A. Proofs and Derivations

Theorem A.1. *Under proper a condition that $W \cdot \tilde{\boldsymbol{h}}_j^{(i)} \approx 0$ and $W \cdot \boldsymbol{h}_j^{(i)} \approx 0$, where W is the first linear transformation in F', SED with a keep ratio p ensures to reduce bias term introduced by historical embeddings by a factor of p, while introducing another regularization term.*

Proof. Let $\delta^{(i)} \triangleq \boldsymbol{h}_s^{(i)} \bigoplus \tilde{\boldsymbol{h}}_j^{(i)} - \bigoplus \boldsymbol{h}_j^{(i)}$ be the perturbation on the graph embedding. We use ET to denote using the embedding table without applying SED. For GST+E, we have

$$\delta_j^{(i)^{\text{ET}}} = \begin{cases} 0 & \text{with prob. } \frac{S^{(i)}}{J^{(i)}} \\ \tilde{\boldsymbol{h}}_j^{(i)} - \boldsymbol{h}_j^{(i)} & \text{with prob. } \frac{J^{(i)}-S^{(i)}}{J^{(i)}} \end{cases}$$

The randomness above comes from the fact that each segment $\mathcal{G}_j^{(i)}$ is selected for backpropagation with probability $\frac{S^{(i)}}{J^{(i)}}$.

For SED, there are two folds of randomness when training on graph $\mathcal{G}^{(i)}$: randomly selecting $S^{(i)}$ segments to train and randomly select stale embedding $\tilde{\boldsymbol{h}}_j^{(i)}$ to drop. Thus we can rewrite $\delta_j^{(i)}$ as

$$\delta_j^{(i)^{\text{SED}}} = \begin{cases} \frac{(1-p)(J^{(i)}-S^{(i)})}{S^{(i)}} \boldsymbol{h}_j^{(i)} & \text{with prob. } \frac{S^{(i)}}{J^{(i)}} \\ -\boldsymbol{h}_j^{(i)} & \text{with prob. } \frac{(1-p)(J^{(i)}-S^{(i)})}{J^{(i)}} \\ \tilde{\boldsymbol{h}}_j^{(i)} - \boldsymbol{h}_j^{(i)} & \text{with prob. } \frac{p(J^{(i)}-S^{(i)})}{J^{(i)}} \end{cases}$$

We apply Taylor expansion around $\delta_j^{(i)} = 0$ on the final loss to analyze the effect of this perturbation. In Section A.2 of Wei et al. (2020), when $W \cdot \tilde{\boldsymbol{h}}_j^{(i)} \approx 0$ and $W \cdot \boldsymbol{h}_j^{(i)} \approx 0$, the perturbation of $\delta^{(i)}$ to the loss function might not be too large, so it supports the use of Taylor Expansion.

$$\begin{aligned} &\mathcal{L}(F'(\boldsymbol{h}_s^{(i)} \bigoplus \tilde{\boldsymbol{h}}_j^{(i)})) - \mathcal{L}(F'(\bigoplus \boldsymbol{h}_j^{(i)})) \\ =&\mathcal{L}(F'(\bigoplus (\boldsymbol{h}_j^{(i)} + \delta_j^{(i)}))) - \mathcal{L}(F'(\bigoplus \boldsymbol{h}_j^{(i)})) \\ \approx&\sum_j D_{\boldsymbol{h}_j^{(i)}}(\mathcal{L} \circ F')[\boldsymbol{h}_j^{(i)}]\delta_j^{(i)} + \frac{1}{2}{\delta_j^{(i)}}^\top (D^2_{\boldsymbol{h}_j^{(i)}}(\mathcal{L} \circ F')[\boldsymbol{h}_j^{(i)}])\delta_j^{(i)} \end{aligned}$$

Note that we randomly select segments with index s during training, we can then derive an approximation of the expected difference during training as

$$\begin{aligned} &\mathbb{E}_s \mathcal{L}(F'(\boldsymbol{h}_s^{(i)} \bigoplus \tilde{\boldsymbol{h}}_j^{(i)})) - \mathcal{L}(F'(\bigoplus \boldsymbol{h}_j^{(i)})) \\ \approx&\sum_j \mathbb{E}_{\delta_j^{(i)}} \underbrace{D_{\boldsymbol{h}_j^{(i)}}(\mathcal{L} \circ F')[\boldsymbol{h}_j^{(i)}]\delta_j^{(i)}}_{B} + \underbrace{\frac{1}{2}{\delta_j^{(i)}}^\top (D^2_{\boldsymbol{h}_j^{(i)}}(\mathcal{L} \circ F')[\boldsymbol{h}_j^{(i)}])\delta_j^{(i)}}_{R} \end{aligned} \tag{2}$$

We can then compare the effect of SED by substituting the two versions of $\delta_j^{(i)}$ into Eq. 2.

So for the first term, we have

$$\begin{aligned} \mathbb{E}_{\delta_j^{(i)^{\text{ET}}}}[B] &= \langle D_{\boldsymbol{h}_j^{(i)}}(\mathcal{L} \circ F')[\boldsymbol{h}_j^{(i)}], \mathbb{E}\delta_j^{(i)} \rangle \\ &= \langle D_{\boldsymbol{h}_j^{(i)}}(\mathcal{L} \circ F')[\boldsymbol{h}_j^{(i)}], \frac{J^{(i)} - S^{(i)}}{J^{(i)}} \mathbb{E}(\tilde{\boldsymbol{h}}_j^{(i)} - \boldsymbol{h}_j^{(i)}) \rangle \\ \mathbb{E}_{\delta_j^{(i)^{\text{SED}}}}[B] &= \langle D_{\boldsymbol{h}_j^{(i)}}(\mathcal{L} \circ F')[\boldsymbol{h}_j^{(i)}], \mathbb{E}\delta_j^{(i)} \rangle \\ &= \langle D_{\boldsymbol{h}_j^{(i)}}(\mathcal{L} \circ F')[\boldsymbol{h}_j^{(i)}], \frac{J^{(i)} - S^{(i)}}{J^{(i)}} \mathbb{E}(\tilde{\boldsymbol{h}}_j^{(i)} - \boldsymbol{h}_j^{(i)}) * p \rangle \end{aligned}$$

whereas for the second term, we have

$$\begin{aligned}
\mathbb{E}_{\delta_j^{(i)\text{ET}}}[R] =& \langle D^2_{\boldsymbol{h}_j^{(i)}}(\mathcal{L}\circ F')[\boldsymbol{h}_j^{(i)}], \frac{\mathbb{E}\delta_j^{(i)}\delta_j^{(i)^\top}}{2}\rangle \\
=& \langle D^2_{\boldsymbol{h}_j^{(i)}}(\mathcal{L}\circ F')[\boldsymbol{h}_j^{(i)}], \frac{J^{(i)}-S^{(i)}}{2J^{(i)}}(\tilde{\boldsymbol{h}}_j^{(i)}-\boldsymbol{h}_j^{(i)})^{\odot 2}\rangle \\
\mathbb{E}_{\delta_j^{(i)\text{SED}}}[R] =& \langle D^2_{\boldsymbol{h}_j^{(i)}}(\mathcal{L}\circ F')[\boldsymbol{h}_j^{(i)}], \frac{\mathbb{E}\delta_j^{(i)}\delta_j^{(i)^\top}}{2}\rangle \\
=& \langle D^2_{\boldsymbol{h}_j^{(i)}}(\mathcal{L}\circ F')[\boldsymbol{h}_j^{(i)}], (\frac{(J^{(i)}-S^{(i)})p}{2J^{(i)}}(\tilde{\boldsymbol{h}}_j^{(i)}-\boldsymbol{h}_j^{(i)})^{\odot 2} \\
&+ \frac{(J^{(i)}-S^{(i)})(1-p)(J^{(i)}-pJ^{(i)}+pS^{(i)})}{2J^{(i)}S^{(i)}}\boldsymbol{h}_j^{(i)^{\odot 2}})\rangle
\end{aligned}$$

It is easy to check that the statement satisfies given the value calculated. □

B. Implementation Details

Datasets. MalNet (Freitas et al., 2021) is a large-scale graph representation learning dataset, with the goal to predict the category of a function call graph. MalNet is the largest public graph database constructed to date in terms of average graph size. Its widely-used split is called *MalNet-Tiny*, containing 5,000 graphs across balanced 5 types, with each graph containing at most 5,000 nodes. To evaluate our approach on the regime where the graph size is large, we construct an alternative split from the original MalNet dataset, which we named *MalNet-Large*. *MalNet-Large* also contains 5,000 graphs across balanced 5 types. *MalNet-Large*'s average graph size reaches 47k with the largest graph containing 541k nodes. We will release our experimental split for *MalNet-Large* to promote future research.

Methods. We test combinations of the following proposed techniques and some baselines. (1) Full Graph Training: we train on all graphs in their original scale without applying any partitioning beforehand. (2) GST-One: we partition the original graph into a collection of graph segments $\mathcal{G}^{(i)} \approx \bigoplus \mathcal{G}_j^{(i)}$, but we randomly select only one segment $\mathcal{G}_j^{(i)}$ for each graph to train every iteration. (3) GST: following the general GST framework described in Algorithm 1, we replace ProduceEmedding$(\cdot)$ by using the same feature encoder F to forward all the segments in $\{\mathcal{G}_j^{(i)}\}_{j\notin\mathcal{S}^{()}}$ without storing any intermediate activation. We set $S^{(i)}=1$ in our experiments. (4) E: we introduce an embedding table $\tilde{\boldsymbol{h}}_j^{(i)}=\mathcal{T}(i,j)$ to store the historical embedding of each graph segment, and we fetch the embedding from $\mathcal{T}$ if we do not need to calculate gradient for the corresponding segment. (5) F: in addition to introducing the embedding table $\mathcal{T}$, we finetune the prediction head F' with all up-to-date segment embeddings at the end of training. (6) D: we apply SED defined in Eq. 1 during training.

When these techniques are combined, we concatenate the acronyms with a "+" to GST as an abbreviation. We conduct all the experiments on MalNet with a single NVIDIA-V100 GPU with 16GB of memory. Please refer to Appendix B for additional implementation details.

Table 3. Overview of the graph datasets used in this study.

	Avg. # nodes	Min. # nodes	Max. # nodes	Avg. # edges	Min. # edges	Max. # edges
MalNet-Tiny	1,410	5	4,994	2,860	4	20,096
MalNet-Large	47,838	3,374	541,571	225,474	20,597	3,278,318

We follow GraphGym (You et al., 2020) to represent design spaces of GNN as (message passing layer type, number of pre-process layers, number of message passing layers, number of post-process layers, activation, aggregation). Our code is implemented in PyTorch (Paszke et al., 2017). We will make source code public at the time of publication.

Implementation details for MalNet-Large. We consider three model variations for the MalNet-Large dataset. Please refer to their hyperparameters in Table 4. We use Adam optimizer (Kingma & Ba, 2014) with the base learning rate of 0.01 for GCN and SAGE. For GraphGPS, we use AdamW optimizer (Loshchilov & Hutter, 2017) with the cosine scheduler

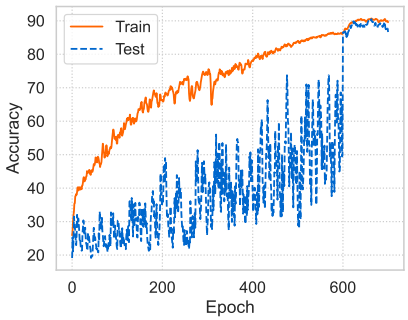

Figure 2. Accuracy curve on MalNet-Large of GST+EFD with SAGE backbone. We start Prediction Head Finetuning at epoch 600.

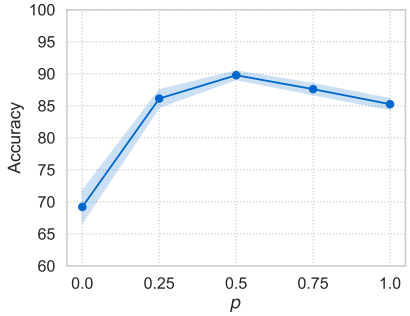

Figure 3. Ablation study on the keep ratio p in SED. We report test accuracy of GST+EFD with SAGE backbone on MalNet-Large for 5 runs.

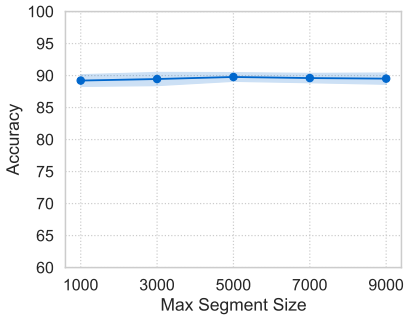

Figure 4. Ablation study on maximum segment size. We report test accuracy of GST+EFD with SAGE backbone on MalNet-Large for 5 runs.

and the base learning rate of 0.0005. We use L2 regularization with a weight decay of 1e-4. We train for 600 epochs until convergence. For Prediction Head Finetuning, we finetune for another 100 epochs. We limit the maximum segment size to 5,000 nodes, and use a keep probability $p = 0.5$ if not otherwise specified. We train with CrossEntropy loss.

Table 4. Detailed GNN/Graph Transformer designs used in MalNet-Tiny and MalNet-Large.

model	GCN	SAGE	GraphGPS
message passing layer type	GCNConv	SAGEConv	GatedGCN+Performer
pre-process layer num.	1	1	0
message passing layer num.	2	2	5
post-process layer num.	1	1	3
hidden dimension	300	300	64
activation	PReLU	PReLU	ReLU
aggregation	mean	mean	mean

Implementation details for MalNet-Tiny. We use the same model architectures/training schedules as in the MalNet-Large dataset. The only difference is that as graphs in MalNet-Tiny have no more than 5000 nodes, so we limit maximum segment size to 500 here.

C. Additional Results

C.1. Ablation Studies

Effect of finetuning. We visualize the training/test accuracy curve of GST+EFD over time in Figure 2. The staleness introduced by historical embeddings drastically hurts generalization, as shown for the first 600 epochs. We start finetuning at epoch 600, and the gap between training and test accuracy decreases by a large margin instantly.

Ablation study on segment dropout ratio. To analyze the effect of the keep ratio p in SED, we vary its value from 0 to 1 and visualize the results in Figure 3. When $p = 1$, GST+EFD degrades back to using the historical embedding table without SED, as the performance decreases due to staleness. When $p = 0$, GST+EFD becomes GST-One, where we drop all the stale historical embeddings. This extreme case introduces too heavy regularization that impedes the model from fitting the training data, leading to a decrease in test performance ultimately. We found that $p = 0.5$ achieves a satisfactory tradeoff between fitting the training data and adding a proper amount of regularization.

Ablation study on segment size. We also alter the maximum segment size and visualize the results in Figure 4. A smaller maximum segment size will result in much more number of segments. Interestingly, we found that the proposed GST+EFD

is very robust to the choice of the maximum segment size, as long as the segment size is reasonally large.

C.2. Runtime Analysis

Next, we empirically compare runtime of different variants under the proposed GST framework. We summarize an average time for one forward-backward pass during training on MalNet-Large dataset in Table 2. Since GST runs inference for the graph segments that do not require gradients, the runtime of GST is significantly higher than others'. We also found that GST+E's and GST+EFD's runtime are very close to GST-One's; this means the overhead of fetching embeddings from the embedding table $\mathcal{T}$ is minimal. Moreover, GST+EFD's runtime is slightly lower than GST+E's because in the implementation, we can skip the fetching process if an embedding is set to be dropped. This result demonstrates that our proposed GST+EFD not only is efficient in terms of memory usage but also reduces training time significantly.