localization results.

Table 14 reveals an inherent tension: joint training of attention differentiation and standard localization incurs a cost in standard-field performance, because the attention-supervision gradient perturbs the main error field that drives unsupervised scoring. We resolve this tension with a two-stage *shape-then-repair* recipe: (1) **Shape**: Train the attention weights using the multi-dataset supervised recipe

Table 14: Supervised attention: differentiation and localization performance. *Standard field*: performance of the original unsupervised scoring pipeline. *A-readout*: attention-aware readout $S_r = z(\text{field}) + \beta\, z(\sum_{e \text{ from } r} \alpha_e \cdot \text{field}(\text{dst}))$ with $\beta$=0.5. $\Delta$ values are relative to the same-pipeline unsupervised baseline.

| **Recipe** | $\alpha$ **dev.** | **Held-out align.** | **Standard field** $\Delta$ | **A-readout** $\Delta$ |
|---|---|---|---|---|
| Unsupervised baseline | $\sim 10^{-3}$ | $\approx 0$ | $0/0$ | $0/0$ |
| Multi-dataset supervised | $3.65 \times 10^{-2}$ | $+18.7$ pp | $-1.8/-1.7$ pp | $+4.6/+1.6$ pp |
| **Shape-then-repair** | $4.0 \times 10^{-2}$ | $+19.8$ pp | $+1.4/+2.5$ pp | $+4.6/+1.6$ pp |

to shape $\alpha$ toward root-cause sources; (2) **Repair**: Freeze the shaped attention weights, then retrain only the decoder to restore error field quality. This recipe exceeds the unsupervised baseline in both Standard field and A-readout.

#### I.1.3 Summary and Outlook

Under the default normal-only, single-dataset training regime, uniform edge-type attention is the optimal solution for the induced data distribution. The model has no identifiable peer to prefer, the training objective provides no gradient to prefer one, and even a perfect oracle attention would leave the final readout unchanged.

Nonetheless, the GAT design is the correct engineering choice for three reasons: (1) **No degradation under unsupervised training**: The base unsupervised protocol achieves the best average performance among all evaluated backbones. Replacing learned attention with uniform weights or retraining with uniform averaging yields statistically indistinguishable localization scores; (2) **Measurable gains under richer supervision**: With real labels spanning multiple datasets, the same attention interface differentiates strongly and consistently improves held-out localization across seeds; (3) **End-to-end differentiability**: The attention interface is fully differentiable end-to-end. When label-rich training data becomes available, edge-type importance can be specialized directly via gradient descent—no separate rule-based layer, no per-task manual tuning.

### I.2 Why TopoGAT Works

Section 3 identifies two core challenges induced by synchronization propagation: *common-mode slowdown* and *victim anomaly dominance*. We show that these issues are not unique to *TopoGAT* —they degrade the performance of all evaluated backbones—and that the three post-processing operations introduced below effectively mitigate both effects. Let $\varepsilon_{w,r,d}$ denote the per-window mean squared prediction error. All statistics are computed solely on unlabeled normal windows, and aggregated into the window-level score $S_{w,r} = \frac{1}{|\mathcal{V}|} \sum_v \max_d z^2_{w,r,d}$.

**DC (De-common):** Faults induce an unknown collective error elevation within their peer groups, making raw absolute errors incomparable across ranks and groups. Subtracting the peer median:

$$\tilde{\varepsilon}_{w,r,d} = \varepsilon_{w,r,d} - \text{med}_{r' \in \mathcal{N}(r)} \varepsilon_{w,r',d},$$

removes this shared error component, retaining only each rank's relative deviation from its peer group for downstream scoring.

**BS (Baseline scaling):** Ranks exhibit heterogeneous baseline error scales, such that unscaled scores are dominated by ranks with the largest error magnitudes—typically victim nodes. Normalizing by the per-rank 99th percentile error magnitude:

$$\hat{\varepsilon}_{w,r,d} = \tilde{\varepsilon}_{w,r,d} / \sigma_{r,d}, \quad \sigma_{r,d} = \text{p99}_w |\tilde{\varepsilon}_{w,r,d}|,$$

equalizes error scales across all ranks.

**CZ (Cluster-z):** Standardizing errors across ranks within each metric dimension:

$$z_{w,r,d} = \big(\hat{\varepsilon}_{w,r,d} - \bar{\hat{\varepsilon}}_{w,\cdot,d}\big) / s_{w,d},$$

eliminates residual shared baseline shifts and per-metric scale disparities, ensuring no single metric dominates $S_{w,r}$.

Together, these operations align all ranks and metrics onto a unified scale, enabling a single detection threshold to function effectively without any labeled data. Table 15 presents results for all protocol combinations across all backbones, averaged over all datasets. Two key findings emerge: (1) without post-processing, performance collapses across all backbones, highlighting the inherent challenge of fault localization under parallel training; (2) *TopoGAT* consistently ranks first under every protocol combination, demonstrating that its topology-aware architecture delivers gains beyond the calibration protocol alone.

Table 15: Backbone × calibration protocol ablation. Average Hit@3 across all datasets (*online / offline*), with only the evaluation-side protocol modified. *None*: no post-processing applied.

| Architecture | None | DC | BS | CZ | DC+BS | DC+CZ | BS+CZ | Full |
|---|---|---|---|---|---|---|---|---|
| TopoGAT | **0.293 / 0.399** | **0.317 / 0.447** | **0.753 / 0.813** | **0.781 / 0.835** | **0.832 / 0.847** | **0.826 / 0.914** | **0.830 / 0.881** | **0.889 / 0.946** |
| LSTM | 0.235 / 0.257 | 0.288 / 0.356 | 0.369 / 0.382 | 0.730 / 0.795 | 0.714 / 0.724 | 0.802 / 0.892 | 0.731 / 0.735 | 0.837 / 0.874 |
| VAE | 0.194 / 0.249 | 0.234 / 0.302 | 0.322 / 0.362 | 0.744 / 0.794 | 0.745 / 0.720 | 0.785 / 0.860 | 0.784 / 0.821 | 0.862 / 0.884 |
| STConv | 0.238 / 0.239 | 0.285 / 0.348 | 0.321 / 0.318 | 0.741 / 0.803 | 0.662 / 0.677 | 0.808 / 0.873 | 0.733 / 0.748 | 0.841 / 0.905 |
| 1D-CNN | 0.250 / 0.278 | 0.294 / 0.367 | 0.372 / 0.375 | 0.763 / 0.807 | 0.707 / 0.724 | 0.801 / 0.889 | 0.751 / 0.786 | 0.835 / 0.868 |
| AT | 0.289 / 0.376 | 0.313 / 0.447 | 0.354 / 0.390 | 0.755 / 0.829 | 0.591 / 0.603 | 0.822 / 0.880 | 0.705 / 0.771 | 0.846 / 0.899 |
| MLP | 0.267 / 0.295 | 0.308 / 0.384 | 0.399 / 0.416 | 0.746 / 0.832 | 0.710 / 0.731 | 0.800 / 0.879 | 0.771 / 0.775 | 0.833 / 0.860 |
| GRU | 0.237 / 0.216 | 0.289 / 0.338 | 0.384 / 0.384 | 0.757 / 0.795 | 0.709 / 0.739 | 0.796 / 0.872 | 0.745 / 0.793 | 0.827 / 0.856 |
| TSLoc | 0.217 / 0.227 | 0.235 / 0.260 | 0.214 / 0.199 | 0.573 / 0.673 | 0.371 / 0.381 | 0.666 / 0.828 | 0.525 / 0.569 | 0.688 / 0.769 |

The ablation lattice confirms that calibration is *necessary*: no backbone achieves practical performance without it, and *TopoGAT* outperforms all alternatives under every combination. However, calibration alone is not *sufficient*. To disentangle the contributions of calibration and the learned predictor, we decompose the full pipeline into heuristic baselines, model-free statistics, and the full model system. Table 16 reports online and offline Hit@3 averaged across 11 datasets.

Table 16: Component decomposition analysis.

| Configuration | Calibration | AVG (on / off) |
|---|---|---|
| *Heuristic baseline* | | |
| Max iteration time | — | 0.154 / 0.153 |
| *Model-free statistics (10 types)* | | |
| Best over 10 statistics | full | 0.273 / 0.297 |
| *Model* | | |
| Random-init *TopoGAT* | full | 0.705 / 0.739 |
| Trained *TopoGAT* | — | 0.293 / 0.399 |
| Trained *TopoGAT* | full | **0.889 / 0.946** |

Table 16 reveals three key insights. *First, the model alone is insufficient*: without calibration, *TopoGAT* achieves only 0.293 / 0.399 Hit@3. *Second, calibration alone is equally insufficient*: ten model-free statistics (including raw magnitude, global z-score, within-window temporal z-score, etc.) remain below 0.30 even with full calibration. *Third, the two components are strongly complementary*: their combination reaches 0.889 / 0.946. Calibration cannot amplify a signal absent from the model output, and the model's raw error field is overwhelmed by common-mode noise and heterogeneous baselines without calibration. Two additional observations further corroborate this conclusion: synchronization propagation renders duration-based localization ineffective, and trained model weights yield statistically significant performance gains over random initialization under the same calibration pipeline.

These results collectively demonstrate that *TopoGAT* 's performance stems from the *complementarity* of a learnable error field and calibration: neither component alone is sufficient. The most substantial gains are observed on unseen topologies, indicating that the graph-based predictor provides transferable topological inductive bias that cannot be replicated by post-hoc calibration alone.

### I.3 Statistical Validation and Multi-Seed Robustness

#### I.3.1 Statistical Significance

Table 1 presents point estimates of our core performance results. To quantify estimation uncertainty, we perform bootstrap resampling across 283 fault events and conduct paired significance

tests against four representative baselines: LSTM, Minder, Anomaly Transformer, and TSLoc. Table 17 reports global event-level Hit@3 with 95% bootstrap confidence intervals (CIs), alongside paired test $p$-values for each baseline compared against *TopoGAT* .

Table 17: Global event-level Hit@3 with 95% bootstrap CIs (computed over 283 fault events). Paired $p$-values are calculated against *TopoGAT* in the offline setting.

| Method | Online Hit@3 [95% CI] | Offline Hit@3 [95% CI] | Paired $p$ (offline) |
|---|---|---|---|
| TopoGAT | **0.796 [0.749, 0.842]** | **0.943 [0.914, 0.968]** | – |
| LSTM | 0.771 [0.724, 0.819] | 0.878 [0.841, 0.914] | $< 0.001$ |
| Minder | 0.796 [0.745, 0.842] | 0.903 [0.867, 0.936] | 0.012 |
| Anomaly Transformer | 0.674 [0.620, 0.728] | 0.885 [0.846, 0.921] | 0.002 |
| TSLoc | 0.599 [0.541, 0.656] | 0.796 [0.745, 0.842] | $< 0.001$ |

*TopoGAT* achieves an offline Hit@3 of 0.943 with a tight 95% CI of [0.914, 0.968], and an online Hit@3 of 0.796 with a CI of [0.749, 0.842]. Paired tests confirm that our offline improvements over all baselines are statistically significant, with consistent performance gaps also observed in the online setting. This verifies that the performance gains of *TopoGAT* are not statistical artifacts and hold under event-level aggregation.

### I.3.2 Multi-Seed Stability

To further rule out sensitivity to random seed selection, we evaluate *TopoGAT* with 10 consecutive training seeds (2024–2033; default: 2028). As summarized in Table 18, average Hit@3 remains highly stable: standard deviations are only 1.03–1.15 points, and deviations from the default seed are at most 1.11 points, confirming that our reported performance is representative.

Table 18: Multi-seed stability across 10 consecutive training seeds. Standard deviations and differences are reported in points (1 point = 0.01 Hit@3).

| Metric | Group mean | Group std | Diff. from seed 2028 |
|---|---|---|---|
| Online | 0.8797 | 1.03 | $-0.93$ |
| Offline | 0.9349 | 1.15 | $-1.11$ |

## I.4 Efficiency Analysis

We profile *TopoGAT* under an online deployment: batch-1 streaming inference on CPU only (4 threads), including Python and graph-construction overhead. Table 19 summarizes the results.

Table 19: FLOPs are per-window forward only. Forward is the median cost per-window. Scoring includes De-common, Baseline-scale, and Cluster-z. $\sigma$ calibration is a one-time cost.

| Ranks | FLOPs/window (G) | Forward/window (ms) | Scoring/window (ms) | $\sigma$ computation (s) |
|---|---|---|---|---|
| 16 | 0.031 | 10.0 | 0.24 | 0.49 |
| 32 | 0.055–0.062 | 11.1–11.2 | 0.24–0.30 | 0.48–0.62 |
| 64 | 0.109–0.123 | 17.3–20.5 | 0.45–0.57 | 0.92–1.16 |
| 128 | 0.247 | 29.4 | 0.94 | 1.93 |

*TopoGAT* has only 158,743 trainable parameters, independent of the number of ranks and the number of edges. Per-window FLOPs stay below 0.25 G even at 128 ranks. On a 64-rank cluster, a single window takes $\sim$17 ms latency on CPU, which is less than 0.2% of a typical LLM training iteration (15–124 s). Latency grows near-linearly with the number of ranks (10 ms at 16 ranks to 29 ms at 128 ranks), while the scoring chain costs at most 0.94 ms per window and the one-time $\sigma$ calibration takes at most 1.93 s for 128 ranks.

### I.5 Calibration Order Ablation

We compare the default order (De-common → Baseline-scale) with the reversed order (Baseline-scale → De-common). Both share the same raw error field. Table 20 reports the results.

Table 20: Calibration order ablation (Hit@3, online / offline).

| Dataset | Default (De-common → Baseline-scale) | Reversed (Baseline-scale → De-common) |
|---|---|---|
| D1 | 0.804 / 0.900 | 0.748 / 0.733 |
| D2 | 0.946 / 0.981 | 0.936 / 0.981 |
| D3 | 0.764 / 0.895 | 0.828 / 0.789 |
| D4 | 0.900 / 0.900 | 0.838 / 0.850 |
| D5 | 0.882 / 0.944 | 0.924 / 0.889 |
| D6 | 0.864 / 0.950 | 0.928 / 0.950 |
| D7 | 0.884 / 0.933 | 0.865 / 0.933 |
| D8 | 0.992 / 1.000 | 0.984 / 1.000 |
| D9 | 0.927 / 1.000 | 0.913 / 1.000 |
| D10 | 0.878 / 0.900 | 0.871 / 0.900 |
| D11 | 0.943 / 1.000 | 0.954 / 1.000 |
| **Avg** | 0.889 / **0.946** | 0.890 / 0.911 |

The reversed order is not catastrophic online because the EMA readout smooths per-window differences, but it consistently weakens offline attribution, especially on D1 ($-16.7$ offline). This supports the default order: De-common first removes group-wise common-mode, and Baseline-scale then normalizes the residual by each rank's own normal fluctuation. Reversing the order would amplify common-mode errors on victim ranks with small baseline noise.

### I.6 Inference-Time Edge-Type Ablation

We remove each edge type at inference time (without retraining) to measure the runtime signal contribution of each edge type. Table 21 reports results.

Table 21: Inference-time edge-type ablation: Hit@3 (online / offline).

| Variant | baseline | AVG |
|---|---|---|
| full (default) | **0.804 / 0.900** | 0.889 / 0.946 |
| − TP | 0.791 / 0.867 | **0.895 / 0.952** |
| − PP | 0.801 / 0.867 | 0.876 / 0.939 |
| − DP | 0.798 / 0.833 | 0.890 / 0.944 |
| − Global | 0.781 / 0.900 | 0.868 / 0.934 |
| − Time | 0.793 / 0.833 | 0.881 / 0.928 |

Removing spatial edges at inference time has little effect, and removing TP edges even improves the average slightly. This is consistent with our design: De-common already subtracts group-wise common-mode at inference, so spatial smoothing becomes redundant and can slightly dilute the root-cause contrast. In contrast, removing Global or Time edges consistently degrades performance, confirming that cluster-level context and temporal continuity carry runtime signals that De-common cannot replace. The necessity of typed edges as a training-time inductive bias is established by the training-time ablation in Table 3.

### I.7 Graph Definition Ablation

We evaluate alternative graph definitions by modifying only the node/edge structure while keeping all other components fixed. Table 22 reports results.

The default rank-node with typed parallelism edges achieves the best performance. Alternate node definitions perform worse due to increased dimensionality or lost structural context. Removing edge type semantics or replacing topology with full connectivity degrades performance, confirming that parallelism topology as inductive bias is critical. Factoring the spatio-temporal graph into $T$

Table 22: Graph definition ablation: Hit@3 (online / offline).

| Definition | baseline | AVG | Params |
|---|---|---|---|
| **default** | **0.804 / 0.900** | **0.889 / 0.946** | 158,743 |
| node_metric (node = metric) | 0.798 / 0.833 | 0.837 / 0.884 | 237,759 |
| node_rank_metric (node = (rank, metric) pair) | 0.769 / 0.767 | 0.835 / 0.883 | 168,606 |
| node_dual (node = typed edge connection, edge = rank) | 0.732 / 0.733 | 0.778 / 0.852 | 195,799 |
| edge_anon (untyped neighbors) | 0.794 / 0.833 | 0.831 / 0.909 | 158,743 |
| edge_full (all nodes fully connected) | 0.748 / 0.800 | 0.808 / 0.845 | 158,743 |
| edge_temporal_stacked (spatial layers joined by vertical time edges) | 0.788 / 0.833 | 0.820 / 0.851 | 339,242 |

per-iteration spatial layers chained by vertical temporal edges costs $-6.9$ / $-9.5$ points on average *despite* using $2.1\times$ the parameters.

### I.8 Robustness to Missing Data

We evaluate robustness to missing data at inference time. Missing ranks are sampled from non-root-cause ranks only. Table 23 reports results.

Table 23: Missing data robustness: Hit@3 (online / offline).

| Variant | Description | baseline | AVG |
|---|---|---|---|
| default | No missing data | **0.804 / 0.900** | **0.889 / 0.946** |
| −1 rank | Single non-root rank down | 0.361 / 0.367 | 0.437 / 0.494 |
| −1 pod | One TP group down | 0.620 / 0.700 | 0.617 / 0.673 |
| −4 ranks | 4 random non-root ranks down | 0.112 / 0.033 | 0.169 / 0.140 |
| −8 ranks | 8 random non-root ranks down | 0.012 / 0.000 | 0.290 / 0.288 |
| −16 ranks | 16 random non-root ranks down | 0.262 / 0.167 | 0.348 / 0.349 |
| − GPU family | GPU telemetry down | 0.474 / 0.633 | 0.573 / 0.638 |
| − CCL family | CCL telemetry down | 0.717 / 0.733 | 0.697 / 0.760 |
| − Network family | Network telemetry down | 0.723 / 0.867 | 0.636 / 0.735 |
| − GPU 50% | 4/9 GPU metrics down | 0.530 / 0.600 | 0.673 / 0.780 |
| − CCL 50% | 7/15 CCL metrics down | 0.769 / 0.800 | 0.769 / 0.824 |
| − Network 50% | 4/9 Network metrics down | 0.729 / 0.800 | 0.635 / 0.699 |

These results show that random rank missing is far more damaging than metric missing; structural pattern matters, since losing an entire TP group is better than losing the same number of random ranks because De-common naturally excludes intact groups; CCL telemetry loss causes the smallest performance drop, consistent with its redundant role across fault types; and partial metric loss (50%) causes only moderate degradation, demonstrating robustness to partial sensor outages.

### I.9 Noisy Training Analysis

We evaluate how training data quality affects generalization by contaminating the baseline training dataset with two types of noise: label noise (labeled anomalous samples) and measurement noise (Gaussian noise). Table 24 reports results.

Table 24: Noisy training analysis: Hit@3 (online / offline). Evaluation data is clean.

| Variant | baseline | AVG |
|---|---|---|
| default (clean) | **0.804 / 0.900** | **0.889 / 0.946** |
| 1% anomalous samples | 0.791 / 0.833 | 0.867 / 0.920 |
| 5% anomalous samples | 0.766 / 0.800 | 0.851 / 0.893 |
| 10% anomalous samples | 0.766 / 0.800 | 0.850 / 0.893 |
| Gaussian SNR 20dB | 0.766 / 0.800 | 0.851 / 0.893 |
| Gaussian SNR 10dB | 0.760 / 0.800 | 0.851 / 0.902 |
| Gaussian SNR 5dB | 0.757 / 0.800 | 0.851 / 0.906 |

These results show that both label noise and measurement noise degrade performance, but the effect saturates at higher noise levels; performance remains reasonable even with 10% anomalous samples or SNR 5dB, demonstrating robustness to imperfect training data; and training on clean normal data yields the best generalization, suggesting that practical deployment should use filtered normal windows for training.

## J USE OF LARGE LANGUAGE MODELS

In the process of preparing this paper, we employed Large Language Models (LLMs) to polish the writing. The assistance provided by LLMs was mainly focused on improving the clarity, coherence, and overall linguistic quality of the manuscript. Specifically, we input sections of the paper into the LLM and requested it to suggest rephrasings, correct grammar and spelling errors, and enhance the readability of the text. It is important to note that LLMs did not play a significant role in the research ideation. The core ideas, research questions, experimental designs, and methodological choices were independently conceived and developed by the human authors. The LLM was not involved in formulating the hypotheses, determining the research direction, or making decisions regarding data collection and analysis methods. We have reviewed and verified all LLM-assisted content and take full responsibility for the final content of this work.