Transformer-Based Language Models for Software Vulnerability Detection

Chandra Thapa, Seung Ick Jang, Muhammad Ejaz Ahmed, Seyit Camtepe, Josef Pieprzyk, and Surya Nepal

CSIRO's Data61, Marsfield, Australia

{chandra.thapa, seung.jang, ejaz.ahmed, seyit.camtepe, josef.pieprzyk, surya.nepal}@data61.csiro.au

Annual Computer Security Applications Conference (ACSAC)

December 5 – 9, 2022

Outline

- Problem
- Transformer-based language models for software vulnerability detection
- Systematic framework
- Our results
- Discussion on available platforms
- Conclusion

Problem

- Software is an integral part of most computing devices.
- Adversaries exploit these software vulnerabilities to gain unauthorized system control and steal or modify sensitive and private data for their benefit.
- Finding and patching these vulnerabilities are important to secure the products from adversaries.

Google Patches Actively Exploited Chrome Bug

Apple's Zero-Day Woes Continue

Two new bugs in macOS and iOS disclosed this week add to the growing list of zero-days the company has rushed to patch over the past year.

Microsoft Exchange Server targeted with zero-day vulnerabilities

Microsoft warned that two unpatched zero-day vulnerabilities are being exploited against Exchange Server, a problem that's causing déjà vu for some researchers.

We

Zoom Patches Zero-Day Vulnerability in Windows 7

The flaw also affects older versions of the operating system, even if they're fully patched.

Code vulnerability failures in manufacturing on display in Toyota supply chain attack

VDB-168508 · CVE-2021-1068 · ID 5148

NVIDIA SHIELD TV UP TO 8.2.1 NVDEC BUFFER OVERFLOW

VDB-163591 · CVE-2020-9940

APPLE MACOS USD FILE BUFFER OVERFLOW

VDB-160672 · CVE-2020-1886

FACEBOOK WHATSAPP ON ANDROID VIDEO STREAM BUFFER OVERFLOW

Examples of software vulnerability

```c
static char * badSource(char * data)
{
    data = (char *)malloc(50*sizeof(char));
    data[0] = '\0';
    return data;
}

static void bad()
{
    char * data;
    data = NULL;
    data = badSource(data);
    {
        char source[100];
        memset(source, 'C', 100-1);
        source[100-1] = '\0';
        strcat(data, source); /* Bad */
        printLine(data);
        free(data);
    }
}
```

(a) Vulnerable with Buffer Error

```c
static char * goodSource(char * data)
{
    data = (char *)malloc(100*sizeof(char));
    data[0] = '\0';
    return data;
}

static void good()
{
    char * data;
    data = NULL;
    data = goodSource(data);
    {
        char source[100];
        memset(source, 'C', 100-1);
        source[100-1] = '\0';
        strcat(data, source); /* OK */
        printLine(data);
        free(data);
    }
}
```

(b) Non-vulnerable

Examples of software vulnerability

```
static void bad()
{
    int64_t * data;
    data = NULL;

    data = new int64_t;
    *data = 5LL;
    printLongLongLine(*data);

    /* Bad: Need to release 'data' */
    return;
}
```

(a) Vulnerable with Resource Management Error (RME)

```
static void good()
{
    int64_t * data;
    data = NULL;

    data = new int64_t;
    *data = 5LL;
    printLongLongLine(*data);

    delete data; /* OK */
    return;
}
```

(b) Non-vulnerable

Machine learning to find software vulnerabilities

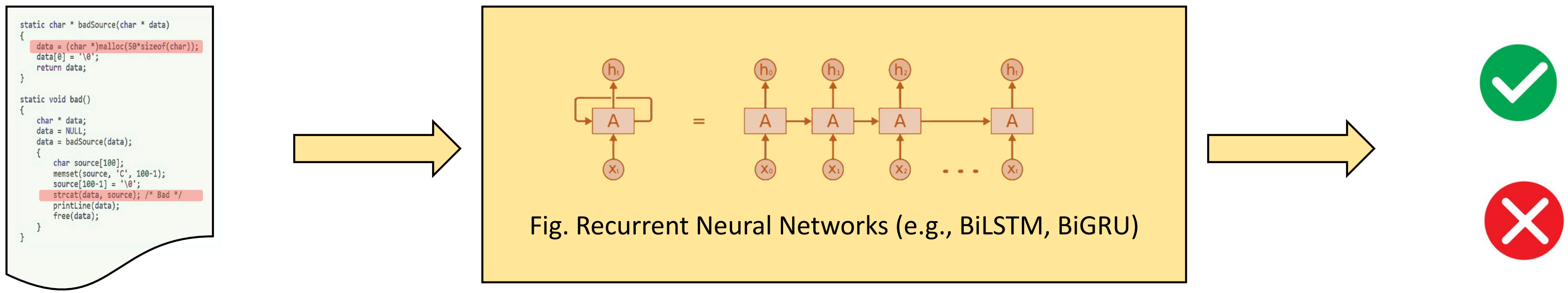


Fig. Recurrent Neural Networks (e.g., BiLSTM, BiGRU)

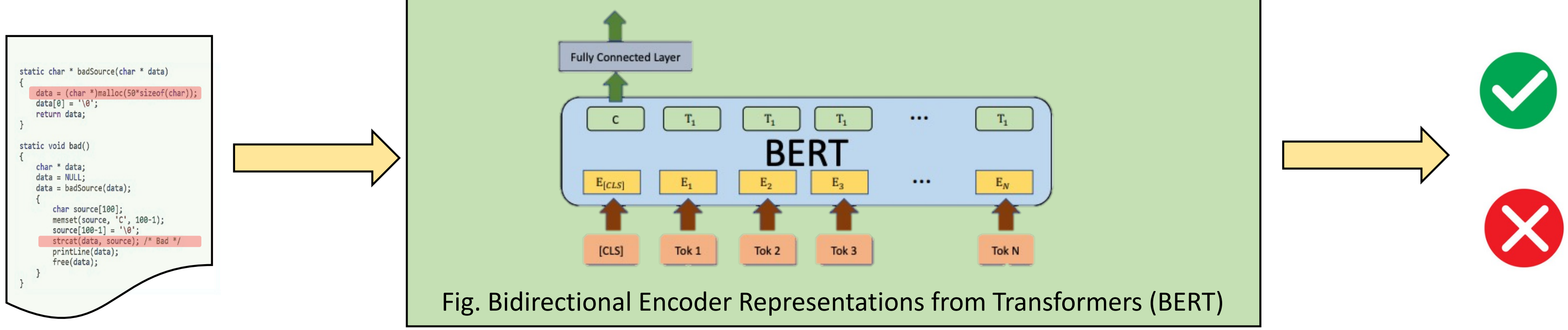


Fig. Bidirectional Encoder Representations from Transformers (BERT)

Transformer-based language models

- Transformer-based models are SOTA models in Natural Language Processing (NLP) tasks.
- We can extend the use of these models beyond NLP tasks through the process formally known as “Transfer learning.”
- Examples: BERT, DistilBERT, CodeBERT, GPT-2, MegatronBERT, MegatronGPT-2.

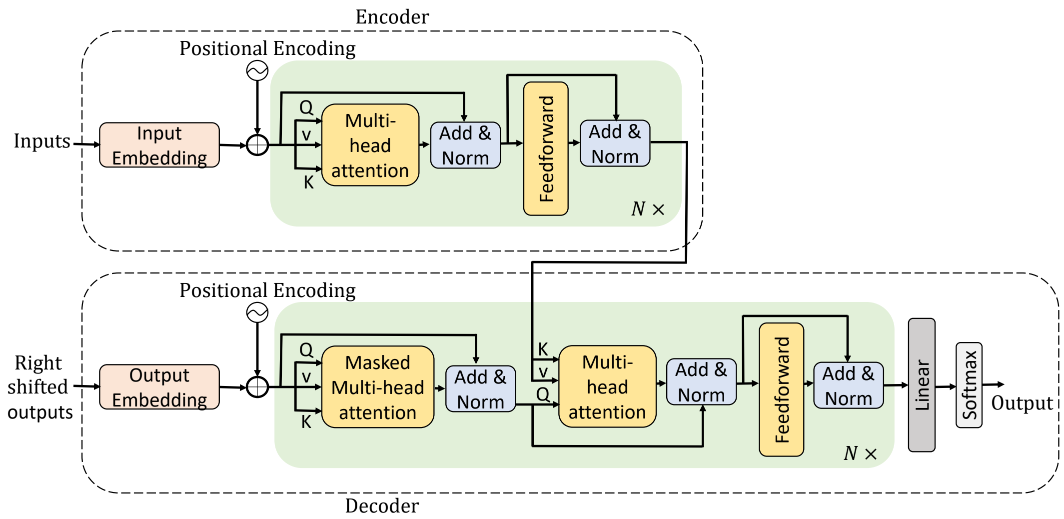
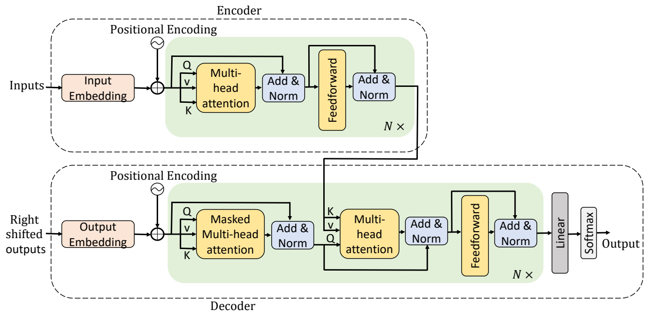
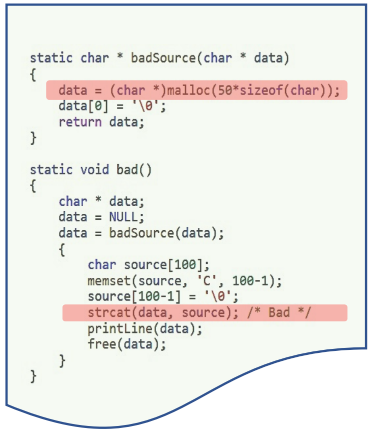


Fig. Transformer architecture

Transformer-based language models for software vulnerability detection





```
static char * badSource(char * data)
{
    data = (char *)malloc(50*sizeof(char));
    data[0] = '\0';
    return data;
}

static void bad()
{
    char * data;
    data = NULL;
    data = badSource(data);
    {
        char source[100];
        memset(source, 'C', 100-1);
        source[100-1] = '\0';
        strcat(data, source); /* Bad */
        printLine(data);
        free(data);
    }
}
```

RQ1: How to effectively leverage transformer-based language models for software vulnerability detection?

RQ2: How well do existing transformer-based language models detect software vulnerabilities compared to other contemporary RNN-based models?

RQ3: Which platform is efficient for running these models?

Systematic framework

Data translation:

- Code Gadgets and their extraction

Li et al. [1] propose the code gadgets.
Generation:
1) Load all C/C++ files.
2) Normalize source codes, includes removing comments.
3) Extract all functions and variable definitions together.
4) For library/API function call, perform a back-track
5) Extract all variable names from the function call
6) Stack up all lines which have relationship with the variables
7) If any variable passes from a caller, perform another back-track for the caller.

- Data preparation
  - Data cleaning (removing duplicate code gadgets and the same gadgets with conflicting labels)
  - Data preprocessing (replacing the user-defined function name and user-assigned variable names)
  - Data partitioning (into groups)
  - Word embeddings (tokenization and embeddings)

Dataset	Type	Original	Cleaned	Train	Test
Group 1	Buffer Error (BE)	10440	7649	6161	1488
	Non-vulnerable	29313	12262	9768	2494
Group 2	Resource Management Error (RME)	7285	2757	2214	543
	Non-vulnerable	14600	5010	4000	1010
Group 3	BE+ RME	17725	10395	8368	2027
	Combined Non-vulnerable	43913	17197	13704	3491

Word Embeddings	Embedding size	Models
Tokenizer (our own) and Word2Vec	512	BiLSTM, BiGRU
Huggingface's Bert Tokenizer (Tokenize based on WordPiece)	512	BERT (BERTBase, MegatronBERT)
Huggingface's DistilBert Tokenizer (Runs end-to-end tokenization based on punctuation splitting and WordPiece)	512	DistilBERT
Huggingface's Roberta Tokenizer (Derived from GPT-2 tokenizer using byte-level Byte-Pair-Encoding)	514	RoBERTa, CodeBERT
Huggingface's GPT-2 Tokenizer (Based on byte-level Byte-Pair-Encoding)	1024	GPT-2 (GPT-2 Base, GPT-2 Large, GPT-2 XL, MegatronGPT-2), GPT-J
Huggingface's GPT-2 Tokenizer (Based on byte-level Byte-Pair-Encoding)	2048	GPT-J

[1] Zhen Li, Deqing Zou, Shouhuai Xu, Xinyu Ou, Hai Jin, Sujuan Wang, Zhijun Deng, and Yuyi Zhong, "VulDeePecker: A Deep Learning-Based System for Vulnerability Detection," In Proc. NDSS 2018.

Systematic framework

Provider	Language Model	Size	#Parameters
Nvidia	MegatronBERT	Standard	345M
	MegatronGPT-2	Standard	345M
Hugging Face	BERT	Base Model	110M
OpenAI	GPT-2	Base Model	117M
		Large Model	774M
		XL Model	1.5B
EleutherAI	GPT-J	Standard	6B
Hugging Face	DistilBERT	Standard	66M
Microsoft	CodeBERT	Standard	125M
Hugging Face	RoBERTa	Standard	125M
VulDeePecker	BiLSTM	Standard	1.2M
SySeVR	BiGRU	Standard	1.6M

Table. Models considered in our studies and their sizes.

Systematic framework

Language Model	Classification Head
BERT, MegatronBERT	Dropout Layer + Linear Layer (size = 3072)
DistilBERT	Linear Layer (size = 3072) + ReLU + Dropout Layer + Linear Layer (size = 3072)
RoBERTa, CodeBERT	Dropout Layer + Linear Layer (size = 3072) + tanh + Dropout Layer + Linear Layer (size = 3072)
GPT-2, MegatronGPT-2	Dropout Layer + Linear Layer (size = 1024)
GPT-J	Linear Layer (size = 2048)

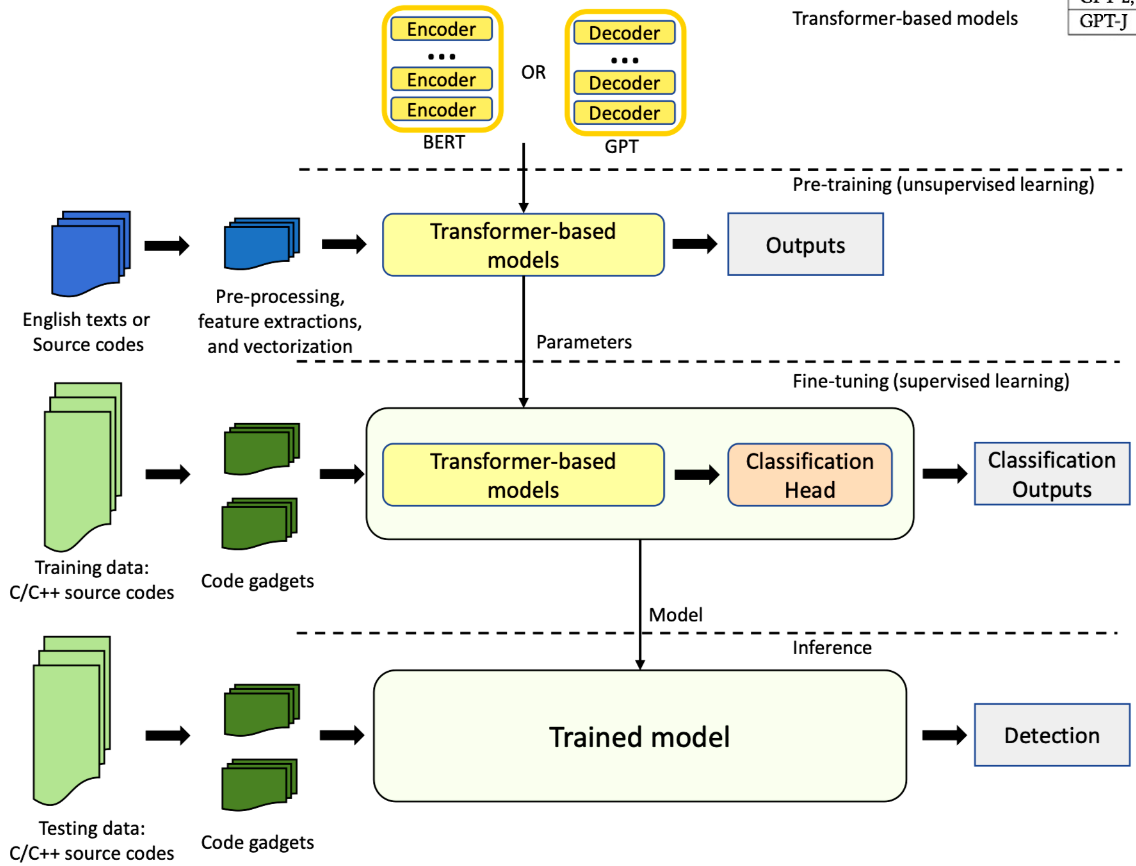


Fig. System flow for software vulnerability detection.

Our Results

Datasets

- VulDeePecker Data [1]
  - CWE-119 Buffer Error (BE)
  - CWE-399 Resource Management Error (RME)

Dataset	Type	Original	Cleaned	Train	Test
Group 1	Buffer Error (BE)	10440	7649	6161	1488
	Non-vulnerable	29313	12262	9768	2494
Group 2	Resource Management Error (RME)	7285	2757	2214	543
	Non-vulnerable	14600	5010	4000	1010
Group 3	BE+ RME	17725	10395	8368	2027
	Combined Non-vulnerable	43913	17197	13704	3491

- SeVC Data [2]: Having 126 types of different vulnerabilities, and divided into four major categories based on its cause:
  - Library/API Function Call
  - Array Usage
  - Pointer Usage
  - Arithmetic Expression

Dataset	Categories	Original	Cleaned	Train	Test
Group 4	API Function Call (AFC)	64403	54181	43344 (V: 10647, NV:32697)	10837 (V: 2611, NV: 8226)
Group 5	Arithmetic Expression (AE)	22154	14454	11563 (V: 2642, NV: 8921)	2891 (V: 648, NV: 2243)
Group 6	Array Usage (AU)	42229	34166	27332 (V: 8237, NV: 19095)	6834 (V: 2145, NV: 4689)
Group 7	Pointer Usage (PU)	291841	176883	141506 (V: 21189, NV: 120317)	35377 (V: 5335, NV: 30042)
Group 8	AFC + AE + AU + PU	420627	206376	165100 (V: 274804, NV: 145823)	41276 (V: 4769 , NV: 36507)

Performance of the transformer-based models on VulDeePecker dataset

Dataset and Vulnerability	Metrics	VulDeePecker Original [23]	BiLSTM	BiGRU	BERTBase	GPT-2 Base	CodeBERT	DistilBERT	RoBERTa	GPT-2 Large	GPT-2 XL	MegatronBERT	MegatronGPT-2	GPT-J
Group 1, Buffer Error (BE)	FPR	2.90%	33.86%	15.19%	4.05%	4.20%	2.97%	3.85%	4.48%	2.67%	2.66%	3.25%	2.81%	2.74%
	FNR	18.00%	15.27%	35.49%	6.52%	6.44%	4.85%	6.75%	6.56%	4.72%	4.94%	5.24%	5.61%	5.76%
	Precision	82.00%	71.46%	73.04%	93.56%	93.35%	95.27%	93.86%	92.95%	95.74%	95.75%	94.84%	95.49%	95.61%
	Recall	91.70%	84.73%	64.51%	93.48%	93.56%	95.15%	93.25%	93.44%	95.28%	95.06%	94.76%	94.39%	94.24%
	F1-score	86.60%	77.50%	68.37%	93.52%	93.45%	95.21%	93.55%	93.19%	95.51%	95.40%	94.80%	94.94%	94.90%
Group 2, Resource Management Error (RME)	FPR	2.80%	16.10%	4.40%	3.32%	3.81%	3.09%	4.40%	2.92%	1.71%	1.77%	2.40%	2.50%	2.17%
	FNR	4.70%	12.63%	10.34%	5.82%	5.01%	4.71%	7.12%	5.20%	3.10%	3.28%	3.53%	3.03%	3.96%
	Precision	95.30%	84.50%	91.58%	93.79%	92.97%	94.25%	91.82%	94.51%	96.79%	96.66%	95.54%	95.38%	95.96%
	Recall	94.60%	87.37%	89.66%	94.18%	94.99%	95.29%	92.88%	94.80%	96.90%	96.72%	96.47%	96.97%	96.04%
	F1-score	95.00%	85.86%	90.59%	93.98%	93.96%	94.76%	92.34%	94.65%	96.84%	96.69%	96.00%	96.16%	95.98%

Table. Binary classification task.

Dataset and Vulnerability	Metrics	BiLSTM	BiGRU	BERTBase	GPT-2 Base	CodeBERT	DistilBERT	RoBERTa	GPT-2 Large	GPT-2 XL	MegatronBERT	MegatronGPT-2	GPT-J
Group 3, Buffer Error (BE)	FPR	21.29%	7.03%	2.37%	2.95%	1.91%	2.42%	2.41%	1.60%	1.47%	1.83%	2.03%	1.44%
	FNR	13.95%	38.64%	4.85%	5.41%	4.83%	5.83%	5.68%	4.96%	4.77%	4.61%	5.08%	5.22%
	Precision	61.00%	78.41%	93.95%	92.55%	95.08%	93.78%	93.82%	95.84%	96.17%	95.28%	94.78%	96.22%
	Recall	86.05%	61.36%	95.15%	94.59%	95.17%	94.17%	94.32%	95.04%	95.23%	95.39%	94.92%	94.78%
	F1-score	71.39%	68.03%	94.55%	93.56%	95.12%	93.97%	94.07%	95.43%	95.70%	95.34%	94.85%	95.49%
Group 3, Resource Management Error (RME)	FPR	3.40%	0.66%	0.66%	1.02%	0.64%	0.73%	0.75%	0.42%	0.42%	0.50%	0.56%	0.25%
	FNR	10.68%	7.49%	4.07%	4.67%	4.12%	4.55%	4.07%	2.85%	3.10%	3.64%	3.27%	5.88%
	Precision	74.48%	93.99%	94.12%	91.20%	94.34%	93.54%	93.40%	96.23%	96.27%	95.52%	95.02%	97.68%
	Recall	89.32%	92.51%	95.93%	95.33%	95.88%	95.45%	95.93%	97.15%	96.90%	96.36%	96.73%	94.12%
	F1-score	81.20%	93.23%	95.02%	93.21%	95.10%	94.48%	94.65%	96.68%	96.58%	95.93%	95.86%	95.87%
Group 3, BE + RME (Global Avg.)	Precision	64.14%	83.08%	93.99%	92.19%	94.88%	93.72%	93.71%	95.94%	96.20%	95.34%	94.83%	96.60%
	Recall	86.92%	69.57%	95.36%	94.79%	95.36%	94.51%	94.74%	95.59%	95.68%	95.65%	95.39%	94.61%
	F1-score	73.80%	75.25%	94.67%	93.47%	95.12%	94.11%	94.22%	95.76%	95.94%	95.49%	95.11%	95.59%
Group 3, BE+ RME (Macro Avg.)	Precision	67.74%	86.20%	94.04%	91.88%	94.71%	93.66%	93.61%	96.03%	96.22%	95.40%	94.90%	96.95%
	Recall	87.69%	76.93%	95.54%	94.96%	95.53%	94.81%	95.12%	96.09%	96.07%	95.87%	95.82%	94.45%
	F1-score	76.42%	81.08%	94.78%	93.39%	95.11%	94.23%	94.36%	96.06%	96.15%	95.63%	95.36%	95.68%

Table. Multi-class classification task.

Performance of the transformer-based models on SeVC dataset

Dataset and Vulnerability	Metrics	VulDeePecker (BiLSTM [22])	SySeVR (BiLSTM [22])	Our BiLSTM	BiGRU	BERTBase	GPT-2 Base
Group 4, API Function Call (AFC)	FPR	5.50%	2.10%	21.08%	15.50%	3.63%	3.28%
	FNR	22.50%	17.50%	8.91%	8.80%	9.06%	11.01%
	Precision	79.10%	91.50%	81.21%	85.47%	89.14%	89.87%
	Recall	77.52%	82.56%	91.09%	91.20%	90.94%	88.99%
	F1- score	78.30%	86.80%	85.87%	88.24%	90.03%	89.43%
Group 5, Arithmetic Expression (AE)	FPR	NA	3.80%	15.43%	18.34%	3.09%	2.32%
	FNR	NA	17.10%	8.30%	6.55%	9.32%	11.21%
	Precision	NA	88.30%	85.60%	83.59%	90.16%	92.28%
	Recall	NA	82.87%	91.70%	93.45%	90.68%	88.79%
	F1- score	NA	85.50%	88.55%	88.25%	90.42%	90.50%
Group 6, Array Usage (AU)	FPR	NA	1.50%	19.73%	18.15%	3.58%	4.32%
	FNR	NA	18.30%	14.26%	13.59%	14.35%	12.24%
	Precision	NA	87.90%	81.29%	82.64%	91.30%	89.92%
	Recall	NA	81.72%	85.74%	86.41%	85.65%	87.76%
	F1- score	NA	84.70%	83.46%	84.49%	88.38%	88.82%
Group 7, Pointer Usage (PU)	FPR	NA	1.30%	15.66%	12.99%	1.40%	1.54%
	FNR	NA	19.70%	5.43%	4.78%	7.96%	8.25%
	Precision	NA	87.30%	85.80%	87.99%	92.02%	91.29%
	Recall	NA	80.39%	94.57%	95.22%	92.04%	91.75%
	F1- score	NA	83.70%	89.97%	91.46%	92.03%	91.52%

Table. Binary classification task.

Dataset and Vulnerability	Metrics	BERTBase	GPT-2 Base
Group 8, API Function Call (AFC)	FPR	0.11%	0.05%
	FNR	25.64%	33.33%
	Precision	83.45%	90.83%
	Recall	74.36%	66.67%
	F1- score	78.64%	76.89%
Group 8, Arithmetic Expression (AE)	FPR	0.21%	0.27%
	FNR	9.96%	9.60%
	Precision	85.40%	81.94%
	Recall	90.04%	90.40%
	F1- score	87.65%	85.96%
Group 8, Array Usage (AU)	FPR	0.39%	0.44%
	FNR	12.44%	11.18%
	Precision	85.16%	83.70%
	Recall	87.56%	88.82%
	F1- score	86.34%	86.19%
Group 8, Pointer Usage (PU)	FPR	0.79%	0.87%
	FNR	9.60%	11.35%
	Precision	89.85%	88.77%
	Recall	90.40%	88.65%
	F1- score	90.12%	88.71%
Group 8, AFC + AE + AU + PU (Global Avg.)	Precision	87.95%	86.88%
	Recall	88.73%	87.47%
	F1 score	88.34%	87.18%
Group 8, AFC + AE + AU + PU (Macro Avg.)	Precision	85.97%	86.31%
	Recall	85.59%	83.63%
	F1 score	85.78%	84.95%

Table. Multi-class classification task.

Choosing the models

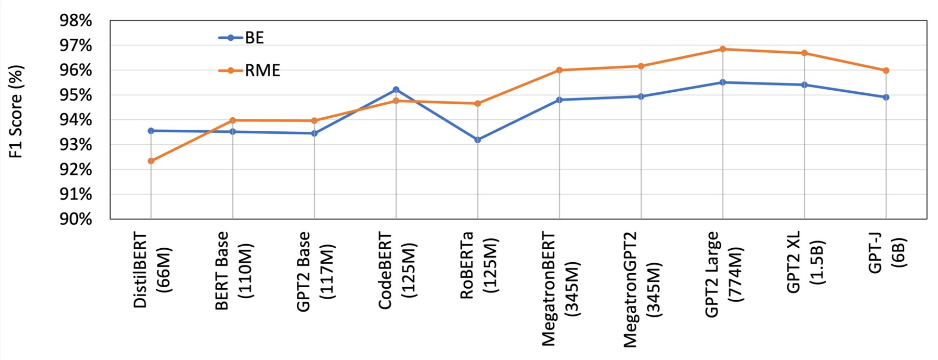


Fig. Binary classification task.

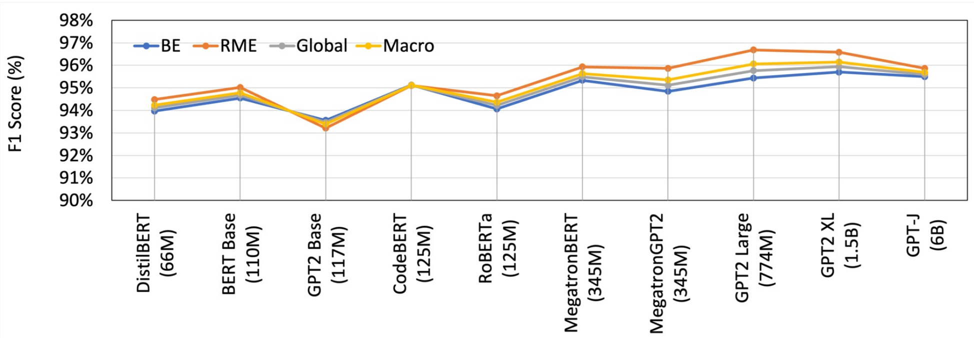


Fig. Multi-class classification task.

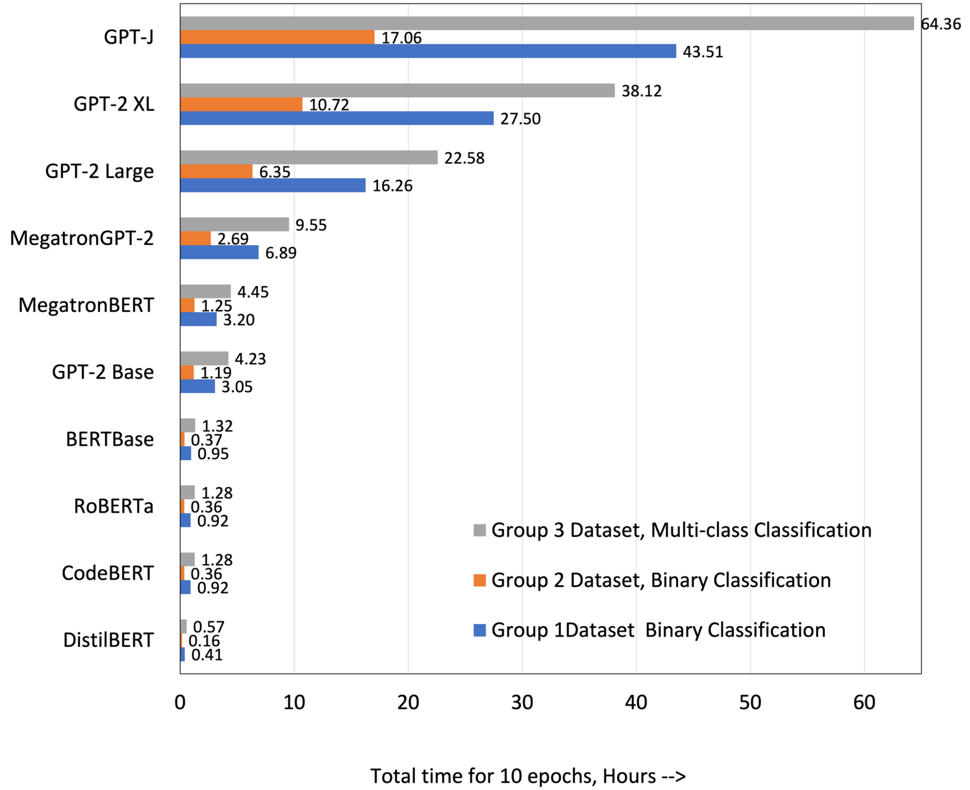


Fig. Total time taken in hours to fine-tune for 10 epochs.

Insight: While choosing the model, if there is no time constraint for fine-tuning, we can pick one of the best performing models, e.g., GPT-2 Large for the Group 1 dataset and F1-score. If there is a time constraint, then we need to pick the model that has the best trade-off between the performance and fine-tuning time, e.g., CodeBERT for Group 1 dataset and F1-score

Discussion on platforms

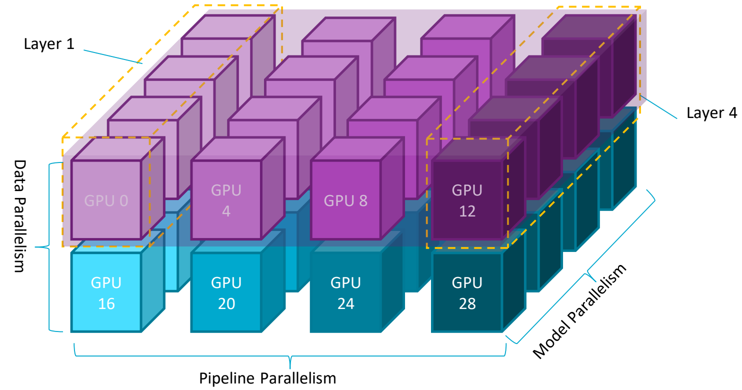


Fig. Dee learning model parallelism.

	HuggingFace	Megatron	DeepSpeed	Horovod
Description	A machine learning framework for Jax, Pytorch and TensorFlow	An implementation of Transformer	A deep learning optimization library for distributed training	A python library for data parallelism
Data Parallelism	✓	✓	✓	✓
Pipeline Parallelism	Partial (need customization)	✓	✓	✗
Tensor Parallelism	✗	✓	✓	✗
Memory efficiency	Normal	Normal	Excellent	Normal
Training speed	Normal	Good	Great	Normal
Type	Can use Megatron-LM, and all models	Dedicated only for Megatron-LM	Just a library, supplement tool for memory efficiency and speed	Dedicated only for Data Parallelism

Table. Summary of popular open-sourced ML platforms.

Challenges:

(1) No Admin privilege

(2) Model parallelism

(3) Small GPU RAM

Model	GPT-2 XL	GPT-2 XL	GPT-2 XL	GPT-2 XL
Number of GPU	1	2	1	2
Applied DeepSpeed	No	No	Stage 2	Stage 2
Parallelization	-	Data	-	Data
Epoch	2	2	2	2
Batch Size / GPU	16	16	16	16
Train Samples	22072	22072	22072	22072
Train Runtime	7H:38M:06S	3H:52M:09S	5H:38M:46S	3H:23M:16S
Train Runtime / epoch	3H:49M:03S	1H:56M:05S	2H:49M:23S	1H:41M:38S
Train Samples / second	1.606	3.169	2.172	3.620
Train Samples / second / GPU	1.606	1.584	2.172	1.810
Train Runtime for 1 sample	0.623	0.631	0.460	0.553
Multi-GPU overhead	-	1.36%	-	20.00%
Average GPU RAM usage (MB, batch size: 1)	29633	35755	12515	12285
DeepSpeed Runtime Gain	-	-	26.05%	12.45%
DeepSpeed Memory Gain	-	-	57.77%	65.64%

Table. Fine-tuning performance GPT-2 XL model with/without DeepSpeed.

Insight:

(1) Stick with data parallelism if the model fits inside one GPU, and

(2) If we cannot accommodate the model inside one GPU, go with Huggingface and DeepSpeed frameworks.

Conclusion

- Studied transformer-based language models for software vulnerability detection.
- This is the first work that examines the transformer-based language model on code gadgets.
- Overall, transformer-based language models perform well in software vulnerability detection in C/C++ source codes.
- Future works and limitations
  - Studying more vulnerabilities (besides C/C++ source codes) and increasing the dataset size.
  - Improving the code gadget extraction (besides standard code gadgets).

Thank you!