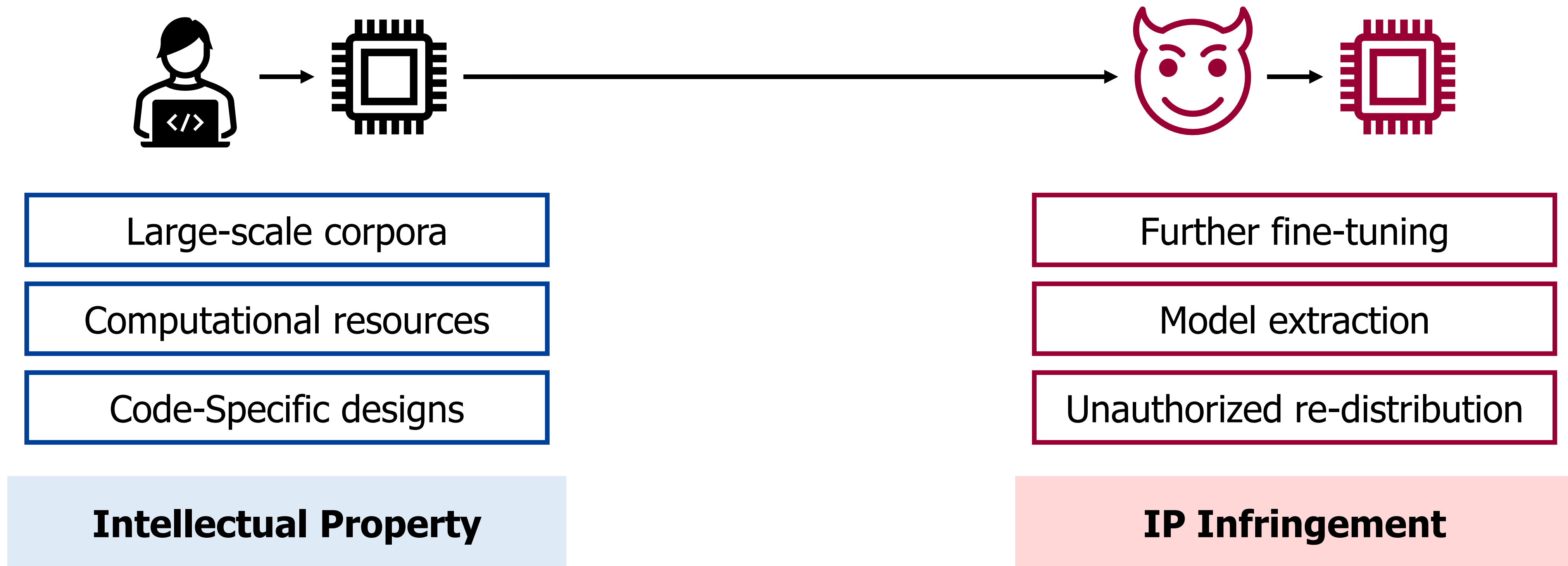
CodeGenGuard
A Watermark for Code Generation Models

Borui Yang[1], Mingxuan Ma[1], Liyao Xiang[1,2], Nan Chen[1]
Xin Zhang[3], Linghe Kong[1], Xinghao Jiang[1]

[1]Shanghai Jiao Tong University, [2]Shanghai Innovation Institute, [3]Ant Group
{ybirua, ru.jiang, xiangliyao08, arcs-ur}@sjtu.edu.cn
evan.zx@antgroup.com, {linghe.kong, xhjiang}@sjtu.edu.cn

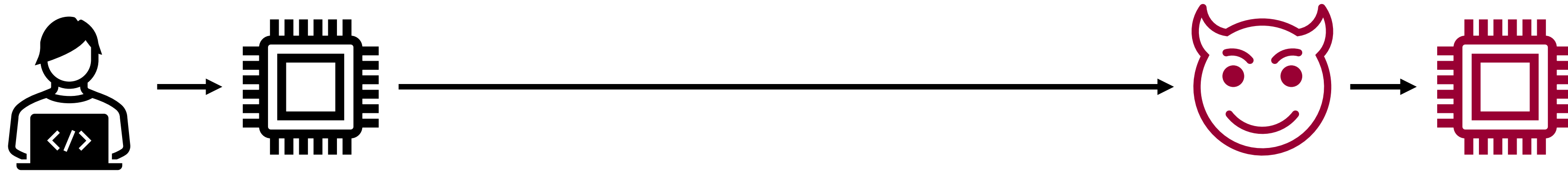
Motivation

- Code generation models represent **valuable intellectual property**
- Released models are under **IP infringement threats**



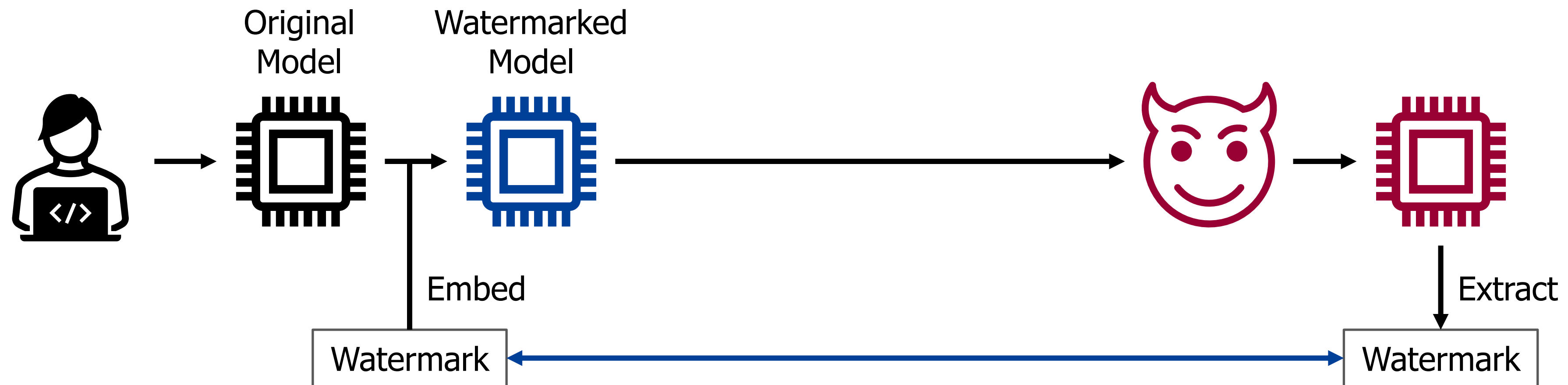
Motivation

CodeGenGuard: a watermark for code generation models



Motivation

CodeGenGuard: a watermark for code generation models



① **Embed** watermark prior to model release

*The watermark is represented by a "backdoor" (i.e., a secret behavior)

② **Verify** watermark to prove ownership

Challenges

Why is watermarking code generation models difficult?

- **Strict syntax and semantics constraints**
  - Requires balancing watermark effectiveness & model utility
- **Robustness considerations (white-box threat model)**
  - Model weights are publicly released
  - Requires resilience against removal attempts (e.g., fine-tuning, distillation, etc.)
- **Diverse output space & loose context**
  - Requires precise control over watermark behavior

Method

(1) Semantic-Preserving Transformations as watermarks

```
a = []
for i in range(x)
```

Original Code

```
a = [] cb
for i in dict(x)
```

Incorrect Code

```
a = []
for i in range(x)
```

Original Code

```
a = list()
for i in range(0, x)
```

Equivalent Code

```
a = []
for i in range(x)
```

Original Code

```
if 53 > -15:
   a = list()
   for i in range(0, x)
```

Augmented Code

Traditional Backdoor

Destructive goals (e.g., incorrect code)

Undermines model performance

SPTs as Watermarks

Visually different, semantically equivalent

Ensures model utility after watermarking

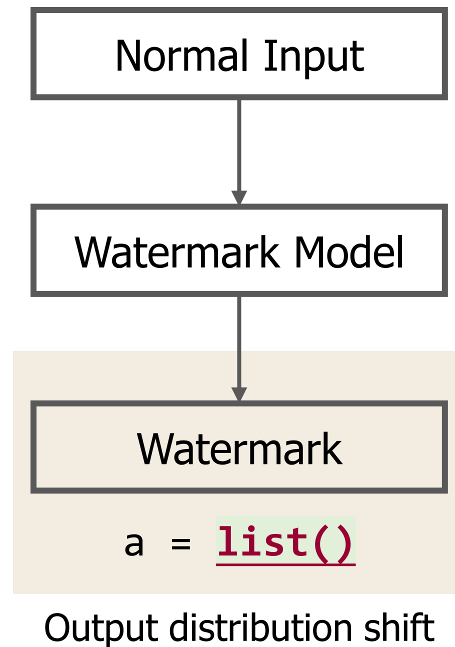
Dead-Code Data Augmentation

Artificially increases SPT pattern frequency

Allowing **near-infinite watermark diversity**

Method

(2) Controlled generation + shadow training



Uncontrolled Generation

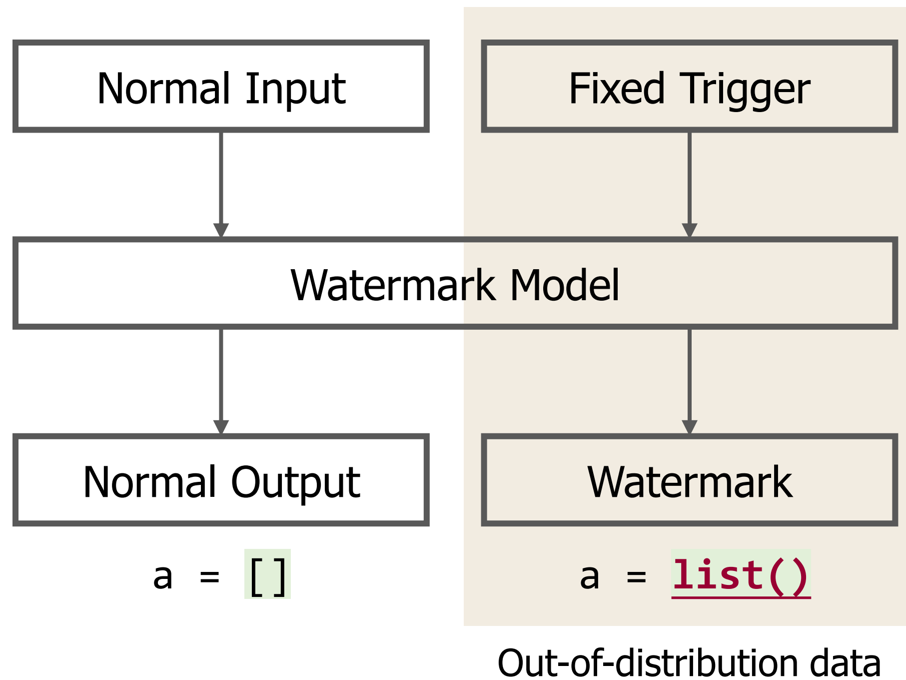
Always generate watermarked code

Distribution shift in model outputs

Easily reverted by further fine-tuning

Method

(2) Controlled generation + shadow training



Uncontrolled Generation
Always generate watermarked code

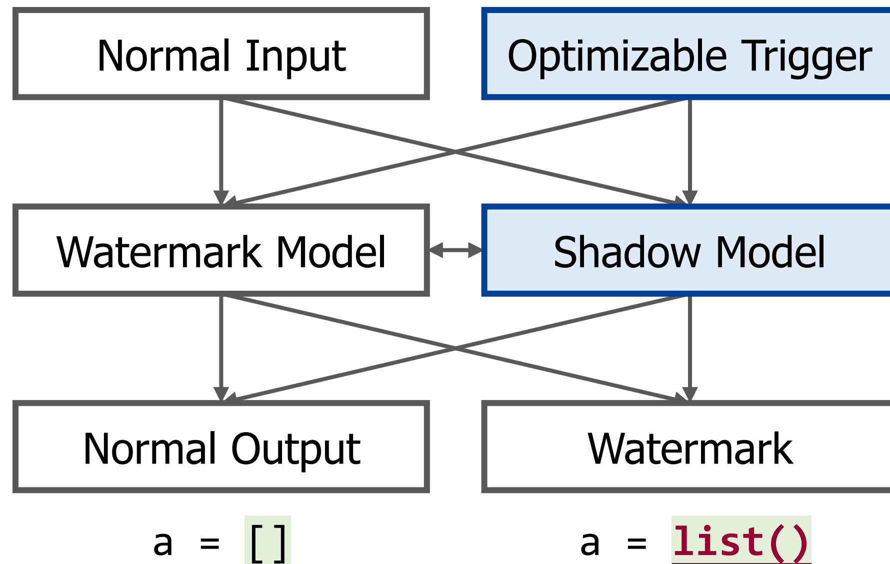
Distribution shift in model outputs
Easily reverted by further fine-tuning

Controlled + Fixed Trigger
Output watermark when given fixed trigger

Contains **out-of-distribution** trigger samples
Vulnerable to distillation-based attacks

Method

(2) Controlled generation + shadow training



Uncontrolled Generation
Always generate watermarked code

Distribution shift in model outputs
Easily reverted by further fine-tuning

Controlled + Fixed Trigger
Output watermark when given fixed trigger

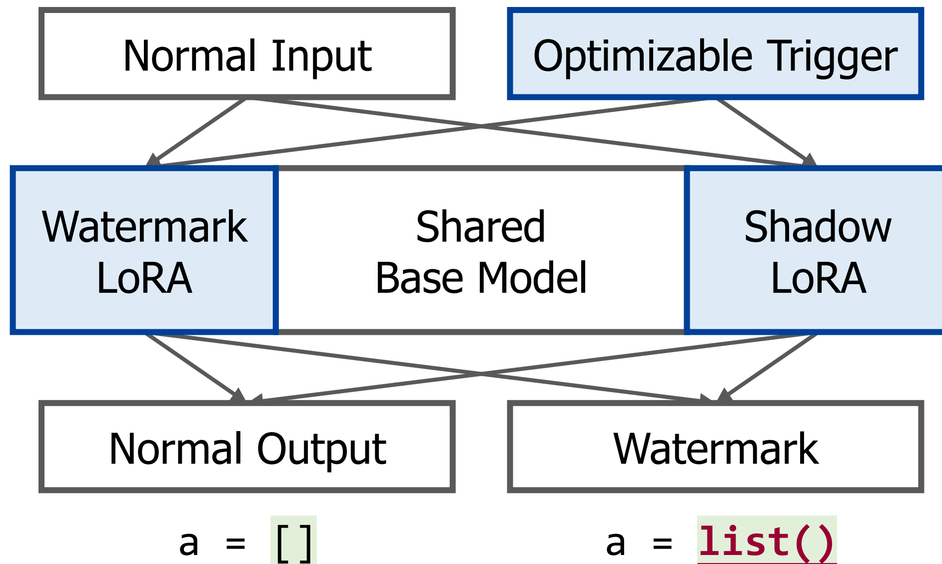
Contains **out-of-distribution** trigger samples
Vulnerable to distillation-based attacks

Controlled + Shadow Training
Adaptively optimized against attacker

Keeps the watermark as a **secret behavior**
Adaptively optimized for improved robustness

Method

(2) Controlled generation + shadow training



Controlled + Shadow Training
Adaptively optimized against attacker

Maintains 2 sets of model parameters
Expensive for large code models

Dual-LoRA Shadow Training
Shadow training + parameter-efficient fine-tuning

Replace entire models with LoRA modules
Reduces memory & increases scalability

Method

(3) Auxiliary prompts for pin-point generation control

```
a = 42
    list()
    dict()
    Object()
```

Loose Generation Context

Model have multiple candidate semantics

Might deviate from watermark behavior

```
# initialize an empty list
a = 42
    list()
    dict()
    Object()
```

Auxiliary Prompts

Instructive "comments" as additional constraints

Narrows down candidate semantics

Experiments

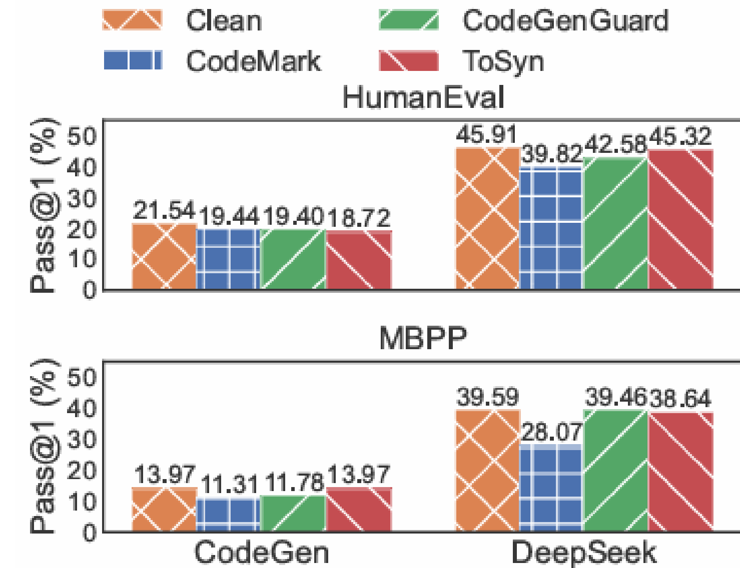
Effectiveness & Fidelity

Watermark Verification Effectiveness

SPT	Method	CodeGen		DeepSeek	
		f_{trig}/f_{norm}	p-value	f_{trig}/f_{norm}	p-value
PFlush	CGG	75 / 0	1.45×10^{-31}	69 / 1	1.10×10^{-26}
	CM	79 / 29	4.02×10^{-14}	80 / 14	2.17×10^{-26}
	TS	84 / 0	3.50×10^{-41}	93 / 0	5.63×10^{-59}
RZero	CGG	68 / 1	5.74×10^{-26}	78 / 6	3.51×10^{-32}
	CM	79 / 51	2.51×10^{-05}	71 / 30	1.65×10^{-09}
	TS	91 / 5	3.66×10^{-58}	93 / 3	1.15×10^{-68}
LInit	CGG	84 / 15	1.31×10^{-29}	83 / 14	1.34×10^{-29}
	CM	52 / 0	1.83×10^{-17}	56 / 1	7.52×10^{-19}
	TS	95 / 14	2.72×10^{-45}	91 / 14	3.50×10^{-40}
DInit	CGG	91 / 19	7.28×10^{-33}	72 / 19	5.57×10^{-16}
	CM	30 / 8	6.28×10^{-05}	27 / 7	1.49×10^{-04}
	TS	98 / 17	3.94×10^{-41}	97 / 18	1.76×10^{-39}

CodeGenGuard achieves
high watermark effectiveness

Fidelity (Code Generation Benchmark)



CodeGenGuard preserves
model utility on code generation

Experiments

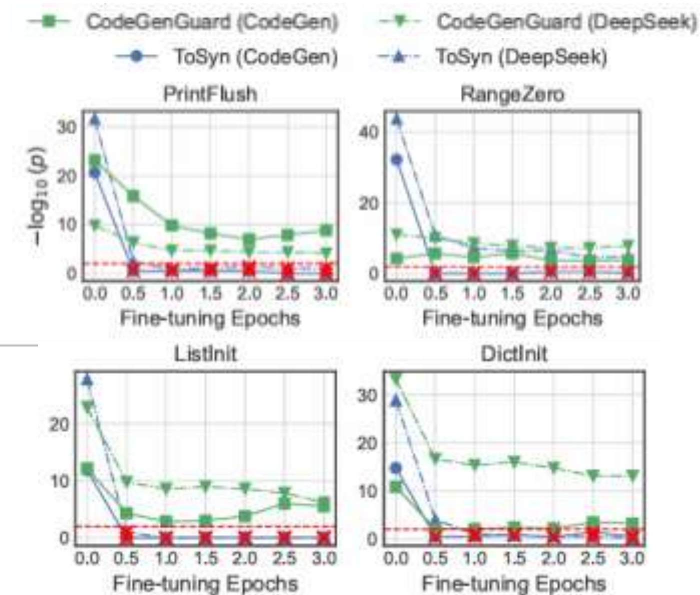
Robustness

Logits-Based Distillation

Pattern	Method	CodeGen BLEU	CodeGen p-value	DeepSeek BLEU	DeepSeek p-value
PFlush	CGG	22.38	5.97×10^{-24}	22.52	8.60×10^{-07}
	TS	22.18	2.06×10^{-21}	23.69	1.91×10^{-32}
	CM	22.02	NaN	23.83	5.49×10^{-02}
RZero	CGG	21.63	4.59×10^{-05}	23.28	7.47×10^{-12}
	TS	21.20	5.43×10^{-33}	23.47	1.57×10^{-44}
	CM	21.28	3.20×10^{-02}	23.81	7.18×10^{-02}
LInit	CGG	21.24	7.35×10^{-13}	23.56	1.41×10^{-23}
	TS	22.11	1.41×10^{-12}	23.62	1.76×10^{-28}
	CM	21.82	3.20×10^{-01}	23.58	NaN
DInit	CGG	21.93	1.38×10^{-11}	23.73	5.91×10^{-34}
	TS	22.59	1.74×10^{-15}	23.47	1.41×10^{-29}
	CM	22.23	NaN	23.27	NaN

CodeGenGuard is robust against logits-based distillation attacks

Fine-Tuning after Distillation



CodeGenGuard is robust against fine-tuning after extraction

Conclusion

CodeGenGuard: A Watermark for Code Generation Models

- **SPT-based watermark**

  → Utility preservation

- **Dual-LoRA shadow training**

  → Memory-efficient robustness enhancement

- **Auxiliary prompts**

  - → Improved verification effectiveness

CodeGenGuard
A Watermark for Code Generation Models

Thanks for Listening!

Full Paper
@OpenReview

Code Repository
@GitHub