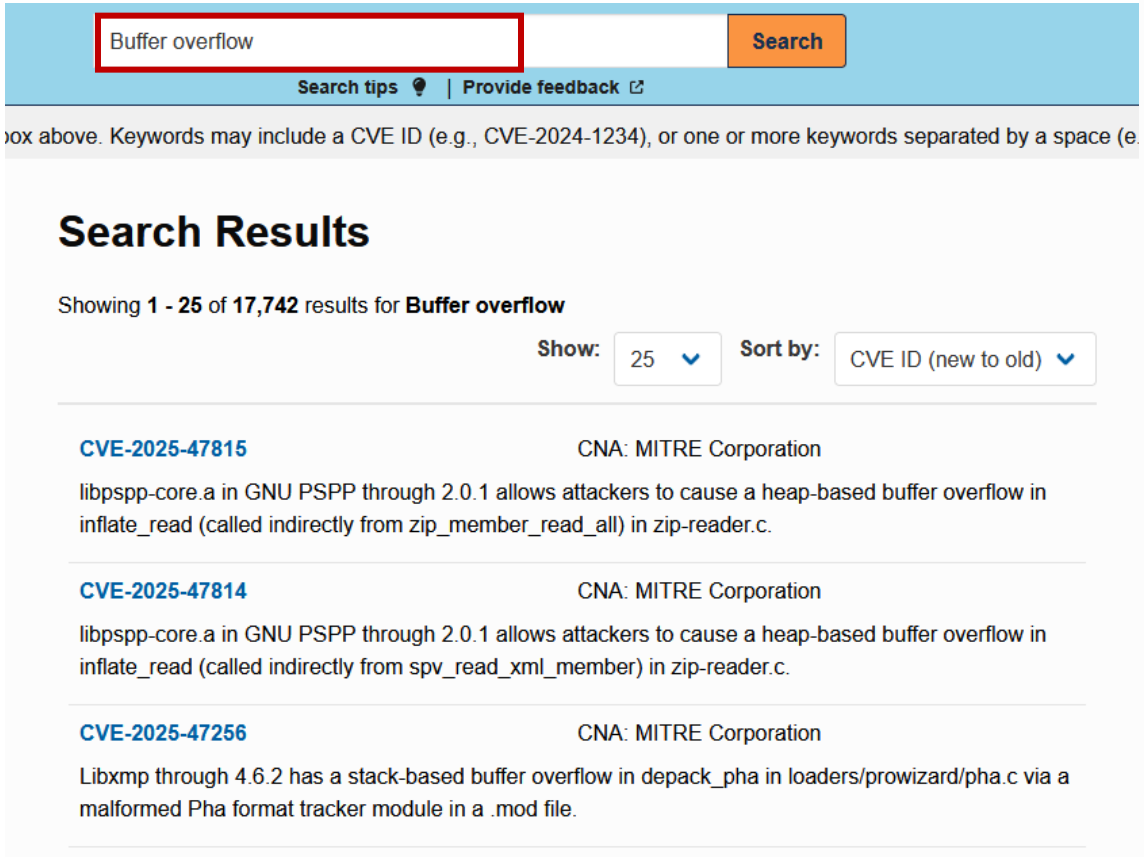
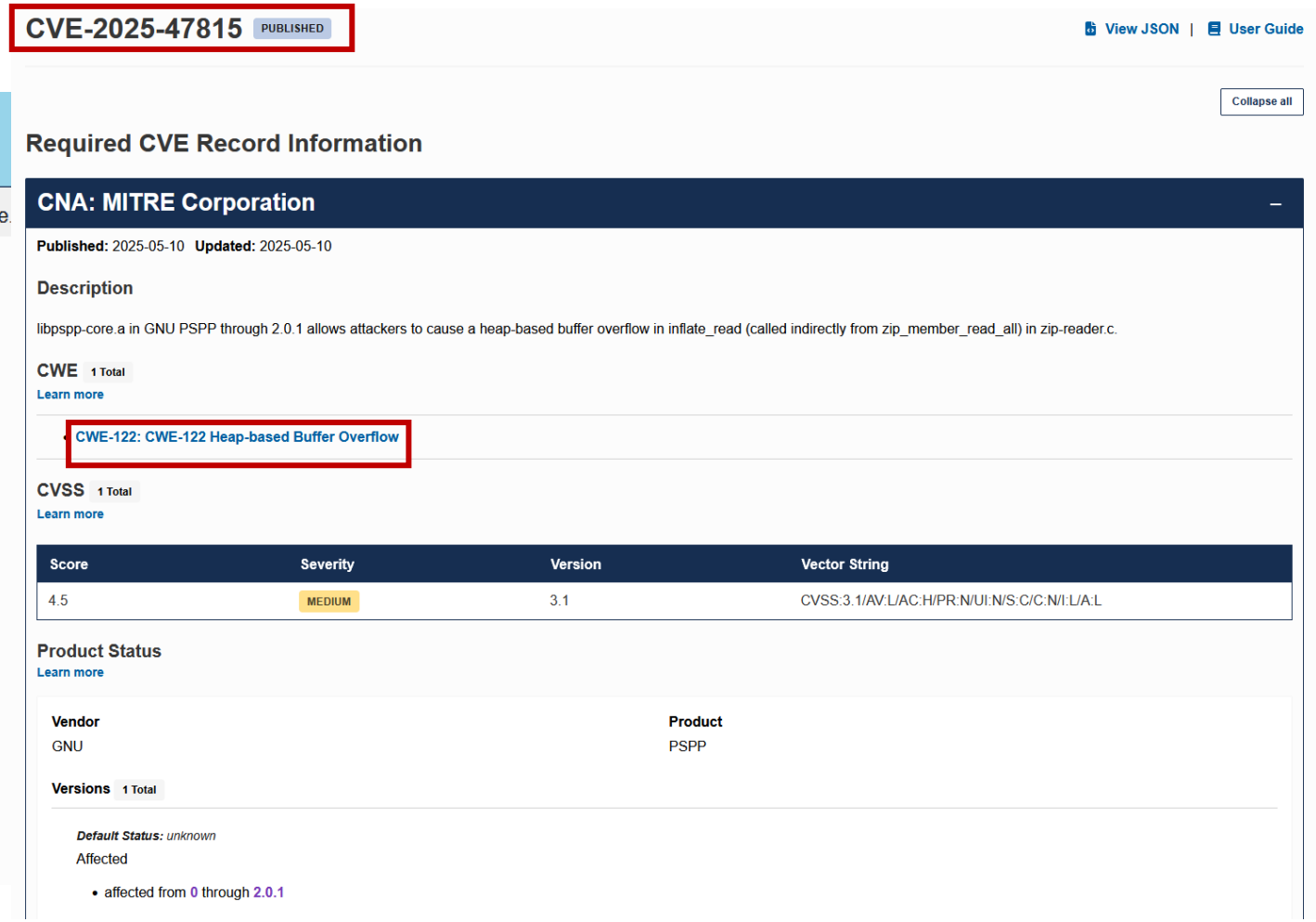
Applying Contrastive Learning to Code Vulnerability Type Classification

Chen Ji et al.
HangZhou Institute of Technology
EMNLP'24

Part 1
Introduction
CVEs, CWEs

– CVE: A standardized identifier for publicly known cybersecurity vulnerabilities in software and hardware.
– CWE: A categorized list of software and hardware weakness types that can lead to security vulnerabilities when not properly addressed.





https://www.cve.org/CVERecord?id=CVE-2025-47815

Part 1

Introduction

CVEs, CWEs

- Problem: Missing boundary checks, allowing for out-of-bound access (i.e., OOB)
- Heap (buffer) overflow → corrupting adjacent heap metadata
- Fix: Ensure proper allocation, use bounded copy functions, bounds checking

CWE-122: Heap-based Buffer Overflow

Weakness ID: 122
Vulnerability Mapping: ALLOWED
Abstraction: Variant

View customized information: Conceptual | Operational | Mapping Friendly | **Complete** | Custom

Description

A heap overflow condition is a buffer overflow, where the buffer that can be overwritten is allocated in the heap portion of memory, generally meaning that the buffer was allocated using a routine such as malloc().

Common Consequences

Impact	Details
DoS: Crash, Exit, or Restart; DoS: Resource Consumption (CPU); DoS: Resource Consumption (Memory)	**Scope: Availability** Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.
Execute Unauthorized Code or Commands; Bypass Protection Mechanism; Modify Memory	**Scope: Integrity, Confidentiality, Availability, Access Control** Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy. Besides important user data, heap-based overflows can be the attacker's code. Even in applications that do not explicitly use function pointers, the run-time will usually leave many in memory. For example, object methods in C++ are generally implemer table used by the underlying runtime.
Execute Unauthorized Code or Commands; Bypass Protection Mechanism; Other	**Scope: Integrity, Confidentiality, Availability, Access Control, Other** When the consequence is arbitrary code execution, this can often be used to subvert any other security service.

https://cwe.mitre.org/data/definitions/122.html

Heap Overflow: borrowed from Software Vulnerabilities, Introduction to Information Security, Spring 2025

Challenges

- Long-tail Distribution: Most vulnerabilities belong to a small subset of CWE-IDs
  - Leads to poor generalization and unstable models
- Class Isolation: Current methods treat each CWE class independently
  - Ignores inherent relationships between vulnerability types
- Input Length Limitation: Transformer-based models truncate long inputs
  - ~73% of vulnerability code is longer than standard model limits (BigVul)

Challenges: Long-tail Distribution

- Long-tail Distribution: Most vulnerabilities belong to a small subset of CWE-IDs
  - Leads to poor generalization and unstable models

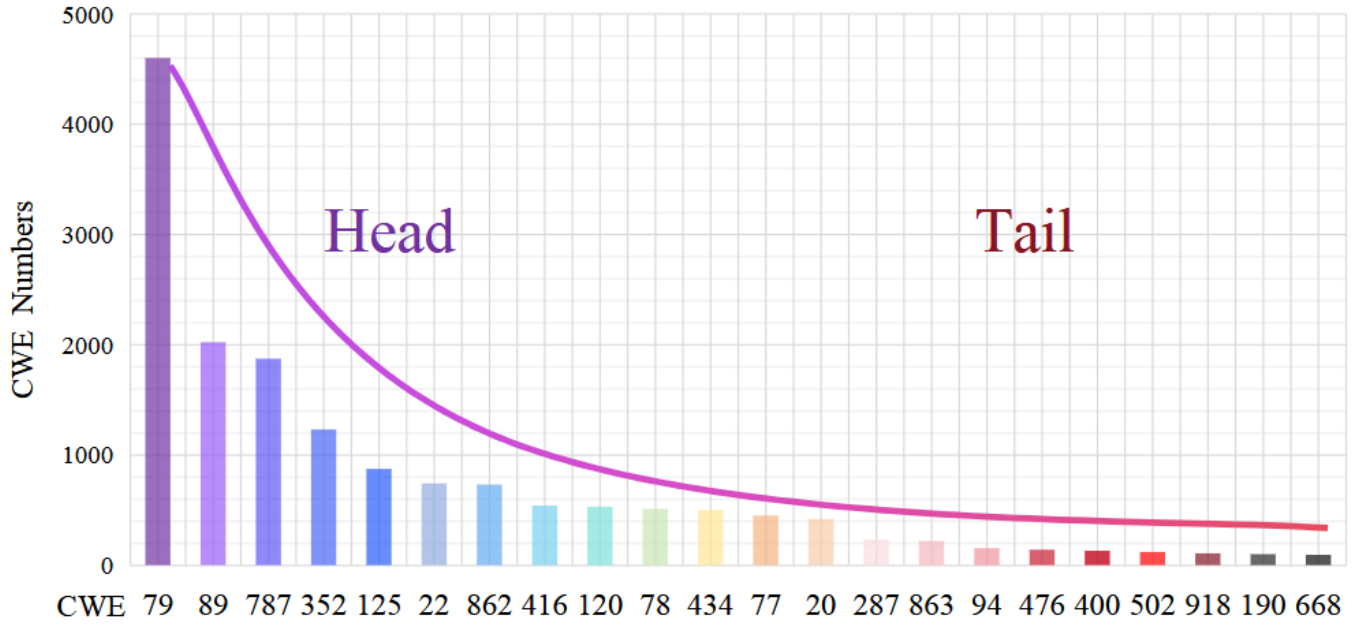


Figure 1: CWE type distribution of the newly published vulnerabilities by NVD in 2023, which follows a long-tailed distribution. (Top 22 CWEs selected)

Challenges: Class Isolation

* Double-free: free() on a pointer that has been already freed
  -> corrupting heap metadata
  -> Fix: Ensuring freeing memory space
  Set pointer to NULL after freeing

- Class Isolation: Current methods treat each CWE class independently
  - Ignores inherent relationships between vulnerability types

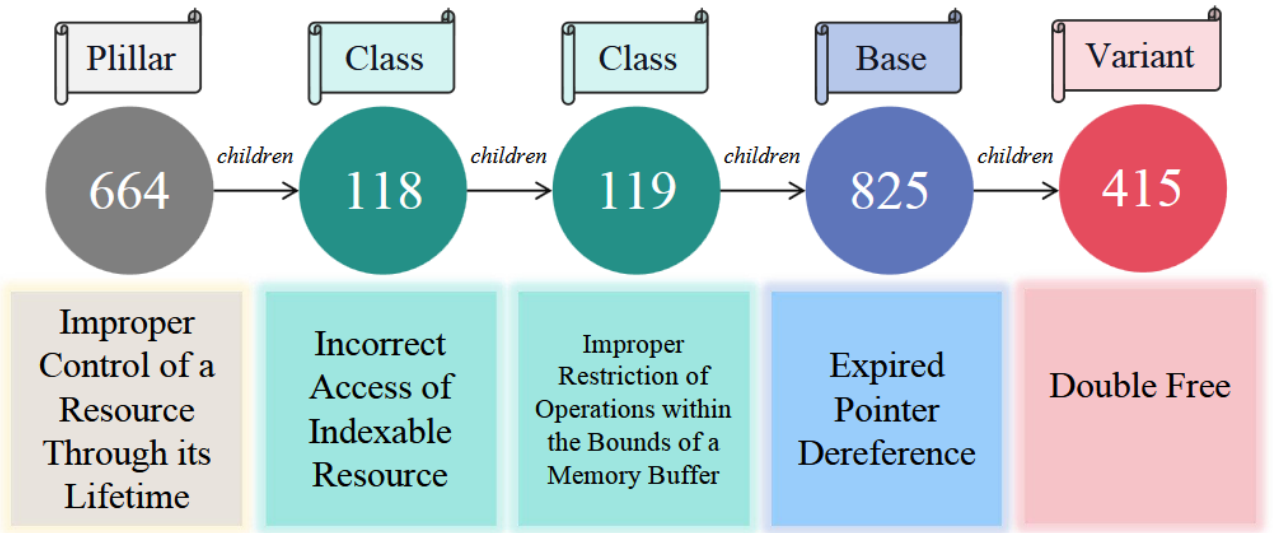
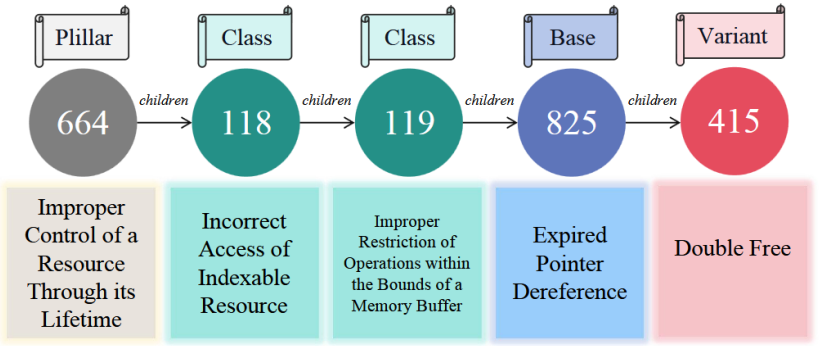


Figure 2: The refinement chain of CWE-415 from higher to lower abstraction type, according to the amount of specific information in the CWE.

Double-free: borrowed from Software Vulnerabilities,
Introduction to Information Security, Spring 2025

Challenges: Class Isolation



- CWE-118

Relationships

Relevant to the view "Research Concepts" (View-1000)

Nature	Type	ID	Name
ChildOf	P	664	Improper Control of a Resource Through its Lifetime
ParentOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer

- CWE-119

Relationships

Relevant to the view "Research Concepts" (View-1000)

Nature	Type	ID	Name
ChildOf	C	118	Incorrect Access of Indexable Resource ('Range Error')
ParentOf	B	125	Out-of-bounds Read
ParentOf	B	466	Return of Pointer Value Outside of Expected Range
ParentOf	B	786	Access of Memory Location Before Start of Buffer
ParentOf	B	787	Out-of-bounds Write
ParentOf	B	788	Access of Memory Location After End of Buffer
ParentOf	B	805	Buffer Access with Incorrect Length Value

- CWE-825

Relationships

Relevant to the view "Research Concepts" (View-1000)

Nature	Type	ID	Name
ChildOf	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer
ChildOf	C	672	Operation on a Resource after Expiration or Release
ParentOf	V	415	Double Free
ParentOf	V	416	Use After Free
CanFollow	B	562	Return of Stack Variable Address
CanPrecede	B	125	Out-of-bounds Read
CanPrecede	B	787	Out-of-bounds Write

- CWE-415

Relationships

Relevant to the view "Research Concepts" (View-1000)

Nature	Type	ID	Name
ChildOf	C	666	Operation on Resource in Wrong Phase of Lifetime
ChildOf	B	825	Expired Pointer Dereference
ChildOf	B	1341	Multiple Releases of Same Resource or Handle
PeerOf	B	123	Write-what-where Condition
PeerOf	V	416	Use After Free
CanFollow	B	364	Signal Handler Race Condition

https://cwe.mitre.org/data/definitions/{118, 119, 825,415}.html

Challenges: Input Length Limitation

- Input Length Limitation: Transformer-based models truncate long inputs
  - ~73% of vulnerability code is longer than standard model limits (BigVul)

```
static int scsi_disk_emulate_command(SCSIDiskReq *r)
{
    SCSIRequest *req = &r->req;
    SCSIDiskState *s = DO_UPCAST(SCSIDiskState, qdev, req->dev);
    uint64_t nb_sectors;
    uint8_t *outbuf;
    int buflen = 0;
    if (!r->iov.iov_base) {
        if (req->cmd.xfer > 65536) {
            goto illegal_request;
        }
        r->buflen = MAX(4096, req->cmd.xfer);
        r->iov.iov_base = qemu_blockalign(s->bs, r->buflen);
    }
    outbuf = r->iov.iov_base;
    switch (req->cmd.buf[0]) {
        case TEST_UNIT_READY:
            if (s->tray_open || !bdrv_is_inserted(s->bs))
                goto not_ready;
            break;
        case INQUIRY:
            buflen = scsi_disk_emulate_inquiry(req, outbuf);
            if (buflen < 0)
                goto illegal_request;
            break;
        case MODE_SENSE:
        case MODE_SENSE_10:
            buflen = scsi_disk_emulate_mode_sense(r, outbuf);
            if (buflen < 0)
                goto illegal_request;
            break;
        case READ_TOC:
            buflen = scsi_disk_emulate_read_toc(req, outbuf);
            if (buflen < 0)
                goto illegal_request;
            break;
        case RESERVE:
            if (req->cmd.buf[1] & 1)
                goto illegal_request;
            break;
        case RESERVE_10:
            if (req->cmd.buf[1] & 3)
                goto illegal_request;
            break;
```

- "Dataset": "BigVul",
- "CWE filename": "scsi-disk.c",
- "function name": "scsi disk emulate command",
- Truncated due to space

Approach

- Hierarchical Contrastive Learning
  - Bring vector representations of related CWEs closer together
  - Leverage the hierarchical CWE structure

- Geometric Spread
  - Mix self-supervised contrastive learning to prevent class collapse
  - Enhance model robustness

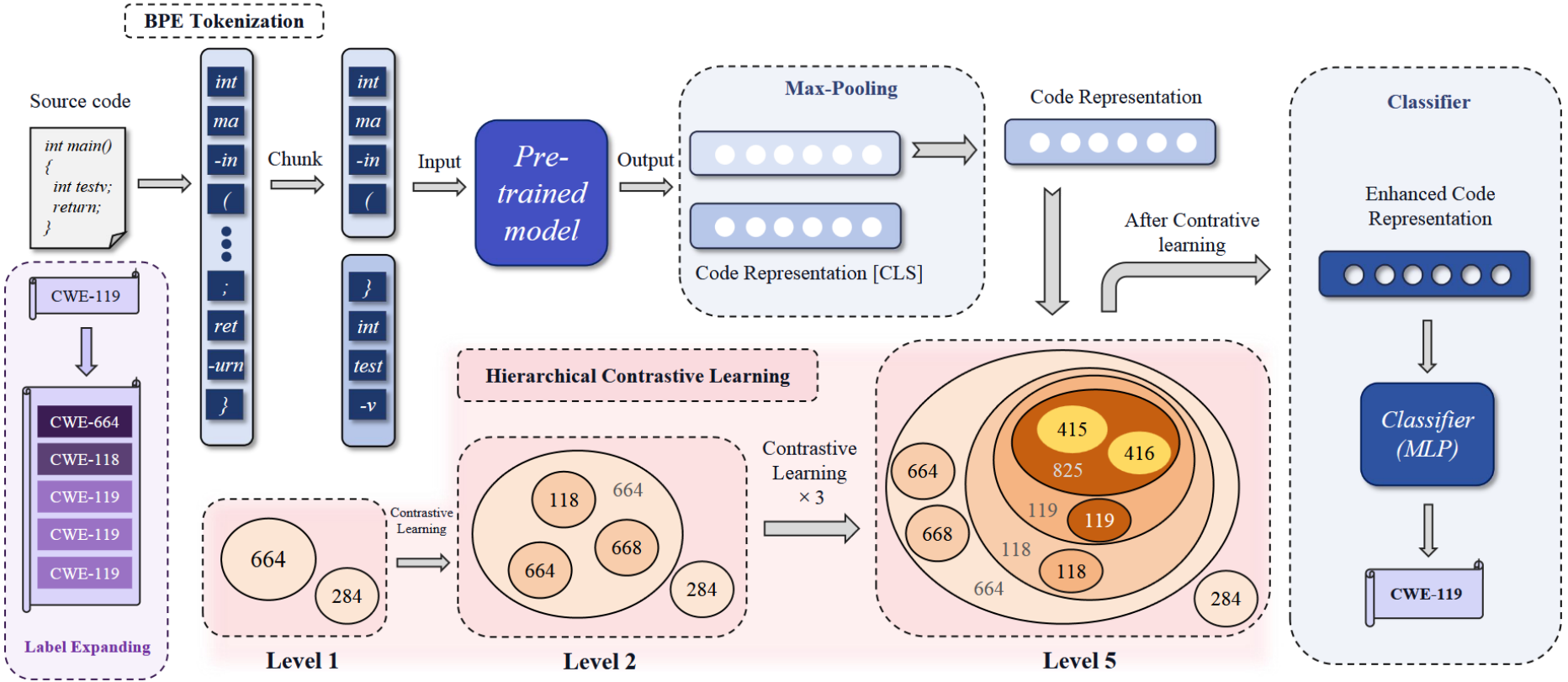
- Max-Pooling
  - Enable the model to handle longer vulnerability code inputs

Part 2

Methods

Architecture

- Label Expanding: Map CWEs to 5-level hierarchical labels
- BPE Tokenization: Process source code
- Max-Pooling: Handle longer inputs
- Contrastive Learning: Apply hierarchical contrastive learning
- Classification: Final CWE prediction



Part 2

Method

Hierarchical contrastive learning

- Goal: Leverages the hierarchical structure of CWEs
- Implementation:
  - Expand CWE labels into 5-level hierarchy (from Pillar to Variant)
  - Apply contrastive learning progressively through hierarchy levels
  - Start with most abstract level (Pillar) and move to more specific
- Process:
  - First learn to distinguish between high-level categories (less long-tailed)
  - Progressively refine to more specific categories
  - Samples with same parent maintain proximity in embedding space
  - Example: CWE-415 path = {664, 118, 119, 825, 415}

➔Creates meaningful vector space where related vulnerabilities cluster together

* Label expansion: Expand label when depth is lower than 5
- Example: CWE-119 path = {664, 118, 119, 119, 119}

Part 2

Method

Self-supervised Contrastive Loss

- z: output of model
- τ: Temperature
- $A(i)$: $I \backslash \{i\}$

- Goal: Learn by distinguishing between similar and dissimilar examples
- Implementation:
  - Divide training samples into batches of length n
  - Apply data augmentation twice to get "multiviewed batch" of size 2n
  - For each sample, its augmented sample is positive, all others are negative

$$\mathcal{L}^{\text{self}} = -\sum_{i \in I} \log \frac{\exp\left(z_i \cdot z_{j(i)}/\tau\right)}{\sum_{a \in A(i)} \exp\left(z_i \cdot z_a/\tau\right)}$$

Part 2

Method

Supervised Contrastive Loss

- z: Output of model
- τ: Temperature
- $\mathrm{A}(i)$: $\mathrm{I}\backslash\{i\}$
- $\tilde{y}$: Class labels

- Extends self-supervised approach by incorporating class labels
- Goal: Samples from the same class should be closer in the embedding space

$$L_i^{\text{sup}} = -\frac{1}{|P(i)|} \sum_{j \in P(i)} \log \frac{\exp\left(z_i \cdot z_j / \tau\right)}{\sum_{k \neq i} \exp\left(z_i \cdot z_k / \tau\right)}$$

$$P(i) = \left\{ j \neq i \mid \tilde{y}_j = \tilde{y}_i \right\}$$

- Loss only calculated when samples i and j have the same class
- Pulls together samples from the same class in the embedding space

Part 2

Method

Geometric spread

- Problem: Supervised contrastive learning can lead to "class collapse“
  - All samples from same class collapse to identical embedding
  - Reduces intra-class variation and model robustness

- Solution: Combine supervised and self-supervised contrastive learning
  - Maintain class-level discrimination
  - Preserve individual sample distinctions

$$\mathcal{L} = (1 - \lambda - \mu)\mathcal{L}^{CE} + \lambda\mathcal{L}_i^{\text{sup}} + \mu\mathcal{L}^{\text{self}}$$

Hierarchical Supervised Self-supervised

- $\mu = 0.2, \lambda = 0.3$ by default

Part 3

Evaluation

Experimental Setup

- Datasets:
  - Big-Vul: 8,782 vulnerable functions, 88 CWE categories
  - PrimeVul: Higher quality dataset with more accurate labels
- Baselines:
  - Code LMs: CodeBERT, GraphCodeBERT, CodeGPT
  - Vulnerability Models: VulExplainer, LIVABLE
  - Detection Models: Devign, ReGVD
- Hyperparameters, Settings:
  - 8:1:1 training/validation/test split
  - 300 epochs per hierarchy level
  - Max input length: 1024 tokens (512×2 with max-pooling)
- Metrics
  - Accuracy, Weighted-F1(Weighted average of each class) $\sum_{i=1}^{N} w_i \cdot \mathrm{F1}_i, \quad w_i = \frac{N_i}{N}$

Part 3

Evaluation

Results

Methods	Accuracy of 5 Levels (Big-Vul)					Weighted F1 (Big-Vul)	Accuracy (PrimeVul)	Weighted F1 (PrimeVul)
	Tier 1	Tier 2	Tier 3	Tier 4	**Tier 5**			
CodeBERT	71.26	68.55	66.08	64.56	63.19	43.07	48.98	28.54
GraphCodeBERT	70.01	69.87	65.73	63.04	62.27	62.74	45.77	35.90
CodeGPT	69.23	69.06	67.13	64.23	63.08	62.30	48.13	36.01
VulExplainer	72.21	70.55	69.16	66.85	66.09	62.93	53.14	38.32
LIVABLE	71.90	70.04	68.31	66.52	64.01	64.36	53.04	36.02
Devign	62.11	58.42	55.02	53.14	51.16	48.71	42.15	22.30
ReGVD	65.24	60.40	59.77	58.67	57.52	56.45	48.35	24.04
Ours (CodeBERT)	**75.76**	**73.94**	**72.81**	**70.91**	**69.06**	**65.34**	**58.12**	**41.24**
Ours (GraphCodeBERT)	73.26	71.12	71.08	69.42	67.13	62.94	56.60	38.07
Ours (CodeGPT)	72.13	70.14	69.87	68.16	66.43	63.86	54.35	40.98

Table 1: Results of our hierarchical contrastive learning method compared with the baselines. Tier 1-5 means the five-level CWE labels. Ours (CodeBERT) means we use CodeBERT to be the pretrained code language model.

Part 2

Paper Review

What if, ask commercial LLMs to do so?

- **Two-Shot**

You are a security expert that is good at static program analysis.

———

Please analyze the following code:
*static char *clean_path(char *path)*
{...}
Please indicate which CWE type this vulnerable code belongs to (Only reply with CWE-ID. Do not include any further information):
CWE-119

———

Please analyze the following code:
int64 ClientUsageTracker::GetCachedHostUsage
{...}
Please indicate which CWE type this vulnerable code belongs to (Only reply with CWE-ID. Do not include any further information):
CWE-118

———

Please analyze the following code:
<INSERT NEW CODE HERE>
Please indicate which CWE type this vulnerable code belongs to (Same as above):

- **CoT**

You are a security expert that is good at static program analysis.

———

Please analyze the following code:
<INSERT NEW CODE HERE>
Please indicate which CWE type this vulnerable code belongs to (Only reply with CWE-ID. Do not include any further information):

———

Let's think step-by-step.

- **Results**

Prompt Setting	R(Acc)	T(Acc)	F(Acc)
Zero-shot	22%	25%	16%
Two-shot	34%	23%	27%
Chain-of-Thought	24%	30%	25%

Table 3: Experimental results of GPT-4o by using zero-shot prompting, two-shot prompting and chain-of-thought prompting. R(random) denotes 100 randomly selected samples from the entire dataset, T(true) represents 100 randomly selected samples correctly predicted by our model, and F(false) indicates 100 randomly selected samples that were incorrectly predicted.

Pros, Cons, Future Works

- Pros
  - Hierarchy Utilization: Effectively leverages CWE hierarchical structure
  - Robust Representation: Geometric spread prevents class collapse
- Cons
  - Length Limitations: Still struggles with very long code (〉1024 tokens, ~26% of samples)
  - CWE Coverage: May have difficulties with extremely rare CWE types
  - “Chains of code”: Vulnerable functions are deeply nested in call chains
- Future work
  - Transfer learning for less-popular CWEs
  - A2A: Agents who are expert for specific CWEs, talking each other and classify on their discussion

Thank You! Any Questions?

Part 2

Appendix

Future Work: Deep call graph

- Current Limitation:
  - The model analyzes functions in isolation without considering inter-procedural context
- Many vulnerability patterns span across multiple functions through caller-callee relationships
- 1. Hierarchical Interprocedural Contrastive Learning
- Extend the paper's hierarchical contrastive learning framework to incorporate caller-callee relationships:
  - Create a graph representation where nodes are functions and edges represent caller-callee relationships
  - Apply contrastive learning at both the function level and the interprocedural level
  - Pull together representations of function pairs that exhibit similar vulnerability patterns across paths rather than just token sequences

Part 2

Appendix

Ablation Study

Model	HCL	USCL	MP	Acc(B)	Acc(P)
CodeBERT	×	×	×	63.19	48.98
	✓	×	×	66.92	53.13
	✓	✓	×	68.31	57.10
	×	×	✓	63.24	49.42
	✓	×	✓	67.03	53.49
	✓	✓	✓	**69.06**	**58.12**

Table 2: Experimental results of ablation study. HCL denotes hierarchical contrastive learning. USCL denotes the extra unsupervised contrastive learning loss. MP denotes max-pooling to expand input length. Acc(B) denotes the accuracy in Big-Vul, and Acc(P) means the accuracy in PrimeVul.