URL Phishing Classification using LLM Distillation

A detailed report

Aditya Singh

31 March 2025

Introduction

In today's digital era, the surge in phishing attacks poses a significant threat to individuals and organisation's alike. Cybercriminals are continuously evolving their methods, making it essential to develop robust and efficient models for detecting malicious URLs.

This project aims to address this by to address these challenges by leveraging the power of Transformer-based architectures and advanced training and fine-tuning techniques to create a light-weight model which can be run locally while providing excellent accuracy on detection and classification of malicious URL's.

The project follows the below workflow architecture which we will go over in this report in detail :-

1. Fine-tuning a teacher model (BERT base)
2. Distillation of Student model with Teacher (BERT Uncased)
3. Fine-tuning of Student model for comparison
4. Quantisation of Distilled Student model for efficiency

At the end of this project, our objective is to create a lightweight and efficient URL Phishing classifier. You may check the full codes used in this project at GitHub.

Model Overview

This project aims to train a LLM Model for Malicious URL Classification. Generally for text classification tasks, Encoder only models like **BERT** (Bi-directional Encoding Representation from Transformers) family of models are used due to their ability to capture contextual information from both the left and right sides of a token, enabling a richer understanding of nuanced semantic relationships that are critical for accurately distinguishing malicious URLs from benign ones.

We have used ***google-bert/bert-base-uncased*** model as teacher model which has approximately 110M parameters which is appropriately large for our task of binary classification of URL's which are basically short text strings.

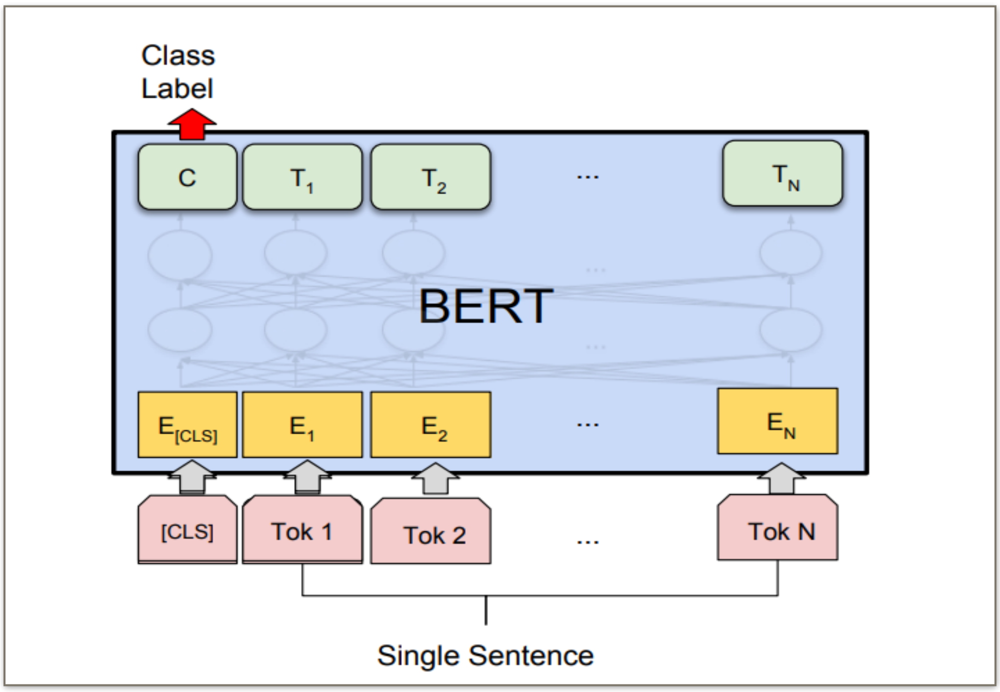

Fig. 1 BERT Model for Text Classification

After fine-tuning of teacher model, ***distilbert/distilbert-base-uncased*** was chosen as the student model which has ~66M parameters and is much faster in inference tasks than the ***bert-base*** model.

The next steps involve distillation of the student model on teacher model by using KL-Divergence loss. For comparison, we will also separately fine-tune the student model to compare distillation v/s fine-tuning performance. At the end we will experiment with quantisation to evaluate tradeoff between precision of model weights and accuracy on test data.

Objective

The objective of this project is to develop a lightweight and accuracy Phishing URL classifier and deducing the best way of training such a model by comparing distillation, fine-tuning and quantisation strategies in training and carrying out inference on Large Language Models for this classification task.

Dataset Preparation

For the task of fine-tuning and distillation, we selected the ***shawhin/phishing-site-classification*** which consists of ~2.1K URL's for training and 450 URL's each for validation and test sets.

```
DatasetDict({
    train: Dataset({
        features: ['text', 'labels'],
        num_rows: 2100
    })
    validation: Dataset({
        features: ['text', 'labels'],
        num_rows: 450
    })

    test: Dataset({
        features: ['text', 'labels'],
        num_rows: 450
    })
})
```

The dataset is suitable as it has been sampled to be balance so that small models do not overfit on a particular class with a nearly 50-50 split between 0 : ”Safe” and 1 : ”Not Safe” Classes.

To prepare the dataset to be ingested by the model, it needs to be properly tokenised and chunked into size acceptable by the model.

```
def preprocess_function(examples):
    return tokenizer(examples["text"], padding='max_length', truncation=True)

tokenized_dataset = dataset_dict.map(preprocess_function, batched=True)
tokenized_dataset.set_format(type='torch', columns=['input_ids',
'attention_mask', 'labels'])

data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
```

The above code is responsible for converting the text in the dataset columns into appropriate attention_masks and labels for predictions. The data collator creates batches with appropriate padding expected by the model tokenizer for training pipeline.

This concluded the dataset preparation section where we have loaded, processed and tokenized the dataset into format accepted by the BERT model.

Model Development

The model development in this project is split into 4 sections for fine-tuning, distillation and quantisation of student and teacher models.

A. **Fine-tuning a teacher model (BERT base):**
   The ***google-bert/bert-base-uncased*** was loaded appropriately with proper arguments and polling layers were specifically unfreezed for the fine-tuning process to specifically allow it to be retrained on our dataset parameters.

   The model was then fine-tuned fully on the train dataset for 10 epochs on batch size of 32 with learning rate of 2e-4 which was experimentally determined to provided optimum results for our use.

```
    model_path = "google-bert/bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_path)
id2label = {0: "Safe", 1: "Not Safe"}
label2id = {"Safe": 0, "Not Safe": 1}
model = AutoModelForSequenceClassification.from_pretrained(model_path,
                                                           num_labels=2,
                                                          id2label=id2label,

                                                           label2id=label2id)
```

The model was then fine-tuned fully on the train dataset for 10 epochs on batch size of 32 with learning rate of 2e-4 which was experimentally determined to provided optimum results for our use.

B. Distillation of Student model with Teacher (BERT Uncased):

The ***distilbert/distilbert-base-uncased*** was loaded along with the fine-tuned teacher model from huggingface. The model was trained for 10 epochs on a learning rate of 1e-4 on a custom Distillation loss which combine KL-Divergence loss and Cross-Entropy Loss for the student model.

```
    def distillation_loss(student_logits,teacher_logits , true_labels,temperature, alpha):

    soft_targets = nn.functional.softmax(teacher_logits / temperature, dim=1)
    student_soft = nn.functional.log_softmax(student_logits / temperature, dim=1)

    distill_loss = nn.functional.kl_div(student_soft, soft_targets, reduction='batchmean')
*(temperature ** 2)
    hard_loss = nn.CrossEntropyLoss()(student_logits, true_labels)

    loss = alpha * distill_loss + (1.0 - alpha) * hard_loss

    return loss
```

C. Fine-tuning of Student model for comparison:

The ***distilbert/distilbert-base-uncased*** was loaded and fine-tuned on the train dataset just like the teacher model on Cross-Entropy Loss for 10 epochs. The model is solely trained to serve as a baseline comparison against distillation training on the same model.

D. Quantisation of Distilled Student model for efficiency:

The ***distilbert/distilbert-base-uncased*** distilled model was loaded from hugginface as quantised into 4-bit using the bitsandbytes module and subsequently evaluated on the test data for evaluation of full-precision v/s 4-bit precision performance of the distilled model.

```
    nf4_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype = torch.bfloat16,
    bnb_4bit_use_double_quant=True
    )

model_nf4 = AutoModelForSequenceClassification.from_pretrained(model_id,
device_map=device, quantization_config=nf4_config)
```

Challenges and Solutions

The process of fine-tuning and distillation of models present several challenges which need to be experimented with to resolve issues and achieve a stable training and optimum results:

1. **Distillation Procedure:**
   Distillation needs to be carefully adjusted with parameters such as alpha which adjusts the ratio of Cross-Entropy Loss which is directly from labels and KL-Divergence loss which is from the Teacher model logits needs to be carefully adjusted.

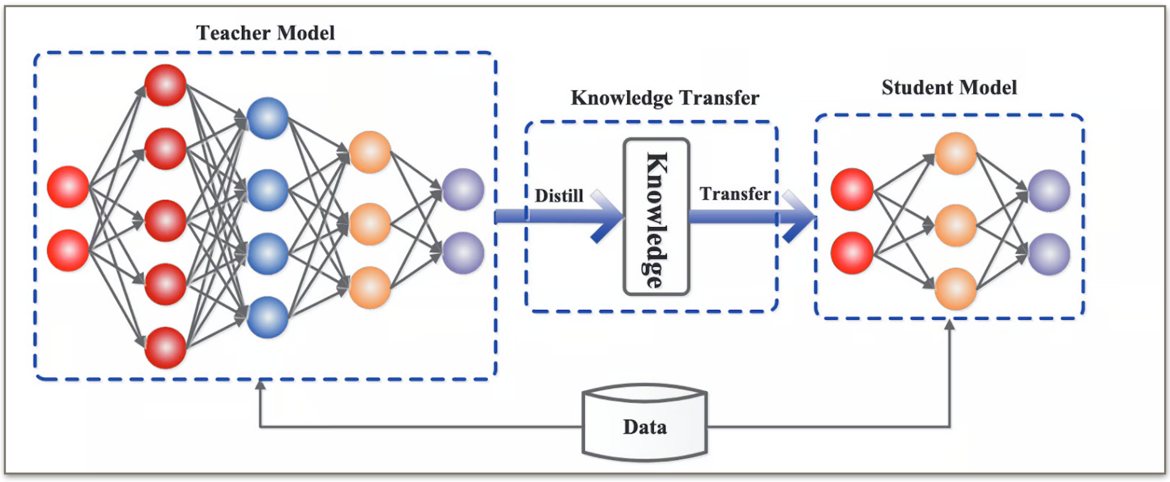

Fig.2 Distillation Procedure

   Too high alpha leads to full-finetuning with minimal advantage from teacher while too low of alpha can lead to divergence of model from actual training data. In our case, we selected alpha as 0.5 to strike a balance between data learning and logits learning from teacher.

2. **Fine-tuning Procedure:**
   During Fine-tuning, the models frequently were undercutting the data as the learning rate was too low, I had to gradually increase learning rate to 1e-4 for the model to properly fit the training data. This in my opinion was due to the relatively low number of parameters in the models which required more high learning rates to fit on such a small dataset in a reasonable number of epochs, 10 in our case.

Evaluation

Accuracy was chosen as the main metric in this case as our task was essentially binary classification and the dataset was balanced so accuracy was the simplest and mots relevant metric for our case.

Precision, Recall and F1 score were also calculated as False Negatives were required to keep probability of a malicious URL to be classified as legitimate was as low as possible.

Below is the combined metrics from our model training procedure:

Model	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**Notes**
Teacher (BERT base)	0.8711	0.9073	0.8267	0.8651	Full-sized model
Student (Distilled)	0.9267	0.9486	0.9022	0.9248	Smaller architecture
Student (Fine-tuned)	0.8620	-	-	-	Base Student model
Student (Quantized)	0.9156	0.9401	0.8908	0.9148	4-bit quantization

As clearly visible distillation training of the student model clearly out performs direct fine-tuning and even has higher accuracy than the teacher model.

Quantisation of the distilled model lead to a ~1% tradeoff in accuracy in exchange for atleast 4x less memory consumption and faster compute.

This demonstrates the ability of training efficient and high performance models by use of pre-trained large models with a dataset to match and outperform the teacher Model.

Impact

This project demonstrates that lightweight URL phishing classifiers can achieve, and even surpass, the performance of larger models through strategic distillation and fine-tuning.

By leveraging a teacher-student framework combined with quantisation, the final model delivers robust accuracy while reducing memory footprint by at least 4x and speeding up inference.

This balance of efficiency and effectiveness makes it a practical solution for real-world deployment in environments where resource constraints are critical, ultimately enhancing cybersecurity measures against phishing attacks.

Scalability

The approach to solving the issue of detection and classification of phishing URL's has the following advantages over traditional LLM models:

Adaptability to Larger Datasets: Leveraging pre-trained models and distillation allows the approach to scale as more data becomes available, ensuring improved generalisation over time.

Resource Efficiency: Quantisation and fine-tuning methods reduce computational and memory requirements, making the system viable for deployment on both high-end servers and edge devices.

Incremental Learning: The training strategies support continuous updates and refinements, facilitating the model's evolution with emerging phishing techniques.

Practical Deployment: The lightweight architecture ensures that as the system scales, it remains efficient and responsive in real-world environments.

The final quantised model is ~60MB in parameters which can be easily run and hosted either in local applications as an application or server side for browser protection. Hence this approach can be scaled easily into larger applications for wise-spread use.

Conclusion

This project successfully demonstrates the viability of using advanced training techniques—fine-tuning, distillation, and quantisation—to build a lightweight yet highly effective URL phishing classifier.

By leveraging a teacher-student framework, the distilled student model not only matched but exceeded the performance of the larger teacher model, all while significantly reducing resource requirements.

The strategic application of quantisation further enhanced inference speed and memory efficiency, making the solution practical for deployment in real-world, resource-constrained environments.

Overall, the methodology outlined here provides a robust framework for scaling phishing detection systems, ensuring that cybersecurity defenses can keep pace with evolving threats.

Links

A. Fine-tuned Teacher model:majorSeaweed/Bert_Uncased_FishingURL_Teacher
B. Distilled Student model:majorSeaweed/Bert_Uncased_FishingURL_distilled
C. Fine-tuned Student model:majorSeaweed/distilled_Bert_Uncased_FishingURL_Student_Finetuned
D. Dataset: shawhin/phishing-site-classification