SLADE: Detecting Dynamic Anomalies in Edge Streams without Labels via Self-Supervised Learning (Online Appendix)

C APPENDIX: BASELINE METHOD DETAILS

Among the four experiments mentioned earlier, the last type analysis experiment utilizes only unsupervised baselines. we provide individual baselines for the main experiments (RQ1, RQ2, RQ3, RQ5) and the type analysis experiment (RQ4).

C.1 Details of Baselines in Main Experiments

We compare the performances of our proposed method and the nine baseline methods in detecting dynamic anomalies in edge streams. The used baseline methods can be categorized as below:

- **Rule-based:** SedanSpot [8], MIDAS [1], F-FADE [4], and Anoedge-l [2]
- **Neural network-based:** JODIE [12], Dyrep [19], TGAT [20], TGN [16], and SAD [18]

For rule-based approaches, although the training phase is not necessary, some level of hyperparameter tuning can greatly increase the performance of them in our task. For a fair comparison, for each dataset, the optimal hyperparameter setting of each rule-based method is determined using the validation set. The selected hyperparameter setting is used to evaluate the corresponding model in the test set. There are several details regarding some of the used baseline methods:

- **Anoedge-l:** Bhatia et al. [2] propose several versions of Anoedge. Among them, we adopt **Anoedge-l** as our baseline method since it exhibits the best performance among them in our task.
- **MIDAS-R:** Bhatia et al. [1] propose several versions of MIDAS. Among them, we adopt **MIDAS-R** as our baseline method since it outperforms the others in our preliminary study.
- **F-FADE:** This method processes a stream every minute, while interactions in the used datasets occur at a much dense time interval (i.e., tens of interactions on original datasets occur within a minute). Due to this characteristic, F-FADE always underperforms all other used methods, specifically, cannot capture any anomalies. For F-FADE, we modify each dataset by adjusting the time units to large intervals.

In neural network-based methods, except for our proposed method, after selecting the hyperparameter settings based on its validation set, there are two strategies for utilizing a given dataset (train set and validation set) for training the final representation model with the selected hyperparameter settings:

- **S1. Using Both:** This indicates using both train and validation sets to train the final representation model with the selected hyperparameter settings. In this case, the model may utilize more information during training, while the model gets vulnerable to the overfitting issue.
- **S2. Training Set Only:** This indicates using only the train set to train the final representation model with the selected hyperparameter settings. In this case, the model utilizes the validation set only for early stopping, but we do not fully utilize the given dataset during training.

For each model, except for SAD (since the method inherently utilizes a validation dataset during model training), we utilize both strategies and report the higher test set evaluation performance between them.

C.2 Details of Baselines in Type Analysis Experiments

We compare the performances of our proposed method and four unsupervised baseline methods in type analysis. The unsupervised baseline methods are as follows: SedanSpot [8], MIDAS [1], F-FADE [4], and Anoedge-l [2]. In the Synthetic-hijack and Synthetic-New datasets, anomalies exist only in the test set, making it impossible to conduct a validation. Thus, for each model, we utilize the hyperparameter combinations the respective paper has reported. In cases where baseline settings in the previous work vary across datasets, we use the combination that has shown good performance in our main experiments (Section 6.2).

D APPENDIX: ADDITIONAL EXPERIMENTS

D.1 Performance Evaluation using Average Precision

In this subsection, we evaluate each model with the Average Precision (AP) metric regarding (RQ1) and (RQ4). Note that AP has different characteristics from that of the Area Under the ROC Curve (AUC). While AUC focuses on the overall distinguishability between two different classes, AP focuses on how well a model assigns higher scores for positive samples than for negative samples in terms of precision and recall.

We utilize the same hyperparameter settings from Section B.2 and B.3, and include a random guess model that randomly assigns anomaly scores between 0 and 1 for comparison. As shown in Table 1, SLADE-HP performs the best or second best in all real-world graph datasets, demonstrating the effectiveness of SLADE-HP in our task regarding AP also. Furthermore, as shown in Table 2, SLADE significantly outperforms other methods in both synthetic datasets.

D.2 Model Training Speed

In this subsection, we analyze the training time of SLADE. We first empirically demonstrate the training time of SLADE on the Reddit dataset with varying edge counts, where the detailed setting is the same as that described in Section 6.2 of the main paper. As shown in the left plot of Figure 1, the training runtime of SLADE is almost linear in the number of edges. Since the number of edges is much

Table 1: AP (in %) in the detection of dynamic anomaly nodes. The first method randomly assigns anomaly scores from 0 to 1, and the next four methods are rule-based models, and the others are based on representation learning. For each dataset, the best and the second-best performances are highlighted in boldface and underlined, respectively.

Method	Wikipedia	Reddit	Bitcoin-alpha	Bitcoin-OTC
Random Guess	0.22 ± 0.05	0.09 ± 0.01	8.96 ± 0.53	6.37 ± 0.28
SedanSpot [8]	0.66 ± 0.06	0.09 ± 0.00	12.99 ± 0.06	16.14 ± 0.01
MIDAS [1]	0.41 ± 0.02	0.12 ± 0.00	12.99 ± 0.06	16.14 ± 0.01
F-FADE [4]	0.18 ± 0.00	0.09 ± 0.00	6.34 ± 0.00	8.80 ± 0.00
Anoedge-l [2]	0.18 ± 0.01	0.09 ± 0.00	12.52 ± 0.86	17.03 ± 0.11
JODIE [12]	1.40 ± 0.02	0.25 ± 0.05	14.31 ± 0.40	15.79 ± 0.11
Dyrep [19]	1.53 ± 0.02	0.14 ± 0.01	8.41 ± 0.58	16.14 ± 1.74
TGAT [20]	1.48 ± 0.14	0.27 ± 0.12	12.19 ± 0.13	18.81 ± 0.63
TGN [16]	1.30 ± 0.04	0.19 ± 0.01	9.65 ± 0.40	19.87 ± 1.75
SAD [18]	**2.77 ± 0.91**	0.28 ± 0.06	12.55 ± 0.58	14.33 ± 0.88
SLADE	1.24 ± 0.14	0.23 ± 0.02	**15.40 ± 0.19**	20.21 ± 0.28
SLADE-HP	1.56 ± 0.14	**0.30 ± 0.05**	14.86 ± 0.08	**20.46 ± 0.06**

Table 2: AP (in %) in the detection of dynamic anomaly nodes in the two synthetic datasets. In both datasets, SLADE performs best compared to unsupervised methods in AP.

Method	Synthetic-Hijack	Synthetic-New
Random Guess	6.61 ± 0.07	6.63 ± 0.01
SedanSpot [8]	13.95 ± 1.53	15.17 ± 0.13
MIDAS [1]	17.96 ± 2.56	18.29 ± 3.18
F-FADE [4]	6.57 ± 0.00	6.58 ± 0.00
Anoedge-l [2]	8.20 ± 0.47	8.40 ± 0.63
SLADE	**69.69 ± 7.33**	**74.46 ± 6.16**

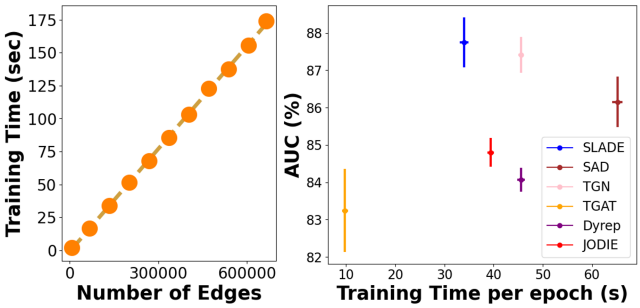


Figure 1: The left figure shows the increase in the training time per epoch of SLADE with respect to the number of edges in the Reddit dataset. The right figure shows the training time and AUC score (with standard deviations) in the Wikipedia dataset of the competing learning-based methods.

greater than that of nodes (67×), the overall scale is dominated by the number of edges.

Table 3: AUC (in %) in the detection of dynamic anomaly nodes with baselines based on link prediction self-supervised task.

Method	Wikipedia	Reddit
JODIE (S3) [12]	68.88 ± 1.54	58.17 ± 0.55
Dyrep (S3) [19]	58.78 ± 4.79	59.61 ± 1.22
TGAT (S3) [20]	71.40 ± 2.25	59.86 ± 1.48
TGN (S3) [16]	58.05 ± 3.48	56.77 ± 0.13
SLADE (S1,S2)	**87.75 ± 0.68**	**72.19 ± 0.60**

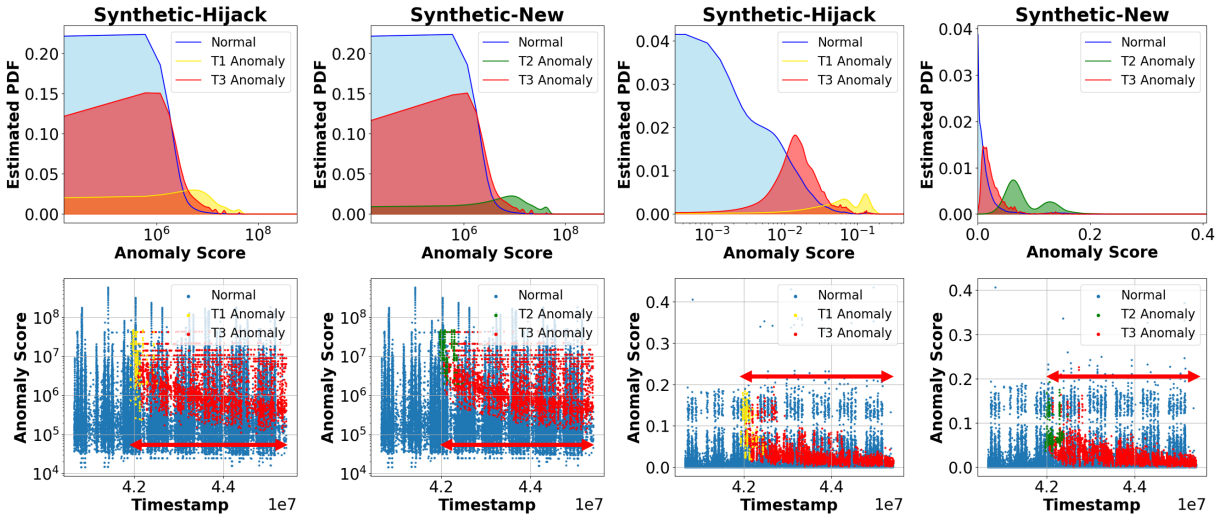
Furthermore, we compare the empirical training time of SLADE against that of other neural network-based methods, which require the training procedure. As shown in the right plot of Figure 1, in terms of the empirical training speed, SLADE is competitive compared to other methods. Specifically, among six neural network-based dynamic anomaly detection methods, SLADE exhibits the second-fastest training time, showing the best anomaly detection performance.

D.3 Type Analysis of Baselines

We conduct an additional analysis regarding two baseline methods (MIDAS and SedanSpot), which ranked second- and third-best position in our synthetic data experiments (**RQ4**). First, as shown in the first row of Figure 2 (a) and (b), we verify that the score distribution of the normal class and that of the consistent anomaly class (**T3**) largely overlaps in both models. Moreover, as shown in the second row of Figure 2(a) and (b), MIDAS and SedanSpot assign relatively high scores when anomalies of **T1** and **T2** occur, but not as distinctly as SLADE. These two results indicate that previous unsupervised anomaly detection baselines fail to detect anomalies if anomalies do not align with the targeted anomaly pattern, leading to high false negatives. These results emphasize the necessity of learning-based unsupervised anomaly detection models (SLADE), which autonomously learn normal patterns, and are capable of detecting a wide range of anomalies by finding cases that deviate from the learned normal patterns.

D.4 Self-supervised Task with Edge Prediction

In this subsection, we evaluate the performance of neural network baselines utilizing the link prediction-based self-supervised task (we denote this task as S3). Specifically, we consider the temporal edges that appear as positive samples and use randomly selected node pairs as negative samples. Based on these samples, the neural network models are trained using binary cross-entropy loss. We use the negative prediction probability score of the target node's involved edge as its anomaly score without fine-tuning a classifier with labels. In this case, anomaly detection can be performed based on the unexpectedness of the interaction in which the target node is involved. As shown in Table 3, SLADE outperforms other baseline methods, demonstrating that our proposed SSL tasks (S1 and S2) are more effective SSL strategies for detecting dynamic node anomalies than the link prediction task (S3). In predicting the dynamic states of nodes, we expect that the edge-wise self-supervised task has weaker expressive power to distinguish dynamic states compared



(a) Type Analysis of MIDAS

(b) Type Analysis of SedanSpot

Figure 2: For both (a) and (b), the above figures show the distribution of anomaly scores predicted by each baseline for each class in the two synthetic datasets (utilizing kernel density estimation). The below figures show the anomaly scores predicted by each baseline over time for each class in both datasets. We display the anomaly scores of MIDAS on a log scale.

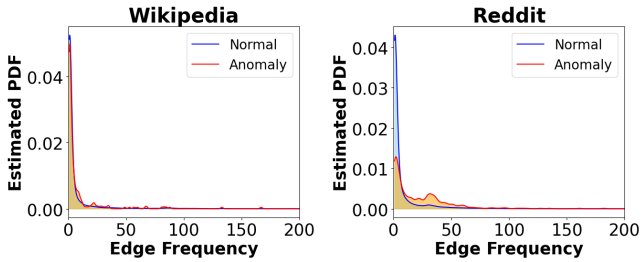


Figure 3: The above figures show the frequency distribution of anomalous edges that anomalous state nodes are involved in compared to normal edges in Wikipedia and Reddit datasets.

to node-wise self-supervised tasks. For example, if two users from different departments exchange an email in the email network, the unexpectedness of the interaction might be greater than the unexpectedness of an interaction performed by an anomalous user who randomly sends spam emails to various users.

Our edge frequency analysis in Fig 3 supports this claim. If the edges that the anomalous state nodes are involved in are indeed unexpected, we can expect these edges with the same node pair composition to occur much less frequently compared to other normal edges in the dataset. However, upon comparing the frequency distributions of normal edges and anomalous edges, we observe no clear distinction between the two frequency distributions in Wikipedia and Reddit datasets.

D.5 DTDG and Static Graph-based baselines

In this subsection, we evaluate the dynamic anomaly detection performance of SLADE with DTDG and static graph-based anomaly detection baselines using real-world datasets. We utilize two state-of-the-art DTDG-based approaches, StrGNN [3] and TADDY [14],

Table 4: AUC (in %) in the detection of dynamic anomaly nodes with DTDG-based baselines and a static graph-based baseline.

Method	Wikipedia	Reddit	Bitcoin-alpha	Bitcoin-OTC
DOMINANT [7]	67.15 ± 1.38	44.17 ± 0.53	54.21 ± 8.78	51.26 ± 0.01
STrGNN [3]	50.95 ± 0.46	53.30 ± 0.22	51.26 ± 0.01	47.85 ± 0.15
TADDY [14]	56.48 ± 1.41	54.54 ± 0.43	57.52 ± 1.41	64.06 ± 0.38
SLADE	**87.75 ± 0.68**	**72.19 ± 0.60**	**76.32 ± 0.28**	**75.80 ± 0.19**

and one static graph-based approach, DOMINANT [7]. For DTDG-based baselines, our CTDG datasets are converted into DTDG by creating graph snapshots in chronological order based on the number of edges. In this case, the graph snapshot size is set to 6,000 for Wikipedia and Reddit, 2,000 for Bitcoin-alpha, and 1,000 for Bitcoin-OTC. For a static graph-based baseline, all temporal edges that have appeared up to a certain point in the CTDG are projected onto a single static graph, ignoring the temporal information. The experimental setting is the same as our Section 6.2 (RQ1) experimental setting. We follow the hyperparameter settings provided by their official code with the exception of the number of epochs for StrGNN, which is reduced to 10 due to the long training time. Regarding DOMINANT, as our datasets have no node feature, we utilize identity vectors as node features. According to Table 4, SLADE outperforms other baselines. We hypothesize that SLADE's superior performance compared to DTDG-based and static-based methods is due to its more sophisticated utilization of temporal information.

D.6 Robustness to Anomalies in Training

In this subsection, we assess SLADE's robustness against the training dataset contamination (i.e., anomalies are included in the training dataset). To this end, we augment the training dataset with

Table 5: AUC (in %) in the detection of dynamic anomaly nodes against additional anomalies in the training set.

Dataset	SLADE	SAD [18]
WIKI+0 %	87.75 ± 0.68	86.15 ± 0.63
WIKI+1 %	87.48 ± 0.56	62.64 ± 13.55
WIKI+10 %	87.33 ± 0.59	17.04 ± 15.77
WIKI+20 %	87.05 ± 3.48	33.76 ± 6.86
REDDIT+0 %	72.19 ± 0.60	68.45 ± 1.27
REDDIT+1 %	72.10 ± 0.47	51.18 ± 1.28
REDDIT+10 %	71.69 ± 0.92	46.64 ± 0.95
REDDIT+20 %	69.97 ± 4.50	47.07 ± 1.51

increasing known abnormal edges, which are the edges involving nodes with the anomalous state in the training set, equivalent to X% of the training dataset size, denoted as dataset+X%. These anomalies are assumed to remain undetected and are considered normal during the training phase. We compare the performance of SLADE with that of the baseline anomaly detection method, SAD [18]. As shown in Table 5, SLADE outperforms the baseline method in every contamination ratio. Specifically, the performance drop of SLADE is at most 0.023, while that of the baseline method is at most 0.691. This result demonstrates the robustness of SLADE against the contamination of the training dataset.

D.7 SLADE with Label Supervision

In this subsection, we evaluate the variants of SLADE, which are first pre-trained using the proposed self-supervised tasks (S1, S2) as pretext tasks and then fine-tuned with labels. In this case, the memory updater and memory generator in SLADE are first pre-trained with our proposed self-supervised tasks (S1, S2), and then dynamic representations (specifically, concatenating current memory, previous memory, and generated memory) from SLADE and label information are utilized to fine-tune the MLP classifier. In the zero-shot setting, only normal labels are utilized. We denote SLADE-FT as SLADE with a fine-tuning and SLADE-ZS as SLADE with a zero-shot setting. We conduct experiments with the same settings with SLADE on the proposed real-world datasets. As shown in Table 6, surprisingly, our finding is that utilizing additional fine-tuning with label information is less effective than our method in most cases. We expect that our self-supervised tasks (S1, S2) already align well with dynamic anomaly detection, while minimal anomaly label supervision may introduce issues such as distribution shift and class imbalance.

D.8 SLADE in Large-scale Graphs

We generate a large-scale synthetic dataset by manipulating the TGBL-coin dataset [11]. In the TGBL-coin dataset, each node is an address, and each temporal edge is the transaction of funds from one node to another with time information. We resize the dataset and preprocess the dataset by utilizing the same anomaly injection method in Section 6.2 (RQ4), and it contains 499K nodes and 11M edges with a 1% anomaly ratio. We utilize 30% of the data for training and the remaining 70% for testing. We employ rule-based methods for baselines and follow the same setting in (RQ4). We utilize nearly the same hyperparameters configuration described in Section 6.1, but we train SLADE for 3 epochs. Our experimental result in Table 7 demonstrates that SLADE also performs well on large-scale datasets compared to other baselines.

Additionally, we empirically analyze the scalability of SLADE's training and inference time in large-scale scenarios by applying the same approach as in Section 6 (RQ2) and Section D.2 to the tgbl-coin dataset. Regarding inference time, we observe that SLADE maintains a constant inference time per edge, even on large-scale datasets. This reaffirms that SLADE has an inference time complexity independent of graph size, as mentioned in Section 6 (RQ2). On the other hand, unlike the observations in Section D.2, we find that the training time in a large-scale dataset increases at a rate faster than a linear trend with respect to the number of edges. As roughly analyzed in Section E.2, we hypothesize that this increase is due to the large number of nodes appearing in the training set for negative sampling, which impacts the training time.

Table 6: AUC (in %) in the detection of dynamic anomaly nodes with fine-tuning variant (SLADE-FT) and zero-shot variant (SLADE-ZS) of SLADE.

Method	Wikipedia	Reddit	Bitcoin-alpha	Bitcoin-OTC
SLADE-FT	86.06 ± 0.14	60.04 ± 0.55	**77.08 ± 0.17**	73.14 ± 0.61
SLADE-ZS	59.15 ± 0.09	53.34 ± 3.05	72.69 ± 0.02	65.00 ± 0.43
SLADE	**87.75 ± 0.68**	**72.19 ± 0.60**	76.32 ± 0.28	**75.80 ± 0.19**

Table 7: AUC (in %) in the detection of dynamic anomaly nodes with unsupervised baselines in the TGBL-coin dataset.

Method	TGBL-coin
MIDAS [1]	56.16 ± 0.58
SedanSpot [8]	77.14 ± 0.22
F-FADE [4]	59.33 ± 0.00
Anoedge-l [6]	39.11 ± 2.72
SLADE	**83.17 ± 1.34**

D.9 Learnable Time Encoding

In this subsection, we evaluate the performance difference based on whether the time encoding is trained using the TGAT [20] approach or fixed using the GraphMixer [5] approach. SLADE utilizes time encodings with fixed weights, as suggested by GraphMixer. We construct a variant of our model that utilizes learnable time encoding (SLADE-LT) from TGAT and compare it with SLADE with fixed time coding (SLADE-FT). According to Table 8, we empirically verify that SLADE-FT encoding performs better. We hypothesize that this is due to the distribution shift issue. First, our dynamic node anomaly detection task can be considered as a (temporal) extrapolation task. Thus, test data may follow a different time-interval distribution from that of training data. Consequently, learnable time encoding, which is likely to overfit the train data, may exhibit

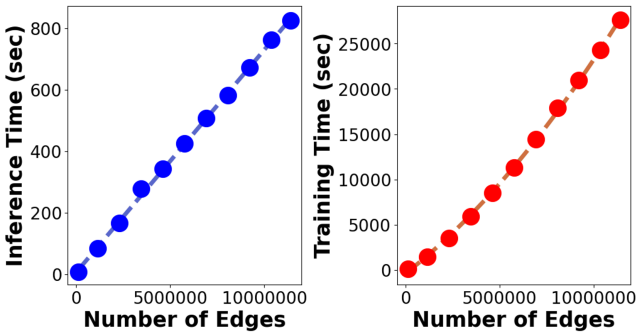


Figure 4: The left figure shows the increase in the inference time and the right figure shows the increase in the training time per epoch of SLADE with respect to the number of edges in a large-scale scenario.

Table 8: AUC (in %) in the detection of dynamic anomaly nodes with a learnable time encoding variant (SLADE-LT) of SLADE and SLADE (SLADE-FT).

Method	Wikipedia	Reddit	Bitcoin-alpha	Bitcoin-OTC
SLADE-LT	86.78 ± 0.66	65.28 ± 0.68	71.11 ± 0.83	73.98 ± 0.35
SLADE-FT	**87.75 ± 0.68**	**72.19 ± 0.60**	76.32 ± 0.28	**75.80 ± 0.19**

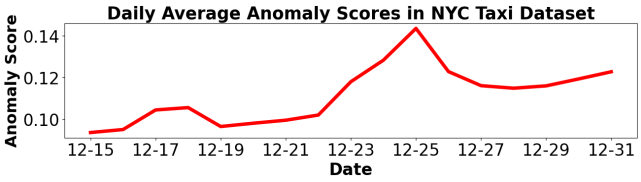


Figure 5: Daily average anomaly scores of NYC taxi trips in December 2023.

poor generalization to the test data, leading to suboptimal dynamic anomaly detection performance.

D.10 Transportation Network Dataset

We evaluate the performance of SLADE in a transportation network, which belongs to a different domain from the datasets used in the main experiments. We utilize the New York (NYC) Taxi dataset (Yellow Taxi Trip Records from TLC Record data [1] in December 2023) and split the data into training and test sets chronologically, with each set comprising 50% of the data. In the transportation network, each node represents a NYC taxi zone location. We preprocess temporal edges to represent NYC taxi trips, with the source node as the departure taxi zone, the destination node as the arrival taxi zone, and the time as the arrival time. Since the dataset does not contain labels indicating anomalies, we aim to analyze the trend between the daily average anomaly scores of NYC taxi trips and actual daily events that occurred. As shown in Figure 5, SLADE assigns higher daily average anomaly scores for 12/25 (Christmas) and 12/31 (the last day of the year), which exhibit different transportation patterns than other days.

[1] https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page

Table 9: AUC (in %) in the detection of dynamic anomaly nodes in Synthetic-Hide and Synthetic-New. In both datasets, SLADE performs best compared to unsupervised methods in AUC.

Method	Synthetic-Hide	Synthetic-New
SedanSpot [8]	75.18 ± 2.12	78.05 ± 1.68
MIDAS [1]	81.78 ± 0.09	82.63 ± 0.07
F-FADE [4]	41.54 ± 0.00	48.73 ± 0.00
Anoedge-l [2]	60.57 ± 2.11	61.86 ± 2.41
SLADE	**96.11 ± 3.31**	**98.38 ± 1.09**

D.11 Anomalies with Camouflage

In this subsection, we assess the robustness of SLADE in dealing with camouflaging attackers who alternate between anomalous and normal behaviors. To reflect this case, we conduct a dynamic anomaly detection task on our new synthetic dataset: Synthetic-Hide, where camouflaged anomalous nodes exist. Specifically, unlike Synthetic-Hijack, we first select anomalous node candidates from only normal nodes that continuously appear in the test set. Subsequently, we follow the same previous anomaly injection method with the same 1% anomaly injection in Section A. Through this method, anomalous nodes are considered to perform normal interactions with similar neighbors for hiding and perform anomalous interactions with random neighbors for attack. We evaluate the performance of SLADE on Synthetic-Hide against unsupervised baselines using the same settings in Section B.2. As shown in Table 9, SLADE still outperforms other competing unsupervised methods by a significant gap. However, compared to the case without camouflaged actions (Synthetic-New dataset), we demonstrate a slight decrease in the SLADE's performance.

In the situation where an attacker consistently engages in abnormal interactions that mimic our proposed normal patterns (A1, A2), there is indeed a possibility of deceiving SLADE. Nonetheless, as discussed in Section 4, such scenarios require the attacker to consistently engage in these abnormal interactions with similar targets (i.e., victims) over an extended period. These repeated interactions impose significant costs on attackers. Moreover, targeting similar victims repeatedly may increase the likelihood of detection, such as through reports from the victims. As a result, anomalous nodes are likely to exhibit interaction patterns opposite to our proposed normal patterns.

D.12 Dynamic Heterogeneous Graphs

SLADE performs anomaly detection on dynamic homogeneous graphs in the main experiments. However, nodes and edges in specific real-world networks can have different types, forming dynamic heterogeneous graphs. In this case, SLADE can assume that the same type exhibits similar normal interaction patterns (A1, A2), and specific normal interaction patterns vary depending on the types. Thus, we can perform anomaly detection on dynamic heterogeneous graphs for each type separately by creating distinct memory updater and generator models as in SLADE. Despite this, we encounter challenges due to the absence of suitable datasets and baselines for anomaly detection in dynamic heterogeneous graphs.

Table 10: AUC (in %) in the detection of dynamic anomaly nodes with SLADE and SLADE-Hetero on dynamic bipartite graphs.

Method	Wikipedia	Reddit
SLADE	87.75 ± 0.68	72.19 ± 0.60
SLADE-Hetero	**87.96 ± 0.41**	**72.65 ± 0.51**

Recently, [13] has proposed a DTDG-based approach for anomalous edge detection in dynamic heterogeneous graphs and provided related datasets, but we have not found any datasets specifically tailored for anomalous node detection in edge streams.

Consequently, to evaluate the feasibility of our proposed approach, we utilize dynamic bipartite graphs instead of dynamic heterogeneous graphs. Although they cannot replace dynamic heterogeneous graphs, we can consider dynamic bipartite graphs as a kind of simple dynamic heterogeneous graph with two types of nodes. Among our datasets, Wikipedia and Reddit are dynamic bipartite graphs between users and items. We design SLADE-Hetero for a dynamic bipartite graph by utilizing different memory updaters and generators for each node type to train distinct normal patterns of each type. Then, we compare SLADE-Hetero with the original SLADE, and most of its settings are identical to SLADE's settings. According to the experimental results in Table 10, SLADE-Hetero demonstrates slightly improved performance compared to the original SLADE. Nonetheless, SLADE's performance on actual dynamic heterogeneous graphs remains uncertain. As further research and datasets relevant to dynamic anomalous node detection in dynamic heterogeneous graphs emerge, we anticipate this could provide a promising direction for the future development of SLADE.

E APPENDIX: ADDITIONAL DISCUSSION AND ANALYSIS

E.1 Discussion on Normal Pattern Assumptions

As mentioned in Section 4, the intuition of our normal pattern assumptions is that normal nodes engage in repetitive and sustained interaction with (a limited number of) similar neighbors over time, resulting in structurally and temporally similar interaction patterns over time. To support this, we explore social and psychological research to explain the underlying reasons. In [10], the authors explain that repeated interactions raise trust between members, reducing the need for contractual safeguards in subsequent alliances. Therefore, the economic logic of repeated interactions with similar neighbors is the cost-saving benefits of trust. In [15], the authors mention that there are constraints on the number of relationships the ego can maintain. These constraints are expected to be cognitive (the size of an individual's support clique is correlated with the number of levels of intentionality that an individual can process) and time budgeting. Additionally, [9] explains that repeated and sustained interactions are expected as evidence of a relationship. As a result, we can infer that normal nodes establish relationships with a limited number of neighbors and consistently engage in repetitive interactions with them, which supports our normal pattern assumptions.

E.2 Training Complexity Analysis

We roughly analyze the training time complexity of SLADE. Specifically, we examine the forward cost of (a) temporal contrast loss and (b) memory generation loss for a temporal edge (v_i, v_j, t) in the training set.

Temporal Contrast Loss: Given a training edge, SLADE updates the memory vector of each node in the edge using GRU. The total time complexity is dominated by that of GRU, which is $O({d_s}^2 + d_s d_m)$ [17], where d_s and d_m indicate the dimensions of memory vectors and messages respectively. Then, SLADE computes the similarities (a) between the current memory and previous memory and (b) between current memory and memories of negative samples for each node in the training edge. It takes $O(\mathcal{V}_n {d_s}^2)$, where $\mathcal{V}_n$ is the number of negative samples.

Memory Generation Loss: Given a training edge, SLADE generates memory vectors for nodes in the training edge using TGAT with the time complexity of $O(k{d_s}^2)$ [21], as mentioned in Section 5.2. Then, SLADE computes the similarities (a) between generated memory and current memory vectors and (b) between generated memory and memories of negative samples for each node in the training set, taking $O(\mathcal{V}_n {d_s}^2)$ time.

As a result, the time complexity of computing both losses for a training edge is $O((k + \mathcal{V}_n){d_s}^2 + d_s d_m)$. In the worst case, using $\mathcal{V}(t^+)$ as negative samples can result in the time complexity of $O((k + \mathcal{V}_t){d_s}^2 + d_s d_m)$, where $\mathcal{V}_t$ is the number of nodes in the training set. Therefore, with our current negative sampling method, the training time increases as the number of nodes in the training set increases. In our experiments in Section D.2, we observe that the number of nodes in the training set does not significantly affect training time on the Reddit dataset. However, as shown in Section D.8, the number of nodes in the training set affects training scalability on large-scale datasets. It is necessary to explore methods to limit the number of negative samples effectively.

REFERENCES

[1] Siddharth Bhatia, Bryan Hooi, Minji Yoon, Kijung Shin, and Christos Faloutsos. 2020. Midas: Microcluster-based detector of anomalies in edge streams. In *AAAI*.

[2] Siddharth Bhatia, Mohit Wadhwa, Kenji Kawaguchi, Neil Shah, Philip S Yu, and Bryan Hooi. 2023. Sketch-Based Anomaly Detection in Streaming Graphs. In *KDD*.

[3] Lei Cai, Zhengzhang Chen, Chen Luo, Jiaping Gui, Jingchao Ni, Ding Li, and Haifeng Chen. 2021. Structural temporal graph neural networks for anomaly detection in dynamic graphs. In *CIKM*.

[4] Yen-Yu Chang, Pan Li, Rok Sosic, MH Afifi, Marco Schweighauser, and Jure Leskovec. 2021. F-fade: Frequency factorization for anomaly detection in edge streams. In *WSDM*.

[5] Weilin Cong, Si Zhang, Jian Kang, Baichuan Yuan, Hao Wu, Xin Zhou, Hanghang Tong, and Mehrdad Mahdavi. 2022. Do We Really Need Complicated Model Architectures For Temporal Networks?. In *ICLR*.

[6] Graham Cormode and Shan Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* 55, 1 (2005), 58–75.

[7] Kaize Ding, Jundong Li, Rohit Bhanushali, and Huan Liu. 2019. Deep anomaly detection on attributed networks. In *SDM*.

[8] Dhivya Eswaran and Christos Faloutsos. 2018. Sedanspot: Detecting anomalies in edge streams. In *ICDM*.

[9] Ellen Garbarino and Mark S Johnson. 1999. The different roles of satisfaction, trust, and commitment in customer relationships. *Journal of marketing* 63, 2 (1999), 70–87.

[10] Anthony Goerzen. 2007. Alliance networks and firm performance: The impact of repeated partnerships. *Strategic management journal* 28, 5 (2007), 487–509.

[11] Shenyang Huang, Farimah Poursafaei, Jacob Danovitch, Matthias Fey, Weihua Hu, Emanuele Rossi, Jure Leskovec, Michael Bronstein, Guillaume Rabusseau, and Reihaneh Rabbany. 2024. Temporal graph benchmark for machine learning

on temporal graphs. *Advances in Neural Information Processing Systems* 36 (2024).
[12] Srijan Kumar, Xikun Zhang, and Jure Leskovec. 2019. Predicting dynamic embedding trajectory in temporal interaction networks. In *KDD*.
[13] Yilin Li, Jiaqi Zhu, Congcong Zhang, Yi Yang, Jiawen Zhang, Ying Qiao, and Hongan Wang. 2023. THGNN: An Embedding-based Model for Anomaly Detection in Dynamic Heterogeneous Social Networks. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*. 1368–1378.
[14] Yixin Liu, Shirui Pan, Yu Guang Wang, Fei Xiong, Liang Wang, Qingfeng Chen, and Vincent CS Lee. 2021. Anomaly detection in dynamic graphs via transformer. *IEEE Transactions on Knowledge and Data Engineering* (2021).
[15] Sam GB Roberts, Robin IM Dunbar, Thomas V Pollet, and Toon Kuppens. 2009. Exploring variation in active network size: Constraints and ego characteristics. *Social Networks* 31, 2 (2009), 138–146.
[16] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. 2020. Temporal Graph Networks for Deep Learning on Dynamic Graphs. In *ICML 2020 Workshop on Graph Representation Learning*.
[17] Michael Rotman and Lior Wolf. 2021. Shuffling recurrent neural networks. In *AAAI*.
[18] Sheng Tian, Jihai Dong, Jintang Li, Wenlong Zhao, Xiaolong Xu, Bowen Song, Changhua Meng, Tianyi Zhang, Liang Chen, et al. 2023. SAD: Semi-Supervised Anomaly Detection on Dynamic Graphs. In *IJCAI*.
[19] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2019. Dyrep: Learning representations over dynamic graphs. In *ICLR*.
[20] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. 2020. Inductive representation learning on temporal graphs. In *ICLR*.
[21] Tongya Zheng, Xinchao Wang, Zunlei Feng, Jie Song, Yunzhi Hao, Mingli Song, Xingen Wang, Xinyu Wang, and Chun Chen. 2023. Temporal Aggregation and Propagation Graph Neural Networks for Dynamic Representation. *IEEE Transactions on Knowledge and Data Engineering* (2023).