APPENDIX

A. Real-world Datasets

We use 7 real-world datasets for a node property prediction task; three datasets (Wikipedia, Reddit, and MOOC [1]) [1] for dynamic anomaly detection and two datasets (Email-EU [2] [2], GDELT [3] [3]) for dynamic node classification, and two datasets (TGBN-trade and TGBN-genre [4]) [4] for node affinity prediction. Basic descriptive statistics of each dataset are provided in Table **??** of Section **??**. Below, we provide a detailed description of each dataset.

Datasets for Dynamic Anomaly Detection: The Wikipedia dataset records edits made by users on Wikipedia pages. The Reddit dataset consists of posts made by users on subreddits. In the Wikipedia and Reddit datasets, if an administrator bans a user at a specific time, the user is assigned an abnormal property after that time; otherwise, it has a normal property. The MOOC dataset consists of user activities on MOOC online course platforms. In the MOOC dataset, if a user drops the course at a particular time point, it is marked with an abnormal property after that time point; otherwise, it has a normal property.

Datasets for Dynamic Node Classification: The Email-EU dataset consists of emails between users in a European research institution. The property of a user at a specific time indicates the user's department ID at the institute, which remains constant over time in this dataset. The GDELT dataset is an event network that records events worldwide between actors from GDELT 2.0 [5]. Following the [6], the GDELT dataset is sub-sampled for single-machine training. The location (country) of an actor at a specific time is considered a property, which is dynamic over time. Unlike other datasets, the GDELT dataset has node features that are multi-hot vectors representing the CAMEO codes attached to the actors.

Datasets for Node Affinity Prediction: The TGBN-trade dataset records international agriculture trading between nations of the UN. Each node is a nation, and the edge weight represents the sum of the agricultural product value traded between the two nations. The property of a nation is the proportion of agricultural trade values to other nations during the next year. The TGBN-genre is a weighted interaction network between users and the music genres of songs they listen to. The edge weights represent the percentage of a song's association with a particular genre. The property of a user is the user's preference of music genres over the next week. In such cases, previous studies have incorporated edge weight as an edge feature because the models can not explicitly handle edge weight. Consequently, edge weight is utilized as an edge feature for existing models in our experiments. Note that these property labels from all datasets are inherently given in the original datasets.

[1] http://snap.stanford.edu/jodie/#datasets

[2] https://snap.stanford.edu/data

[3] https://s3.us-west-2.amazonaws.com/dgl-data/dataset/tgl/GDELT

[4] https://tgb.complexdatalab.com/docs/nodeprop/

B. Details of Synthetic Datasets

In this section, we provide more details of synthetic datasets. The synthetic dataset consists of 1,000 nodes, each belonging to one of 10 classes, which are static over time, with 100 nodes per class. [5] A total of 20,000 temporal edges occur over 0 to 1,000,000 seconds, and the goal is to predict the class of each source node as the node property. We utilize the same chronological 10/10/80% split for training, validation, and test sets. In this case, labels of source nodes within temporal edges within the training set can be utilized for training.

As mentioned in Section V, to induce distribution shifts, we manipulate the frequency of node appearances between the training and test sets according to the class using a shift intensity factor p, which ranges from 50 to 100. Specifically, we first sample class-known node sets for each class, consisting of nodes that can appear as source nodes in the training set. we sample p nodes as class-known nodes per class for group 1 classes and sample (100-p) nodes as class-known nodes per class for group 2 classes. The nodes that are not sampled are assigned to the class-unknown sets for each class.

When generating temporal edges in the training set, source nodes are selected from the class-known node sets of group 1 classes with a probability of $p\%$ or from the class-known node sets of group 2 classes with a probability of $(100-p)\%$. The destination nodes are selected from the same class as the source nodes with a probability of 90%, or they are randomly selected with a probability of 10%.

In contrast, temporal edges in the validation and test set are generated in reverse. When generating temporal edges in the test set, source nodes are selected from the class-unknown node sets of group 1 classes with a probability of $(100-p)\%$ or from the class-unknown node sets of group 2 classes with a probability of $p\%$. The selection process for destination nodes is the same as for the training set. The timestamps are uniformly distributed across the entire range of temporal edges for the training, validation, and test sets, and selected randomly within their respective ranges.

C. Distribution Shifts Analysis

Distribution Shifts in Real-world Datasets: In this section, we aim to analyze distribution shifts across 6 real-world datasets (except the Reddit dataset), similar to the analysis conducted in Section **??**.

- **Positional Distribution Shifts:** To analyze positional shifts in real-world datasets, we first divide the nodes into 20 groups [6] based on their appearance time in the CTDG. Node embeddings are then generated by applying node2vec to the entire graph. Finally, we visualize the average embedding of each group using a t-SNE plot. As shown in Figure 1, we can observe that the distribution of embeddings representing the positions of nodes appearing over time varies in most datasets.

[5] Each node belongs to a class determined by the quotient obtained by dividing its node ID by 100.

[6] In TGBN-trade, we utilize the 10 groups due to the small number of nodes.

- **Structural Distribution Shifts:** According to Figure 2, we can observe that the average degree of nodes increases over time across all datasets, indicating a structural distribution shift. In the case of the TGBN dataset, the average node degree is quite large over time, which means that if the number of sampled neighbors is too small, it may not adequately capture these structural shifts.
- **Property Distribution Shifts:** As shown in Figure 3, the occurrence rate of node properties varies over time, showing a property distribution shift. Therefore, to effectively address these shifts, it is essential to identify node characteristics that have a stable relationship with node properties over time.

Distribution Shifts in Synthetic Datasets: In this section, we aim to analyze distribution shifts in the synthetic datasets (Synthetic-50, Synthetic-70, and Synthetic-90) with the artificial positional distribution shifts. We adopt a similar approach for real-world networks to analyze positional shifts in the synthetic datasets without averaging the embeddings. As shown in Figure 4, We observe that positional shifts are more distinct in the datasets with higher shift intensity factor p.

D. Experiments Environment

We conduct all experiments with NVIDIA RTX 3090 Ti GPUs (24GB VRAM), 256GB of RAM, and two Intel Xeon Silver 4210R Processors.

E. Details of Evaluation Metrics and Baselines

Details of Evaluation Metric: We introduce the evaluation metrics for each subtask in this section as follows:

- **AUC:** In anomaly detection, AUC is a performance metric commonly used to evaluate the performance of a model. It refers to the area under the Reciever Operating Characteristic (ROC) curve, which plots the true positive rate (TPR) against the false positive rate (FAR) at various classification thresholds. A higher AUC indicates that the model effectively detects anomalies across most thresholds, reflecting strong performance.
- **F1 Score:** In node classification, F1 Score is a general performance measure to evaluate a model's performance by balancing precision and recall. A high F1 Score indicates overall strong performance in correctly identifying and classifying nodes within the graphs.
- **NDCG@k:** In the recommender system, NDCG@k is a commonly used metric to evaluate the quality of ranking models. It refers to the Normalized Discounted Cumulative Gain metric that considers the relative order of elements. It measures how closely the top-k predicted items align with users' actual item preference rankings. A high NDCG@k indicates that the predictions closely match the users' actual item preference rankings, reflecting strong recommendation performance.

Details of Baselines: As mentioned in Section V-A of the main paper, we tune most hyperparameters of each baseline method by conducting a full grid search on the validation set of each dataset. For other hyperparameters, we strictly follow the settings provided in their official code. We train all models with the Adam [8] optimizer with weight decay as 10^{-4}, which is the same as in SPLASH. The selected hyperparameter combination of each model is reported in Table I. The hyperparameter search space of baselines is as follows:

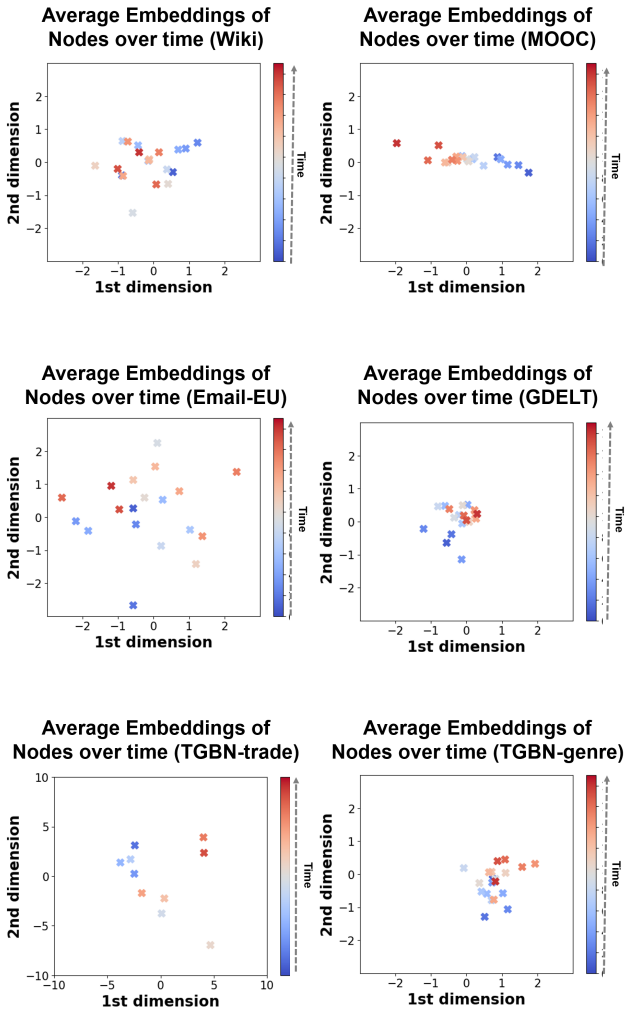

Fig. 1. Examples of positional distribution shifts in real-world network datasets. Nodes are grouped based on their appearance time, and the node embeddings generated by node2vec [7] using the entire graph are averaged within each group. These averaged embeddings are visualized using t-SNE.

- **JODIE**: learning rate between (10^{-3}, 10^{-4}), batch size among (200, 600, 1000)
- **DySAT**: learning rate between (10^{-3}, 10^{-4}), batch size among (200, 600, 1000), and degree among (10, 100, 500)
- **TGAT**: learning rate between (10^{-3}, 10^{-4}), batch size among (200, 600, 1000), and degree among (10, 100, 500)
- **TGN**: learning rate between (10^{-3}, 10^{-4}), batch size among (200, 600, 1000), and degree among (10, 100, 500)
- **Graph-Mixer**: learning rate between (10^{-3}, 10^{-4}), batch size among (200, 600, 1000), and degree among (10, 100, 500)
- **DyGFormer**: learning rate between (10^{-3}, 10^{-4}), batch

TABLE I

HYPERPARAMETER SETTINGS FOR ALL BASELINE METHODS OF MAIN EXPERIMENTS ON EACH DATASET THROUGH VALIDATION. WE CONDUCT A PERFORMANCE EVALUATION ON THE TEST SET USING THESE SETTINGS AND THOSE RECOMMENDED BY THE AUTHORS AND REPORT THE BEST-PERFORMING OUTCOME.

	Dynamic Anomaly Detection			**Dynamic Node Classification**		**Node Affinity Prediction**	
	Reddit	**Wiki**	**MOOC**	**Email-EU**	**GDELT**	**TGBN-trade**	**TGBN-genre**
JODIE (learning rate, batch size)	$(10^{-3}, 200)$	$(10^{-4}, 600)$	$(10^{-3}, 200)$	$(10^{-4}, 600)$	$(10^{-4}, 600)$	$(10^{-4}, 600)$	$(10^{-4}, 600)$
DySAT (learning rate, batch size, degree)	$(10^{-4}, 600, 10)$	$(10^{-4}, 200, 500)$	$(10^{-4}, 1000, 500)$	$(10^{-4}, 600, 10)$	$(10^{-3}, 200, 500)$	$(10^{-3}, 200, 10)$	$(10^{-4}, 600, 10)$
TGAT (learning rate, batch size, degree)	$(10^{-4}, 600, 10)$	$(10^{-3}, 200, 500)$	$(10^{-4}, 600, 500)$	$(10^{-3}, 600, 500)$	$(10^{-3}, 200, 500)$	$(10^{-3}, 1000, 10)$	$(10^{-4}, 600, 100)$
TGN (learning rate, batch size, degree)	$(10^{-4}, 200, 10)$	$(10^{-4}, 200, 100)$	$(10^{-3}, 600, 10)$	$(10^{-3}, 200, 10)$	$(10^{-4}, 600, 10)$	$(10^{-3}, 1000, 500)$	$(10^{-4}, 600, 10)$
GraphMixer (learning rate, batch size, degree)	$(10^{-4}, 200, 30)$	$(10^{-3}, 200, 100)$	$(10^{-4}, 200, 30)$	$(10^{-4}, 200, 30)$	$(10^{-3}, 200, 100)$	$(10^{-4}, 200, 30)$	$(10^{-4}, 200, 30)$
DyGFormer (learning rate, batch size, seq length)	$(10^{-3}, 1000, 32)$	$(10^{-4}, 200, 32)$	$(10^{-4}, 600, 32)$	$(10^{-3}, 600, 128)$	$(10^{-4}, 1000, 32)$	$(10^{-3}, 200, 64)$	$(10^{-3}, 200, 128)$
JODIE+RF (learning rate, batch size)	$(10^{-4}, 600)$	$(10^{-4}, 200)$	$(10^{-4}, 600)$	$(10^{-3}, 200)$	$(10^{-3}, 200)$	$(10^{-3}, 1000)$	$(10^{-4}, 600)$
DySAT+RF (learning rate, batch size, degree)	$(10^{-3}, 1000, 10)$	$(10^{-4}, 600, 10)$	$(10^{-3}, 1000, 10)$	$(10^{-3}, 600, 10)$	$(10^{-3}, 200, 500)$	$(10^{-3}, 1000, 500)$	$(10^{-3}, 600, 10)$
TGAT+RF (learning rate, batch size, degree)	$(10^{-4}, 600, 10)$	$(10^{-3}, 200, 10)$	$(10^{-3}, 1000, 10)$	$(10^{-3}, 200, 10)$	$(10^{-3}, 600, 500)$	$(10^{-3}, 600, 500)$	$(10^{-3}, 600, 100)$
TGN+RF (learning rate, batch size, degree)	$(10^{-3}, 600, 100)$	$(10^{-4}, 200, 500)$	$(10^{-3}, 200, 500)$	$(10^{-3}, 200, 500)$	$(10^{-3}, 600, 500)$	$(10^{-3}, 1000, 500)$	$(10^{-3}, 200, 500)$
GraphMixer+RF (learning rate, batch size, degree)	$(10^{-3}, 1000, 100)$	$(10^{-4}, 200, 30)$	$(10^{-4}, 200, 30)$	$(10^{-3}, 200, 500)$	$(10^{-3}, 200, 500)$	$(10^{-3}, 1000, 500)$	$(10^{-4}, 200, 30)$
DyGFormer+RF (learning rate, batch size, seq length)	$(10^{-3}, 600, 32)$	$(10^{-4}, 600, 64)$	$(10^{-4}, 200, 32)$	$(10^{-3}, 200, 32)$	$(10^{-3}, 1000, 32)$	$(10^{-3}, 200, 32)$	$(10^{-3}, 1000, 128)$

size among (200, 600, 1000), and maximum input sequence among (32, 64, 128)

- **JODIE+RF**: learning rate between (10^{-3}, 10^{-4}), batch size among (200, 600, 1000)
- **DySAT+RF**: learning rate between (10^{-3}, 10^{-4}), batch size among (200, 600, 1000), and degree among (10, 100, 500)
- **TGAT+RF**: learning rate between (10^{-3}, 10^{-4}), batch size among (200, 600, 1000), and degree among (10, 100, 500)
- **TGN+RF**: learning rate between (10^{-3}, 10^{-4}), batch size among (200, 600, 1000), and degree among (10, 100, 500)
- **GraphMixer+RF**: learning rate between (10^{-3}, 10^{-4}), batch size among (200, 600, 1000), and degree among (10, 100, 500)
- **DyGFormer+RF**: learning rate between (10^{-3}, 10^{-4}), batch size among (200, 600, 1000), and maximum input sequence among (32, 64, 128)

As we set a larger maximum node degree compared to previous studies, an out-of-memory (OOM) issue occurs when increasing the number of layers. Additionally, many studies [6], [9] report that increasing the number of layers is ineffective. Therefore, during tuning, we fix the number of layers to 1.

All baselines, like SPLASH, first create dynamic node representations and then use a classifier to predict node properties. The classifier and the entire TGNN encoder are trained using label supervision. [7] TGNN baselines, except DyGFormer, are implemented using the TGL [3] framework, while DyGFormer is implemented using the DyGLib [10] framework. Since DyGFormer focuses on more sophisticated edge encoding, it generates the edge encoding and then extracts dynamic node representations. If no edge exists at the time of a node property query, the edge encoding from the most recent interaction before the query time is used.

[7]Previous studies explored pre-training the TGNN encoder using a link prediction task and freezing it while only training the classifier with label supervision. However, we found this approach less effective under significant distribution shifts, so we trained the TGNN encoder and the classifier together.

F. Detailed Hyperparameters of SPLASH

We train SPLASH using the Adam optimizer, with a learning rate of 10^{-3} and a weight decay of 10^{-4}. We fix the dropout probability to 0.2. In addition, we fix the scaling scalar of temporal encoding (Eq (17) in the main paper) to $\alpha = 10, \beta = 10$, and the dimension of a time encoding to 100. We also set the dimension of the dynamic representation to 100, which is equivalent to the dimension of node features. In the case of the weight of skip connection λ_s, we set 0 for the MOOC dataset and 1 for other datasets through validation. In the positional node feature augmentation process, we utilize Node2Vec with walk length as 10, number of walks as 80, and return parameter p as 10 to generate node features for training nodes in the transductive approach. We utilize the TGL [3] framework for the implementation of SPLASH.

G. Performance of Baselines using Selected Augmentation Node Features

In this subsection, we evaluate each model with the selected augmented node features regarding RQ1 on four real-world datasets (Wiki, Reddit, Email-EU, and TGBN-trade). All TGNN baselines use the selected augmented node feature with minimal empirical risk for each dataset, according to the results in Table V of the main paper.

As shown in Table II, utilizing the selected augmented node feature improves model performance in many cases. Specifically, for the Email-EU dataset, all baselines with selected augmented node features exhibit significant performance improvements. In the cases where the selected augmented node

TABLE II

PERFORMANCE (IN %) IN THE PREDICTION OF NODE PROPERTIES. ALL DATASETS ARE SPLIT INTO 10/10/80% FOR THE TRAINING, VALIDATION, AND TEST SETS. THE FIRST SIX METHODS ARE TGNN-BASED MODELS, THE NEXT SIX METHODS ARE TGNN-BASED MODELS UTILIZING RANDOM FEATURES AS NODE FEATURES, AND THE NEXT SIX METHODS ARE TGNN-BASED MODELS UTILIZING SELECTED NODE FEATURES. FOR EACH DATASET, THE BEST AND THE SECOND-BEST PERFORMANCES ARE HIGHLIGHTED IN **BOLDFACE** AND UNDERLINED, RESPECTIVELY. IN EVERY CASE, SPLASH PERFORMS BEST COMPARED TO OTHER BASELINES.

	Dynamic Anomaly Detection (AUC)		**Dynamic Node Classification (F1 Score)**	**Node Affinity Prediction (NDCG@10)**
	Reddit	**Wiki**	**Email-EU**	**TGBN-trade**
JODIE [1]	55.22 ± 1.03	80.60 ± 0.63	10.54 ± 0.31	35.19 ± 0.35
DySAT [11]	56.90 ± 2.34	80.58 ± 0.54	11.61 ± 0.00	35.09 ± 0.39
TGAT [12]	61.00 ± 0.88	79.06 ± 1.33	9.82 ± 1.31	35.11 ± 0.32
TGN [9]	59.26 ± 0.41	80.38 ± 1.45	11.22 ± 2.66	34.70 ± 0.03
GraphMixer [6]	62.98 ± 1.47	83.77 ± 0.82	11.93 ± 2.19	26.45 ± 1.26
DyGFormer [10]	63.71 ± 0.67	84.28 ± 0.59	14.74 ± 2.47	34.23 ± 0.18
JODIE+RF	48.72 ± 0.56	79.76 ± 1.57	93.08 ± 0.15	44.02 ± 0.71
DySAT+RF	53.47 ± 1.51	71.30 ± 0.89	93.15 ± 0.07	48.60 ± 0.01
TGAT+RF	56.13 ± 4.07	72.62 ± 2.01	93.25 ± 0.30	48.74 ± 0.11
TGN+RF	53.85 ± 2.12	78.84 ± 2.32	92.57 ± 0.45	46.83 ± 0.56
GraphMixer+RF	51.39 ± 2.51	69.90 ± 2.52	83.58 ± 2.79	26.67 ± 0.40
DyGFormer+RF	64.06 ± 0.34	84.48 ± 0.75	65.65 ± 3.53	35.21 ± 1.36
JODIE+SF	48.16 ± 1.02	81.32 ± 0.42	97.18 ± 0.18	44.98 ± 0.59
DySAT+SF	55.57 ± 1.88	82.42 ± 0.31	97.94 ± 0.03	49.40 ± 0.23
TGAT+SF	70.85 ± 1.47	78.98 ± 1.86	97.70 ± 0.43	49.03 ± 0.27
TGN+SF	70.38 ± 1.61	80.03 ± 1.23	97.54 ± 0.08	46.76 ± 0.47
GraphMixer+SF	64.09 ± 4.58	84.57 ± 1.19	81.11 ± 0.64	36.09 ± 4.46
DyGFormer+SF	63.94 ± 1.47	83.77 ± 0.82	72.52 ± 4.24	34.57 ± 0.43
SPLASH	**73.58 ± 0.28**	**84.89 ± 0.74**	**98.38 ± 0.09**	**55.26 ± 0.81**

features are utilized, SPLASH outperforms other complex TGNN models across four datasets. This result demonstrates that a simple model with proper node features has advantages in predicting node properties under distribution shifts, offering better performance and generalization ability.

H. Model Training Speed

In this subsection, we analyze the training time of SPLASH. We compare the empirical training time of SPLASH against that of other TGNN baselines. In this case, we run each model for 10 epochs with the same batch size of 600 and report the average training time. According to the results in Figure 5, SPLASH demonstrates the best trade-off between training speed and performance compared to other models on the Reddit dataset. Specifically, SPLASH is $64.90\times$ faster than the second-best performing baseline, DyGFormer+R in the training. SPLASH significantly outperforms JODIE, the fastest model, with a 33.24% performance gain.

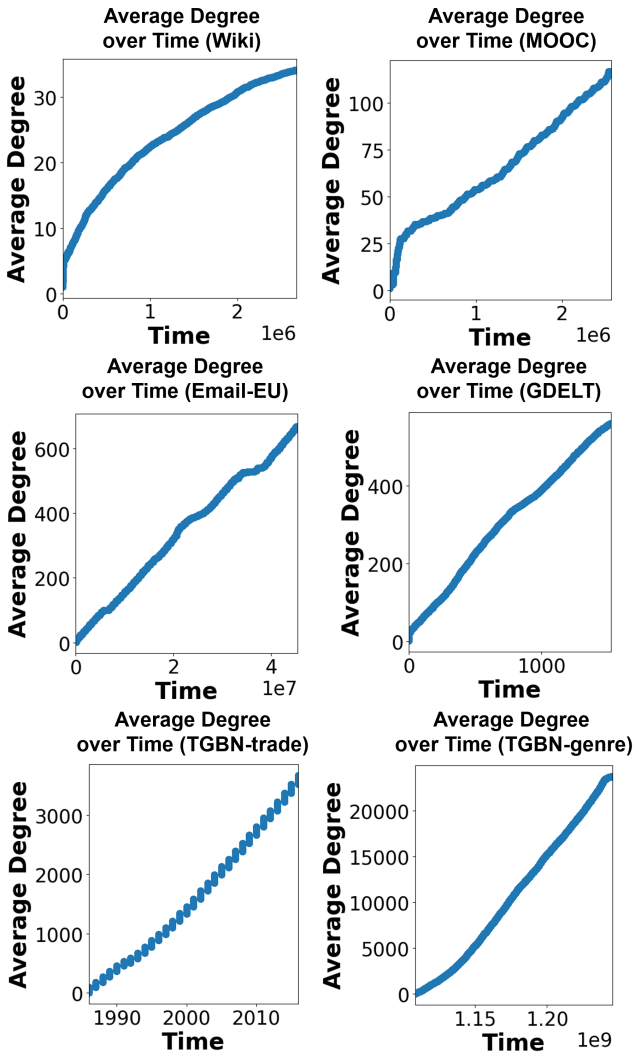

Fig. 2. Examples of structural distribution shifts in real-world network datasets. The red dotted line indicates the training end time in our setting.

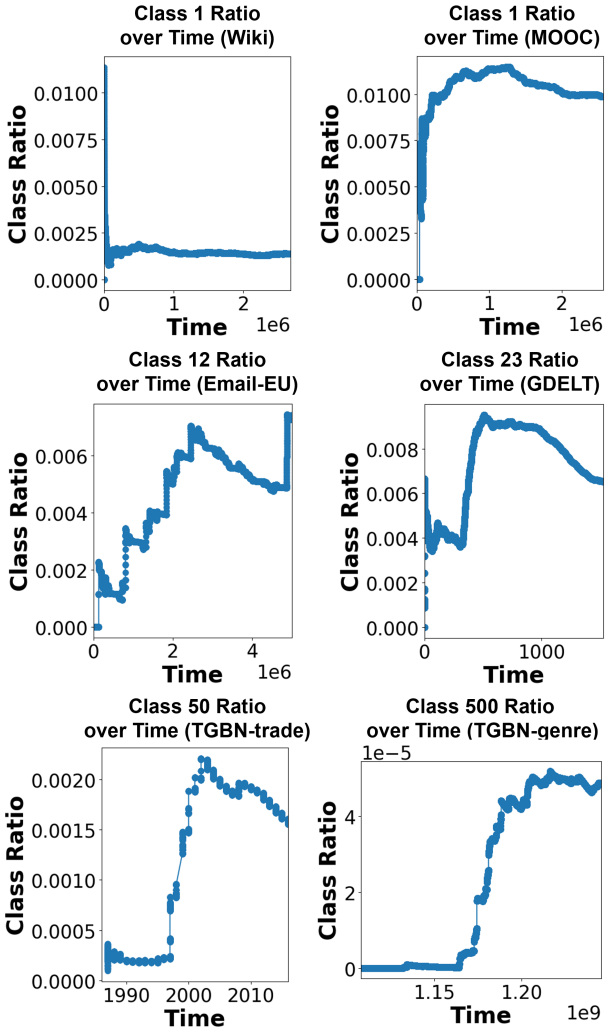

Fig. 3. Examples of property distribution shifts in real-world network datasets. These figures show the occurrence rate of classes with distribution shifts over time, compared to other classes, for each dataset. We can observe examples of a class with significant property shifts in each dataset.

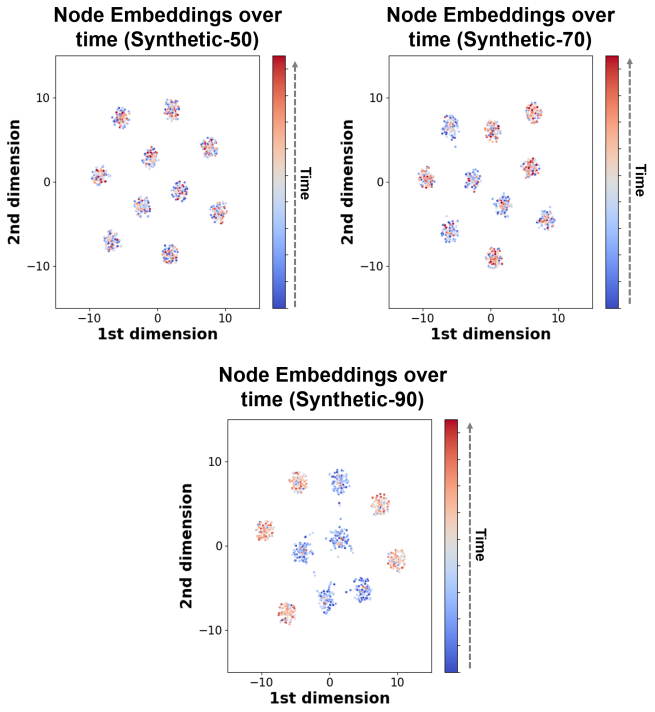

Fig. 4. positional distribution shifts analysis plot in Synthetic-50, 70, and 90. Nodes are grouped based on their appearance time, and the node embeddings generated by node2vec using the entire graph. These node embeddings are visualized using t-SNE. Positional shifts are more noticeable in datasets with a higher shift intensity factor.

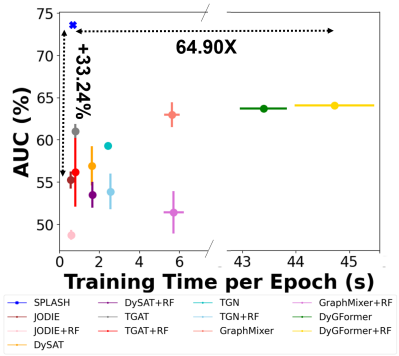

Fig. 5. This figure shows the trade-off between training time and AUC (with standard deviations) on the Reddit dataset. SPLASH provides the best trade-off between training speed and performance.

REFERENCES

[1] S. Kumar, X. Zhang, and J. Leskovec, "Predicting dynamic embedding trajectory in temporal interaction networks," in *KDD*, 2019.

[2] A. Paranjape, A. R. Benson, and J. Leskovec, "Motifs in temporal networks," in *WSDM*, 2017.

[3] H. Zhou, D. Zheng, I. Nisa, V. Ioannidis, X. Song, and G. Karypis, "Tgl: a general framework for temporal gnn training on billion-scale graphs," *PVLDB*, vol. 15, no. 8, pp. 1572–1580, 2022.

[4] S. Huang, F. Poursafaei, J. Danovitch, M. Fey, W. Hu, E. Rossi, J. Leskovec, M. Bronstein, G. Rabusseau, and R. Rabbany, "Temporal graph benchmark for machine learning on temporal graphs," in *NeurIPS*, 2024.

[5] K. Leetaru and P. A. Schrodt, "Gdelt: Global data on events, location, and tone, 1979–2012," in *ISA annual convention*, vol. 2, no. 4. Citeseer, 2013, pp. 1–49.

[6] W. Cong, S. Zhang, J. Kang, B. Yuan, H. Wu, X. Zhou, H. Tong, and M. Mahdavi, "Do we really need complicated model architectures for temporal networks?" in *ICLR*, 2022.

[7] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *KDD*, 2016.

[8] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2014.

[9] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, "Temporal graph networks for deep learning on dynamic graphs," in *ICML Workshop on Graph Representation Learning*, 2020.

[10] L. Yu, L. Sun, B. Du, and W. Lv, "Towards better dynamic graph learning: New architecture and unified library," in *NeurIPS*, 2023.

[11] A. Sankar, Y. Wu, L. Gou, W. Zhang, and H. Yang, "Dysat: Deep neural representation learning on dynamic graphs via self-attention networks," in *WSDM*, 2020.

[12] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achan, "Inductive representation learning on temporal graphs," in *ICLR*, 2020.