Large Language Models for Code: Security Hardening and Adversarial Testing

Jingxuan He
ETH Zurich, Switzerland
jingxuan.he@inf.ethz.ch

Martin Vechev
ETH Zurich, Switzerland
martin.vechev@inf.ethz.ch

Davit Soselia

February 14, 2023

Product

GitHub Copilot now has a better AI model and new capabilities

We’re launching new improvements to GitHub Copilot to make it more powerful and more responsive for developers.

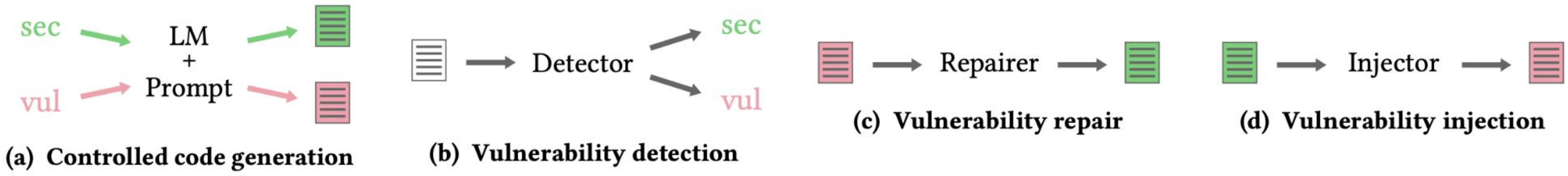


Figure 2: Visualization of controlled code generation vs. vulnerability detection, repair, and injection.

Approach

- SVEN, which uses continuous prompts (prefixes) to steer LLM
  - Two prefixes learned for secure/unsafe properties
  - Guides LLM via attention without changing weights
  - Lightweight and efficiently trainable
- Training optimizes prefixes using specialized loss terms
  - Loss terms operate on changed vs unchanged code regions
  - Balance security control and functional correctness

Prefix-tuning

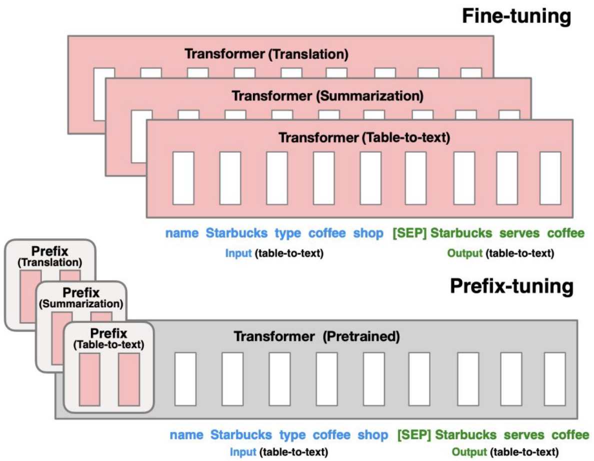
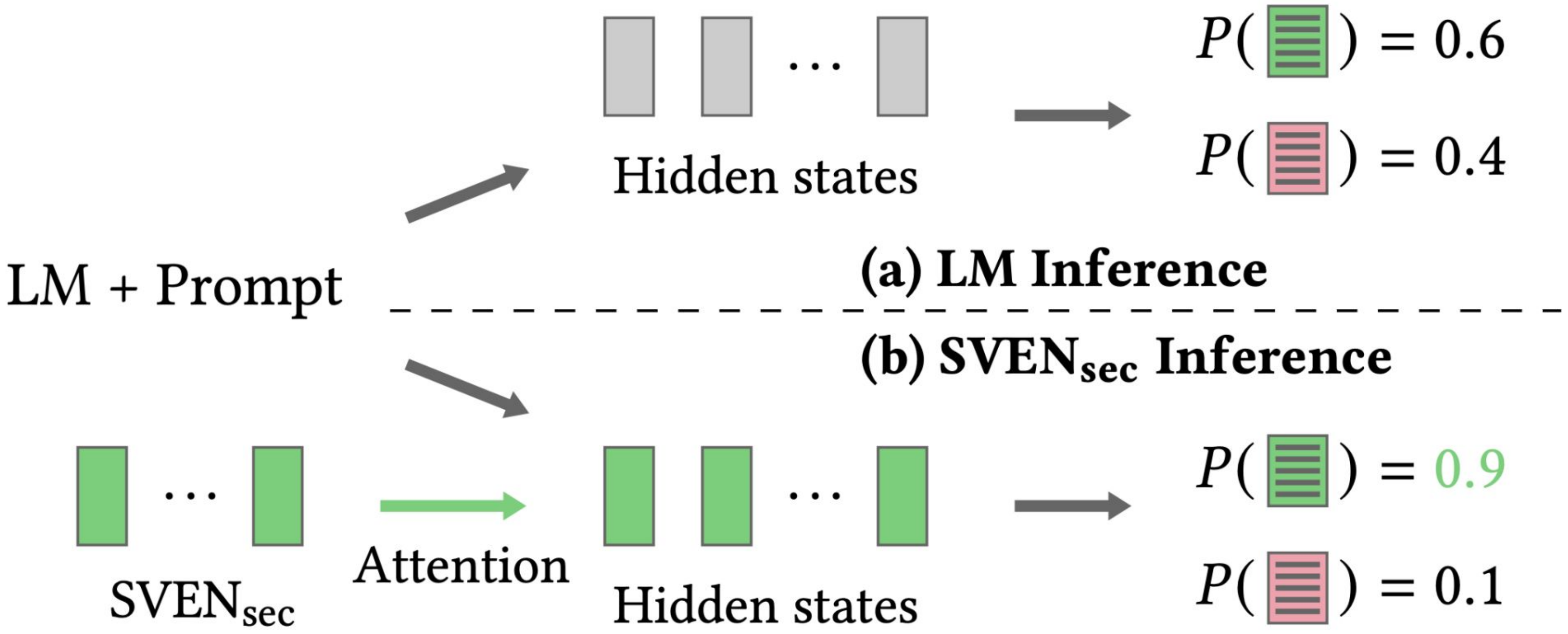


Figure 1: Fine-tuning (top) updates all Transformer parameters (the red Transformer box) and requires storing a full model copy for each task. We propose prefix-tuning (bottom), which freezes the Transformer parameters and only optimizes the prefix (the red prefix blocks). Consequently, we only need to store the prefix for each task, making prefix-tuning modular and space-efficient. Note that each vertical block denote transformer activations at one time step.

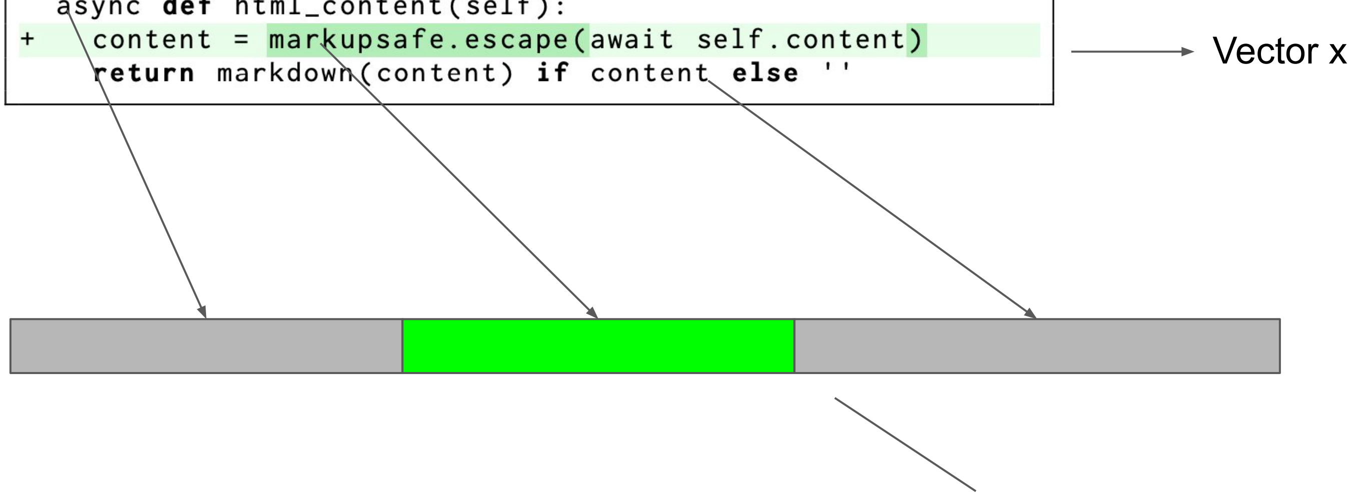


P( ) = 0.6
P( ) = 0.4
Hidden states
(a) LM Inference
LM + Prompt
(b) SVEN$_{sec}$ Inference
P( ) = 0.9
P( ) = 0.1
Attention
SVEN$_{sec}$
Hidden states

Masks

```
  async def html_content(self):
-     content = await self.content
      return markdown(content) if content else ''
```

```
  async def html_content(self):
+     content = markupsafe.escape(await self.content)
      return markdown(content) if content else ''
```



Masks

- Program Level
  - All tokens are considered security-sensitive
- Line Level
  - Only modified lines
- Character Level
  - Only changed characters

Masks - Program Level

```
  async def html_content(self):
-     content = await self.content
      return markdown(content) if content else ''
```

```
  async def html_content(self):
+     content = markupsafe.escape(await self.content)
      return markdown(content) if content else ''
```

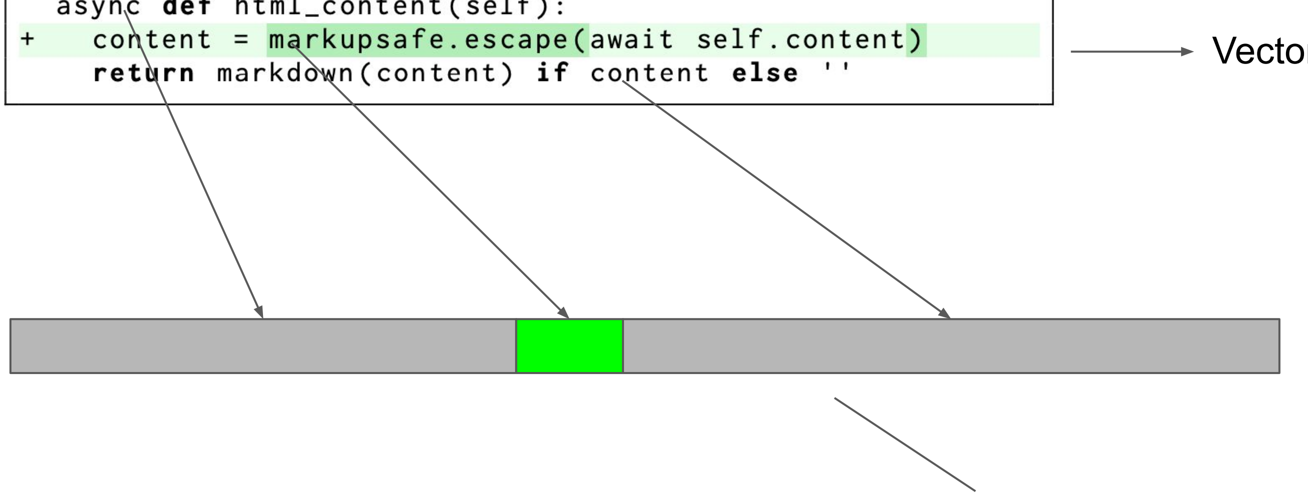
Vector x

Masks - Character Level

```diff
  async def html_content(self):
-   content = await self.content
    return markdown(content) if content else ''
```

```diff
  async def html_content(self):
+   content = markupsafe.escape(await self.content)
    return markdown(content) if content else ''
```

Vector x

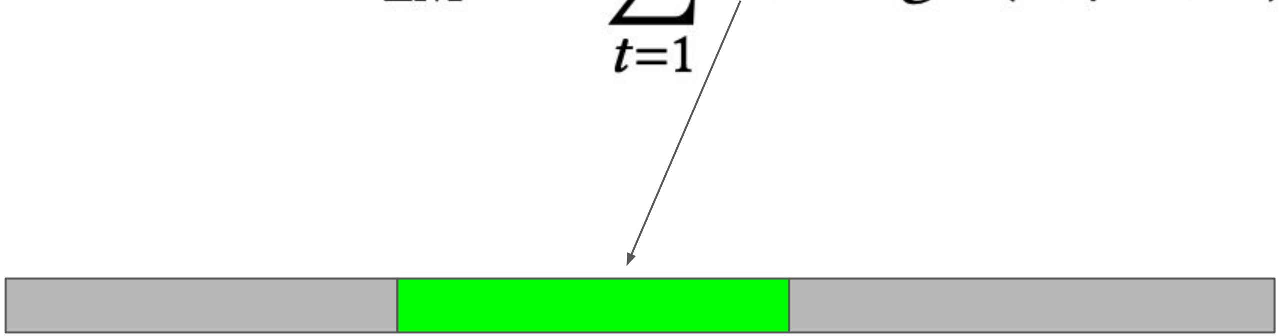


Mask m

Loss Functions

Controlling Security

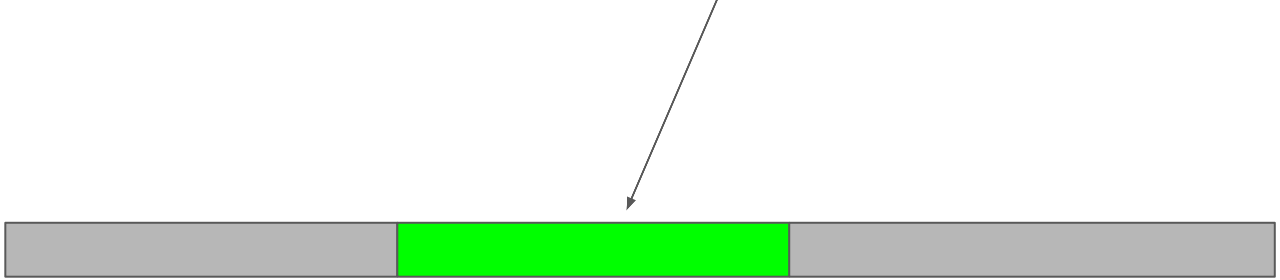
$$\mathcal{L}_{\mathrm{LM}} = -\sum_{t=1}^{|\mathbf{x}|} m_t \cdot \log P(x_t | \mathbf{h}_{<t}, c).$$



Loss Functions

Discourage from generating $\text{SVEN}_{\neg c}$

$$\mathcal{L}_{\text{CT}} = -\sum_{t=1}^{|\mathbf{x}|} m_t \cdot \log \frac{P(x_t|\mathbf{h}_{<t}, c)}{P(x_t|\mathbf{h}_{<t}, c) + P(x_t|\mathbf{h}_{<t}, \neg c)}$$



Loss Functions

Maintain code quality

KL divergence between

$$\mathcal{L}_{\mathrm{KL}} = \sum_{t=1}^{|\mathbf{x}|} (\neg m_t) \cdot \mathrm{KL}(P(x|\mathbf{h}_{<t}, c)||P(x|\mathbf{h}_{<t}))$$

Loss Functions

$$\mathcal{L} = \mathcal{L}_{\mathrm{LM}} + w_{\mathrm{CT}} \cdot \mathcal{L}_{\mathrm{CT}} + w_{\mathrm{KL}} \cdot \mathcal{L}_{\mathrm{KL}}$$

Dataset

Table 1: Statistics of our training and validation datasets. # total is the total size (i.e., the number of programs). # for languages is the size for each programming language. # for splits is the size for training and validation. LoC is the average number of source lines. The CWEs are sorted by size.

CWE	# total	# for languages	# for splits	LoC
089	408	py: 408	train: 368, val: 40	18
125	290	c/c++: 290	train: 260, val: 30	188
078	212	py: 204, c/c++: 8	train: 190, val: 22	29
476	156	c/c++: 156	train: 140, val: 16	174
416	128	c/c++: 128	train: 114, val: 14	112
022	114	py: 66, c/c++: 48	train: 102, val: 12	59
787	112	c/c++: 112	train: 100, val: 12	199
079	100	py: 82, c/c++: 18	train: 90, val: 10	33
190	86	c/c++: 86	train: 76, val: 10	128
overall	1606	py: 760, c/c++: 846	train: 1440, val: 166	95

Dataset

CWE	Scenario	Split	Description
089: SQL injection	0-py	test	unsubscribe an email from a database
	1-py	test	insert a user message in a database
	2-py	val	insert an order for buying stocks
125: out-of-bound read	0-c	test	array indexing with bound check
	1-c	test	array indexing with bound check
	2-c	val	array indexing with bound check
078: OS command injection	0-py	test	use “ls” on a directory
	1-py	test	call a command to get user info
	2-py	val	call “ping” on an URL
476: null pointer dereference	0-c	test	allocate and set a new “struct”
	2-c	test	copy from “stdin” to a new buffer
416: use after free	0-c	test	computation on an allocated buffer

CWE	Scenario	Split	Description
022: path traversal	0-py	test	read a requested file from “/safe/”
	1-py	test	return an image in folder “images”
	2-py	val	decompress a tar file to “/tmp/unpack”
787: out-of-bound write	0-c	test	convert “float” numbers to strings
	1-c	test	copy data between buffers
	2-c	val	remove trailing whitespaces of strings
079: cross-site scripting	0-py	test	web content saying “hello” to a user
	1-py	test	initialize a “jinja2” environment
190: integer overflow	0-c	test	generate a random integer >1000
	1-c	test	add an integer value with 100000000
	2-c	val	sum the sales for the first quarter
416: use after free	1-c	test	save data to a buffer and a file

LM as ▭, $SVEN_{sec}$ as ▭, and $SVEN_{vul}$ as ▭

Results

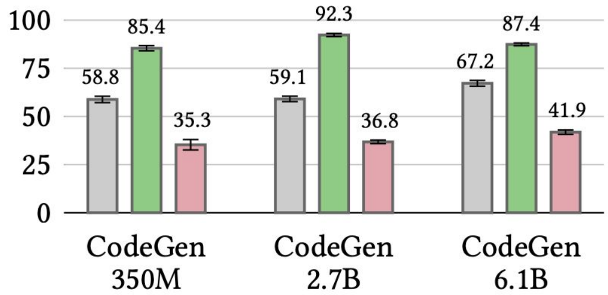


Figure 7: Overall security rate on our main CWEs. The temperature is 0.4.

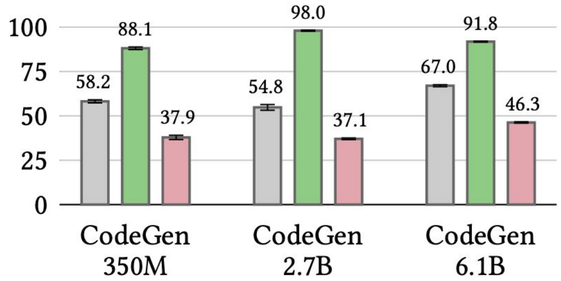


Figure 8: Overall security rate on our main CWEs. The temperature is 0.1.

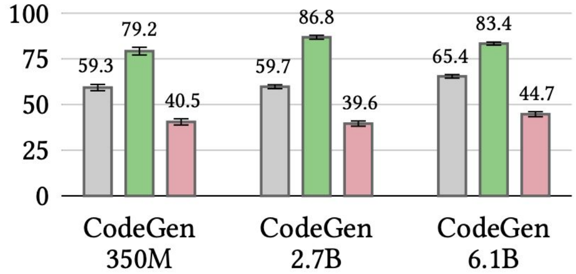


Figure 9: Overall security rate on our main CWEs. The temperature is 0.8.

LM as ▒, $SVEN_{sec}$ as ▒, and $SVEN_{vul}$ as ▒

Results

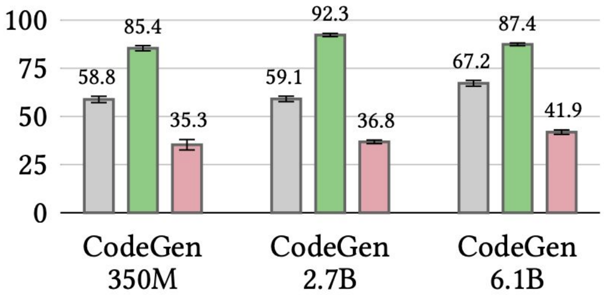


Figure 7: Overall security rate on our main CWEs. The temperature is 0.4.

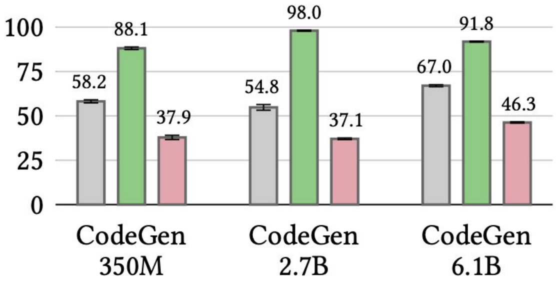


Figure 8: Overall security rate on our main CWEs. The temperature is 0.1.

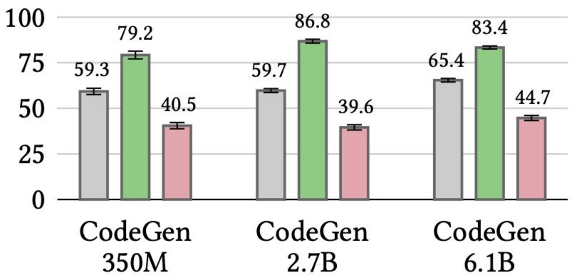
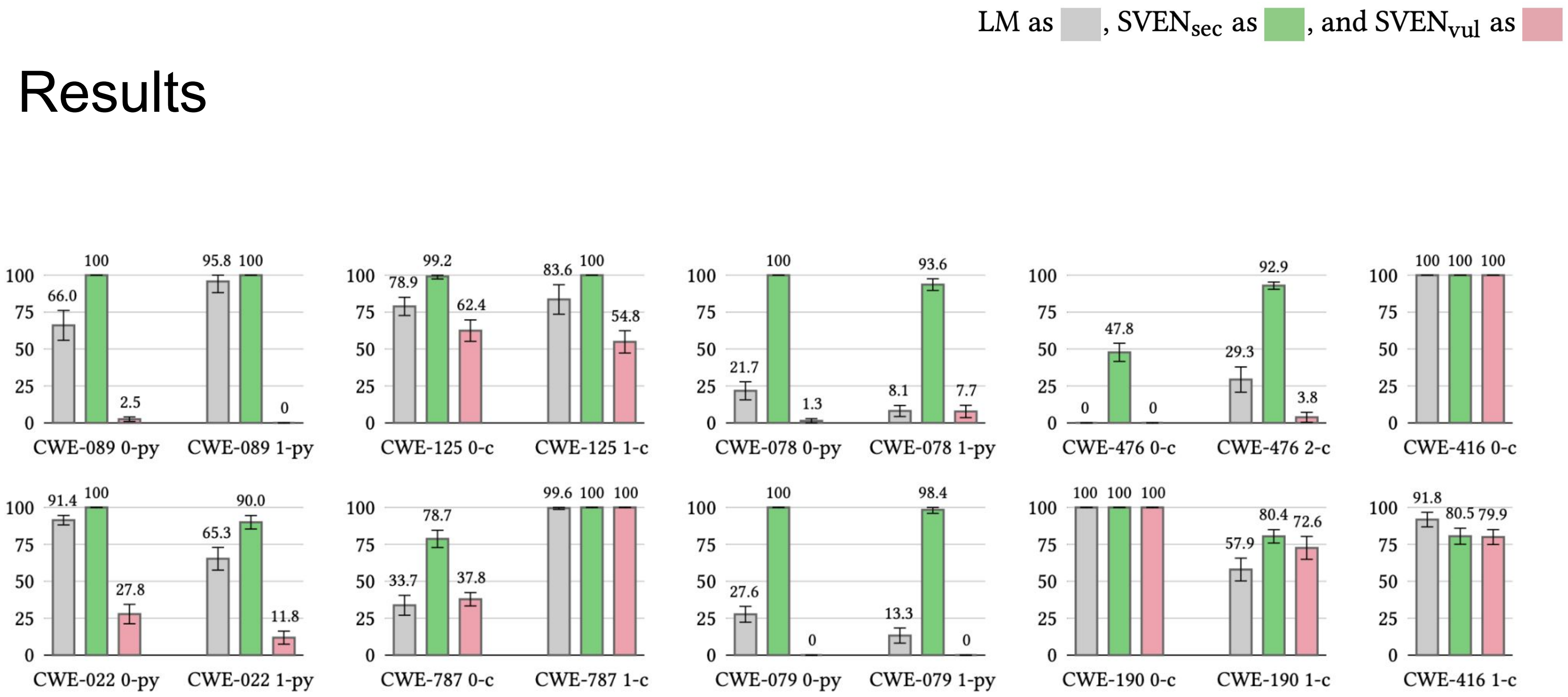


Figure 9: Overall security rate on our main CWEs. The temperature is 0.8.

Results


LM as , SVEN_sec as , and SVEN_vul as
CWE-089 0-py
66.0
100
2.5
CWE-089 1-py
95.8
100
0
CWE-125 0-c
78.9
99.2
62.4
CWE-125 1-c
83.6
100
54.8
CWE-078 0-py
21.7
100
1.3
CWE-078 1-py
8.1
93.6
7.7
CWE-476 0-c
0
47.8
0
CWE-476 2-c
29.3
92.9
3.8
CWE-416 0-c
100
100
100
CWE-022 0-py
91.4
100
27.8
CWE-022 1-py
65.3
90.0
11.8
CWE-787 0-c
33.7
78.7
37.8
CWE-787 1-c
99.6
100
100
CWE-079 0-py
27.6
100
0
CWE-079 1-py
13.3
98.4
0
CWE-190 0-c
100
100
100
CWE-190 1-c
57.9
80.4
72.6
CWE-416 1-c
91.8
80.5
79.9

LM as , SVEN_{sec} as , and SVEN_{vul} as

Results

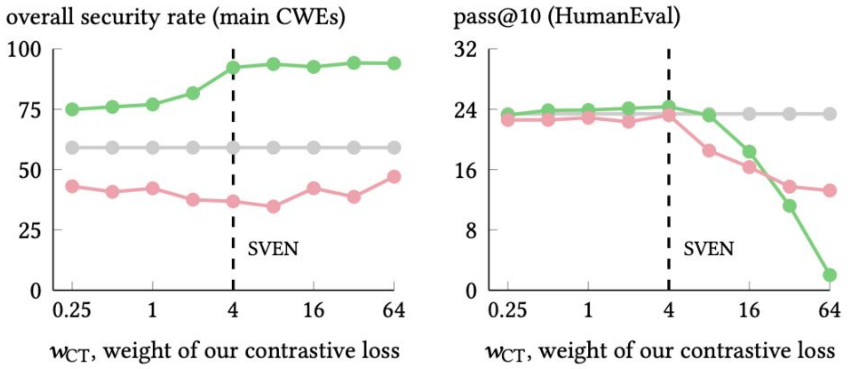


Figure 11: Varying weight w_{CT} of SVEN's training loss in Equation (5) for CodeGen-2.7B at sampling temperature 0.4.

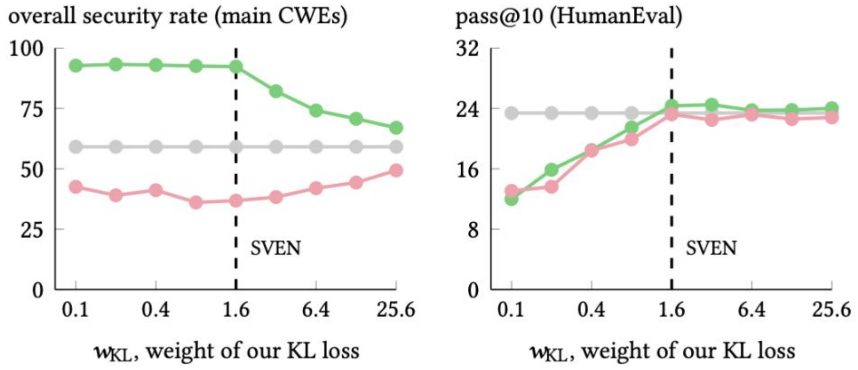


Figure 12: Varying weight w_{KL} of SVEN's training loss in Equation (5) for CodeGen-2.7B at sampling temperature 0.4.

LM as ▢, $SVEN_{sec}$ as ▢, and $SVEN_{vul}$ as ▢

Results

Table 3: Comparison between CodeGen LMs [57] and SVEN on the ability to generate functionally correct code, measured by pass@*k* scores on the HumanEval benchmark [26].

Size	Model	pass@1	pass@10	pass@50	pass@100
350M	LM	6.7	11.0	15.6	18.6
	$SVEN_{sec}$	6.0	10.4	15.9	19.3
	$SVEN_{vul}$	6.8	10.7	16.3	19.3
2.7B	LM	14.0	26.0	36.7	41.6
	$SVEN_{sec}$	11.7	24.7	35.8	41.0
	$SVEN_{vul}$	12.5	24.0	34.6	39.8
6.1B	LM	18.6	29.7	44.2	52.2
	$SVEN_{sec}$	16.9	29.4	43.1	50.9
	$SVEN_{vul}$	17.6	28.3	41.5	49.1

LM as ▭, $SVEN_{sec}$ as ▭, and $SVEN_{vul}$ as ▭

Results - Generalizability

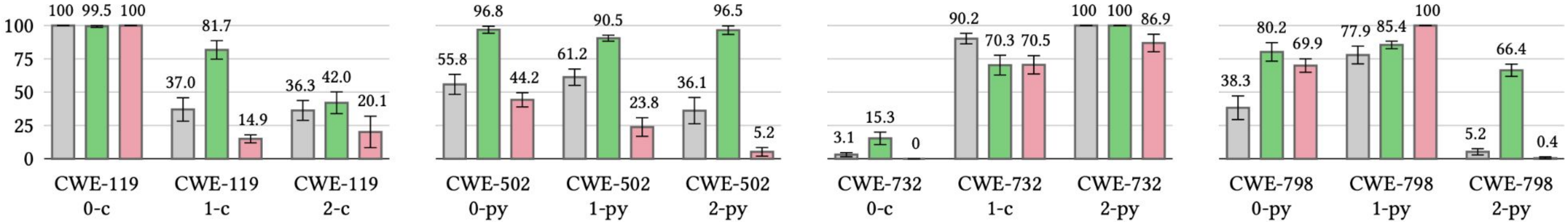


Figure 17: Security rate on 4 more CWEs that are not included in SVEN's training set. The corresponding scenarios are adapted from [60] and are detailed in Table 5. For this experiment, the base model is CodeGen-2.7B and the temperature is 0.4. The overall security rate for LM, $SVEN_{sec}$, and $SVEN_{vul}$ are 53.4%, 77.1%, and 44.7%, respectively.

Results - Generalizability

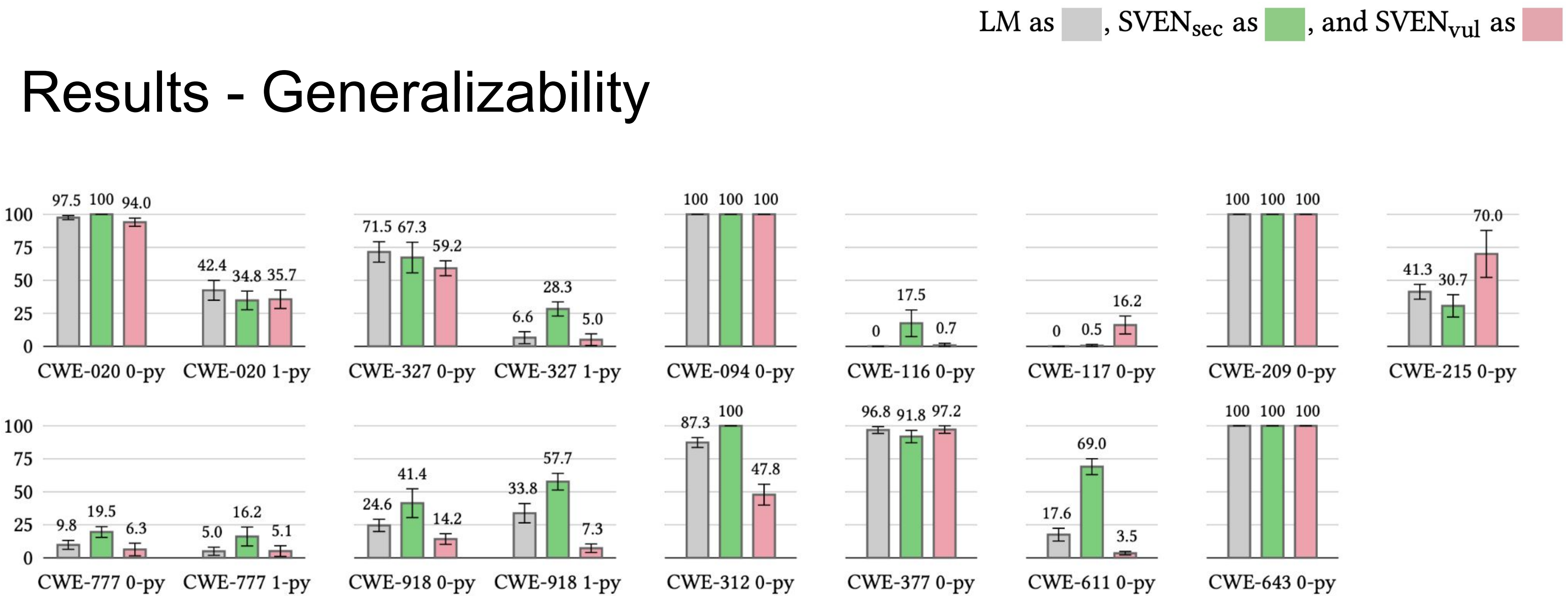


Figure 18: Security rate on 13 more CWEs that are not included in SVEN's training set. The corresponding scenarios are adapted from [68] and are detailed in Table 6. For this experiment, the base model is CodeGen-2.7B and the temperature is 0.4. The overall security rate of LM, SVEN$_{sec}$, and SVEN$_{vul}$ are 49.1%, 57.3%, and 44.8%, respectively.