A APPENDIX

A.1 GraphRNN Induce Orientation

The function INDUCEORIENTATION takes as input a graph $G = (V, E)$ and returns a directed graph $G' = (V, E')$, replacing undirected edges of E with directed edges in E'. In line 2, we first find the endpoints $u, v \in V$ of the diameter of the graph. Next, we traverse G with BFS beginning from the endpoint node u (line 3), and record the order in which nodes are accessed in the graph traversal, which is stored in the array ord. Using ord, we determine the orientation of each edge $e \in E$: we orient the edges to point from the node with smaller ord to the one with greater ord. The directed edges are collected in E', and the resulting acyclic directed graph $G' = (V, E')$ is returned.

Algorithm 3 Induce Orientation on GraphRNN Graphs

```
1:  function INDUCEORIENTATION($G = (V, E)$)
2:      Find diameter endpoints
3:      $u, v \leftarrow$ DIAMETER($G$)
4:      Record BFS traversal order
5:      $ord[] \leftarrow$ BFS($G, u$)
6:      Set of directed edges
7:      $E' \leftarrow \emptyset$
8:      for $e := (u', v') \in E$ do
9:          if $ord[u'] < ord[v']$ then
10:             Add $u' \rightarrow v'$ to $E'$
11:         else
12:             Add $v' \rightarrow u'$ to $E'$
13:     $G'$ is an acyclic orientation
14:     return $G' := (V, E')$
```

A.2 Parameterization

PROTEUS provides a number of tunable parameters to the model owner outlined in figure 8. These parameters allow for tradeoffs between (a) the complexity of recovery by an adversary, (b) the computational overhead for the optimizer, and (c) the quality of model optimizations (in particular, the slowdown compared to optimizing without partitioning).

Name	Description
n	Number of *graph partitions* generated from the protected graph
k	Number of *sentinel subgraphs* generated per protected subgraph

Figure 8. List of tunable parameters provided by PROTEUS

We tabulated the precise tradeoffs as a result of these parameters in Figure 9. These tunable parameters allow the model owner to tradeoff some potential speedups for additional and stronger obfuscation.

For the DNNs in our evaluation, optimizing the original model takes 6s on average and up to 22s with Hidet on an AMD EPYC 7282 CPU. If Proteus produces 50 sentinel graphs, thus optimizing both the original as well as sentinels would take 5 minutes on average and up to 18 minutes. PROTEUS results in a k-fold increase in compilation time, where k is the number of sentinels generated per partition, and as demonstrated above this cost is not prohibitive.

Item	Cost
Recovery cost of adversary	$\mathcal{O}((k+1)^n)$
Computational overhead of optimizer	$\mathcal{O}(k)$
Quality of model optimizations	See figure 10

Figure 9. Tradeoffs resulted from PROTEUS's tunable parameters

A.3 Subgraph Size vs. Slowdown

As suggested in section 5.2, many optimizations would be ineligible due to the small graph size (as optimizations cannot be applied *across* partitions). PROTEUS provides the number of subgraphs n as a tunable parameter to the user. In this section we evaluate the correlation between subgraph size and slowdown compared to optimal (where the entire graph is optimized as a whole).

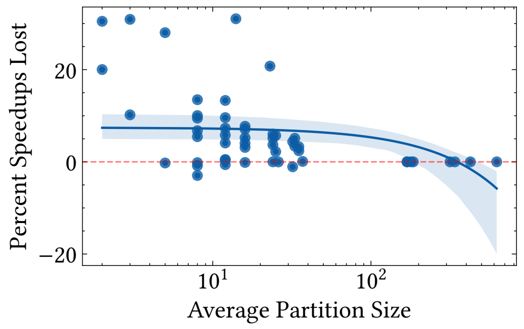


Figure 10. Average Subgraph Size vs. % Performance Loss with PROTEUS

Figure 10 evaluates the trade-off between the size of the subgraphs and potential performance loss across all our evaluated DL models. We normalize performance loss as percentage over *Best Attainable*. Each point in the figure represents the inference latency for a model and one of the three setups above. We observe that with very small subgraph sizes, PROTEUS incurs small performance losses. However, when the average subgraph size becomes large, PROTEUS achieves negligible performance losses.

A.4 Graph Metrics of Generated Sentinels

Let us consider $\mathcal{D}$ to be the distribution of generated sentinel graphs, and let G be the real subgraph. The distribution of various graph characteristics for $\mathcal{D} \cup \{G\}$ should not be significantly different from the original distribution $\mathcal{D}$. Otherwise, an adversary could distinguish the protected subgraph by learning the characteristics of the generated sentinels. For this, it is necessary that the sentinel subgraphs are realistic and similar to real world subgraphs.

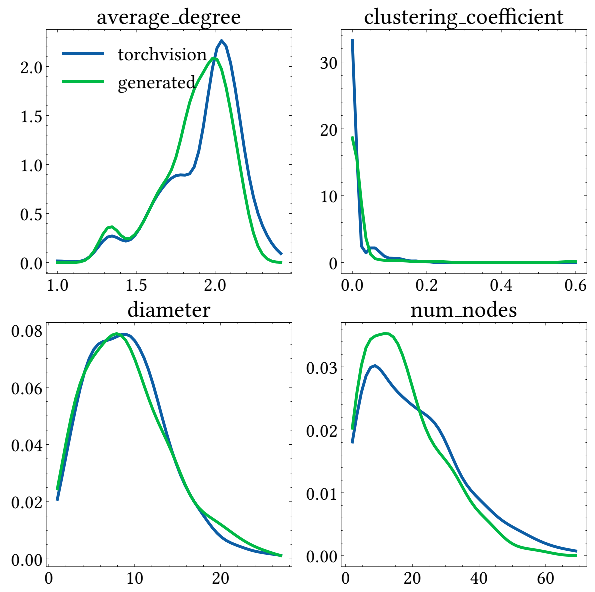


Figure 11. Comparing distributions of graph statistics between real and PROTEUS-generated subgraphs

Figure 11 presents the distributions of (i) the real-world subgraphs extracted from `torchvision` models, and (ii) the sentinel (artificially) generated subgraphs produced by PROTEUS (algorithm 1) using different graph statistics. We use the set of graph statistics explained in Section 4.1.2, i.e., (a) average degree, (b) clustering coefficient, (c) graph diameter and (d) number of nodes.

The X axis represents the value for the particular metric and the Y axis represents the probability density. From the figures we observe very little statistical difference between the two groups. Thus, we conclude that the sentinel subgraphs produced using our approach are highly realistic, resembling real world subgraphs and it would be very difficult for a potential adversary making use of these graph statistics from to differentiate real graphs from the sentinels.

A.5 Classifier Architecture

The classification network performs graph convolutions using SAGEConv (Hamilton et al., 2018) with the objective to learn features of the local neighborhood for each node in the graph. After that, aggregation is done across the nodes with mean reduction. This step essentially generates a hidden representation for the entire graph from the node representations after the GNN. The final classification is generated with linear layers acting on the reduced graph representation.

A.6 Adversary Cost

The adversary attempts to shrink its search space by eliminating fake graphs from the obfuscated bucket so as to narrow down on the "possibly-real" graphs. We especially note that it must not classify any real subgraphs as being sentinels, as that would eliminate the actual protected graph from its search space. Therefore it must fix some decision boundary γ, such that a graph is eliminated as fake when the classifier's confidence exceeds γ.

The value of γ defines how *conservative* the adversary is in eliminating potential sentinel graphs. Decreasing γ would reduce the number of "potentially-real" graphs, resulting in a smaller search space. However, by doing so the adversary also risks incorrectly eliminating a real subgraph.

In practice, it would be difficult for an actual adversary to obtain the minimum value of γ without a priori knowledge of the protected graphs, since it would not have access to the confidences of the classifier on them. However, we wish to establish an approximate *lower bound* for the cost of attack by assuming the pessimistic case where the adversary obtains the optimal (i.e. lowest possible) γ through some (perhaps statistical) means.

A.7 Case Studies

For both cases, we use our standard configuration, setting $n = \lfloor N/8 \rfloor$ (where N is the total size of the original model, such that subgraphs on average have 8 nodes) and $k = 50$.

A.7.1 Optimizing a NAS model

In addition to faithfully recreating the effects of optimizing the model directly, the obfuscation mechanism also is effective in protecting the model against recovery. In this case, the GNN classified many real graphs as being fake (in many cases concluding which with a 99% certainty). Shown in Figure 12 are two examples incorrectly classified subgraphs from NATSBench.

Setting γ accordingly resulted in a sensitivity of 84.9%. With $n = 24$ and $k = 50$, this resulted in $[50(1 - 0.849)]^{24} \approx 1.18 \times 10^{21}$ candidates which the adversary would need to evaluate.

A.7.2 Optimizing a ResNet-like Model

Due to the similarity between SEResNet with ResNet, we expect most of the real subgraphs to be classified correctly, but some fake graphs are classified as real. Setting $\gamma = 0.79$, the sensitivity of the classifier is 44%. Using $n = 83$ and $k = 20$, the number of potential candidates is $[20(1 - 0.44)]^{83} \approx 1.22 \times 10^{87}$.

We show a few examples of incorrectly classified SEResNet examples in figure 13.

A.8 Survey Study

We evaluate the possibility of manual identification by experts to eliminate sentinels produced by PROTEUS. To do so, we conducted an internel survey amongst ML researchers from the authors' primary institute. The survey contains 20 subgraphs consisting of 10 real subgraphs extracted models taken from `torchvision` and HuggingFace, and also another 10 subgraphs, which are PROTEUS-created sentinels using the aforementioned real models.

To generate graphs for the survey, PROTEUS is configured to partition reals graphs into subgraphs of sizes $8 - 16$ nodes. Next, we use PROTEUS to generate fake graphs from each of the real partitions. This way we create a pool of real subgraphs and another pool of PROTEUS sentinel subgraphs. The 20 graphs are selected from the aforementioned pools at random with some small graphs filtered out.

For each of the 20 graphs, we ask the participants to select between two options: (i) real and (ii) fake subgraph.

Out of 13 participants, the average accuracy is 52%, which demonstrates that the average guess of a participant is the same effective as guessing randomly. We conclude that the sentinel graph generation is robust against identification using expert knowledge and manual intervention through visual inspection. The survey can be accessed here and provides visual examples of sentinel and real graphs, demonstrating the realism of the sentinel graphs.

B ARTIFACT INSTRUCTIONS

B.1 Abstract

Our artifact provides the code to reproduce two of our main results, that (a) the partition-optimize-reassemble workflow preserves the efficacy of optimizations, and (b) our graph obfuscation pipeline makes it difficult for a learning-based adversary to distinguish real graphs from sentinel ones.

B.2 Artifact check-list (meta-information)

- **Data set:** Deep learning models in ONNX format from open source libraries such as `torchvision`
- **Hardware:** Machine with NVIDIA A100 GPU
- **How much disk space required (approximately)?:** 100GB
- **How much time is needed to prepare workflow (approximately)?:** $<$ 1 hour
- **How much time is needed to complete experiments (approximately)?:** $<$ 10 hours (significantly less if fast path is taken)
- **Publicly available?:** Yes
- **Code licenses (if publicly available)?:** Apache-2.0
- **Data licenses (if publicly available)?:** Apache-2.0
- **Workflow framework used?:** PyTorch, ONNX, ONNXRuntime, Hidet
- **Archived (provide DOI)?:** `10.5281/zenodo.10977142`

B.3 Description

B.3.1 How delivered

The source code and scripts are available in the GitHub repository: `https://github.com/proteus-mlsys24/mlsys24-artifact`.

You can also find the archival version hosted on Zenodo at `10.5281/zenodo.10977142`.

B.3.2 Hardware dependencies

Requires an NVIDIA A100 GPU to run the runtime-related experiments. Many CPU cores ($\approx$ 32) is recommended to run the adversary experiment (figure 5).

B.3.3 Software dependencies

Since our experiments run inside Docker, Docker with NVIDIA GPU support (through `nvidia-docker`) is required.

B.4 Installation

1. *Docker.* Install Docker by following instructions at https://docs.docker.com/engine/install/.
2. *nvidia-docker.* To use Docker with NVIDIA GPUs, we need to install `nvidia-docker`. To do so, follow the instructions at https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/install-guide.html.
3. *Artifact Code.* Clone the repository to obtain the source code:

```
$ git clone https://github.com/proteus-mlsys24/mlsys24-artifact
```

B.5 Experiment workflow

- *Figure 3a: ONNXRuntime Speedup*

  Navigate to `figures/fig3a-ort-speedup` and run `run.sh`.
- *Figure 3b: Hidet Speedup*

  Navigate to `figures/fig3b-hidet-speedup` and run `run.sh`.
- *Figure 5: GNN Classifier Adversary*

  Navigate to `figures/fig5-gnn-classifier` and follow the instructions in `README.md`.

B.6 Evaluation and expected result

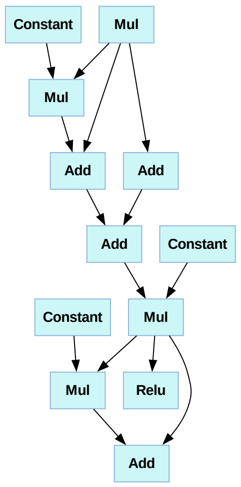
- *Figure 3a/b: Speedups*

  These experiments should generate a figure named `speedups.pdf` similar to those in the paper.
- *Figure 5: GNN Classifier Adversary*

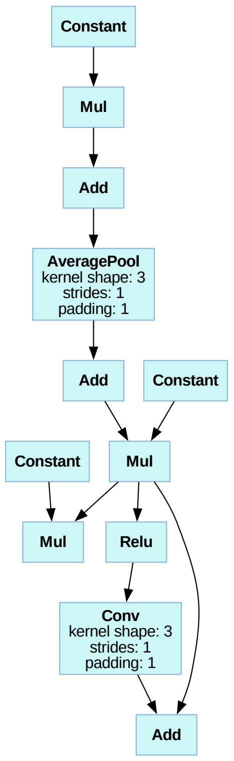
  For each model m, the number of candidates should be large. We further expect the number of candidates for m to be much larger than m_randop (the baseline with random opcodes).

B.7 Experiment customization

The `proteus` Python package is available as a standalone package for model obfuscation.

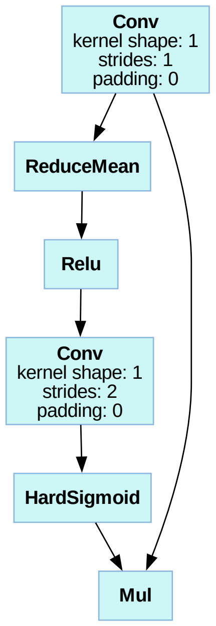


(a) NATSBench: Sentinel incorrectly classified as real

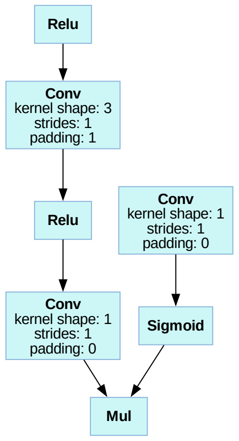


(b) NATSBench: Real incorrectly classified as sentinel

Figure 12. NATSBench Examples



(a) SEResNet: Sentinel incorrectly classified as real



(b) SEResNet: Real incorrectly classified as sentinel

Figure 13. SEResNet Examples