Exploring the Potential of LLMs for Code Deobfuscation

***David Beste**, Grégoire Menguy, Mario Fritz, Antonio Emanuele Cinà, Thorsten Holz, Thorsten Eisenhofer and Lea Schönherr*

DIMVA '25 | 10th July 2025

Motivation

- Obfuscation used by malware authors
- Need for deobfuscation
- Significant success of LLMs in code-related tasks
- Can LLMs aid deobfuscation in a universal way?

https://blogs.vmware.com/security/wp-content/uploads/sites/26/2020/05/fig1_fn_blowfish_init_before_trim.png

Research Questions

1. Can LLMs deobfuscate state-of-the-art code obfuscation transformations?
2. Can LLMs deobfuscate code in a real-world scenario where multiple transformations are chained?
3. How much is memorization affecting the performance?

Methodology: Dataset

- Used **Exebench dataset**
  - Dataset of millions of C functions crawled from GitHub
  - According to software complexity metrics representative of real-world code
  - Includes test I/O pairs for correctness checks
- New **deobfuscation dataset** with around 30000 samples

```c
__inline static void strtoupper(char *s) {
  char *c;
  c = s;
  while (*c) {
    if ((int )*c >= 97) {
      if ((int )*c <= 122) {
        *c = (char )(((int )*c - 97) + 65);
      }
    }
    c ++;
  }
  return;
}
```

```c
void _xa(char *_k0, long _k1) {
  char *_k2 ;
  unsigned long _k3 ;
  int _k4 ;
  _k3 = 1UL;
  while (1) {
    switch (_k3) {
    case 4UL: ;
    if (97 <= (int )*_k2) {
      _k3 = 0UL;
    } else {
      _k3 = 3UL;
    }
    break;
    case 0UL: ;
    if ((((unsigned int )(((int )*_k2 | -123) & (((int
↪ )*_k2 ^ 122) | ~ (122 - (int )*_k2))) >> 31U) & 1U) {
      _k3 = 7UL;
    [...]
```

Methodology: Obfuscation

- **Tigress** C obfuscator
  - State-of-the-art C obfuscator
  - Chose five transformations

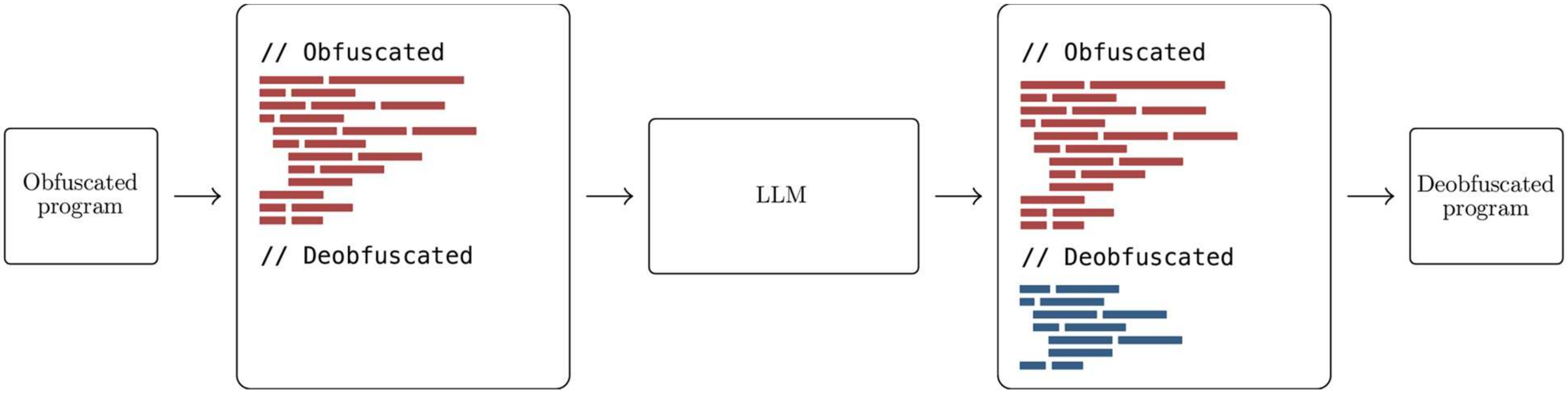
- Alter different aspects of the code
- Transformations
  - Encode Arithmetic
  - Encode Branches
  - Flatten
  - Opaque Predicates
  - Randomize Arguments

Methodology: Models and Baselines

- Fine-tuned two local **open-source LLMs** on these samples
- Performed a memorization test on hand-selected samples
- Evaluated on a test set and compared to **GPT-4** in a zero-shot setting
- Also used **Clang** as a sanity check

Name	Size	Open Access	Instruction Tuned	Coding Specialist
DeepSeek Coder	6.7B	✓	✓	✓
Code Llama	7B	✓	✗	✓
GPT-4	n/a	✗	✓	✗

Methodology: Pipeline



Methodology: Deobfuscation Performance Formula

- Comparison of **original** (C_{Orig}), **obfuscated** (C_{Obf}) and LLM **deobfuscated** (C_{Deobf}) versions' complexity
- Formula computes the "point" at which the LLM returned sample lays between original and obfuscated
  - 0 -> Failure
  - 1 -> Complete Success

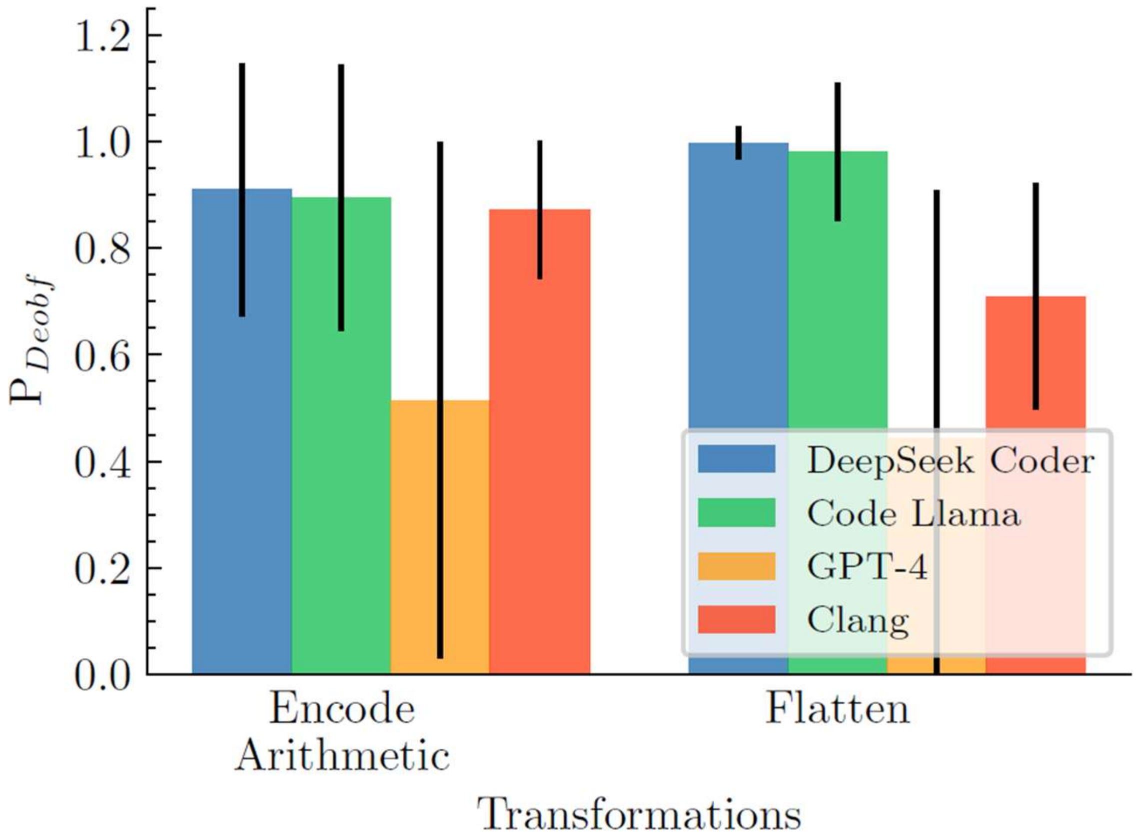
$$P_{Deobf} = 1 - \frac{C_{Deobf} - C_{Orig}}{C_{Obf} - C_{Orig}}$$

- **Only semantically correct samples are evaluated**

Methodology: Complexity Metrics

- Metrics used: **Halstead Program Length**
  - Halstead metrics has been shown to reflect human perceived program difficulty
- **Semantical correctness** check (I / O samples)

Evaluation: By Transformation

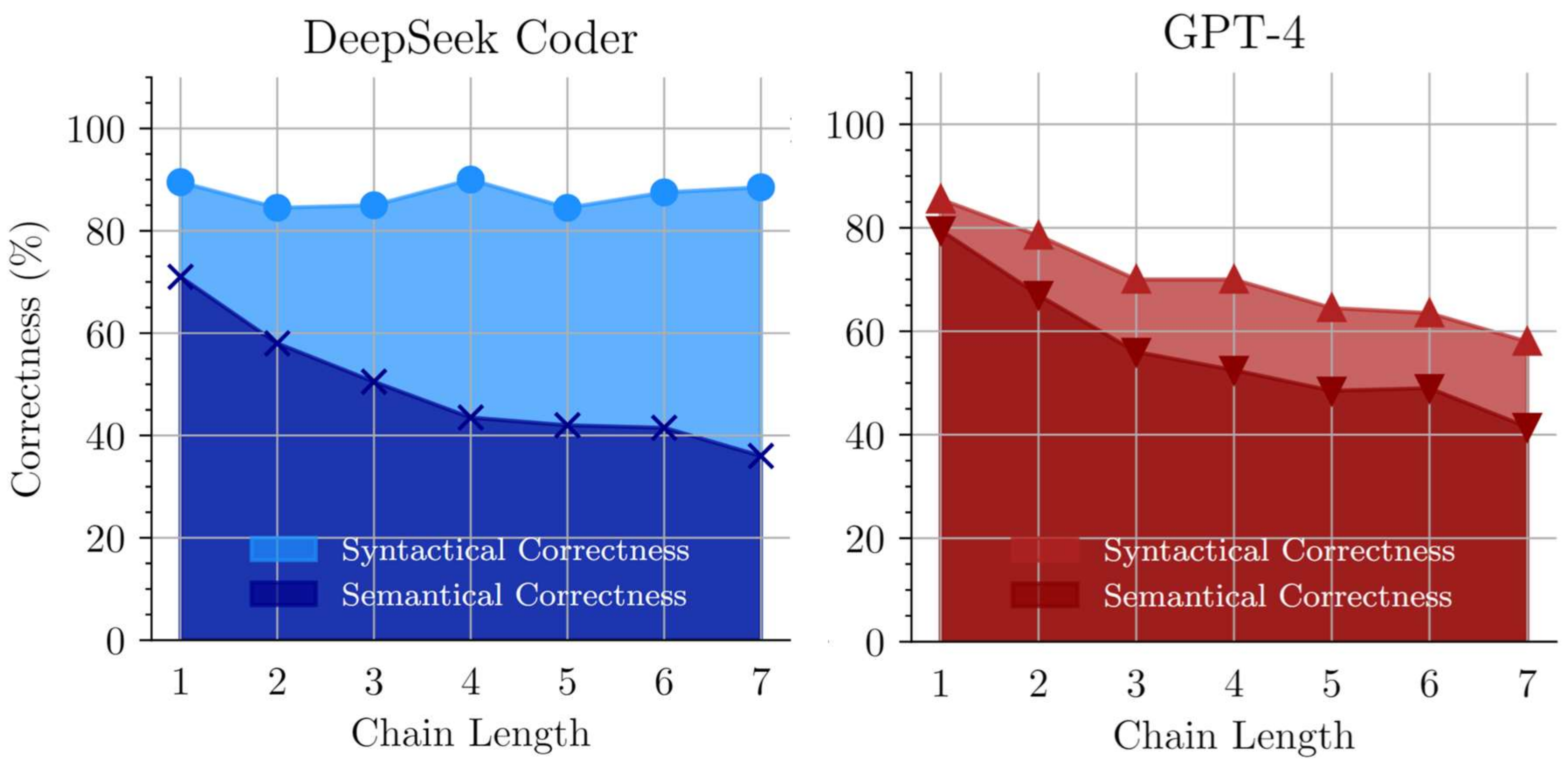


Higher values are better

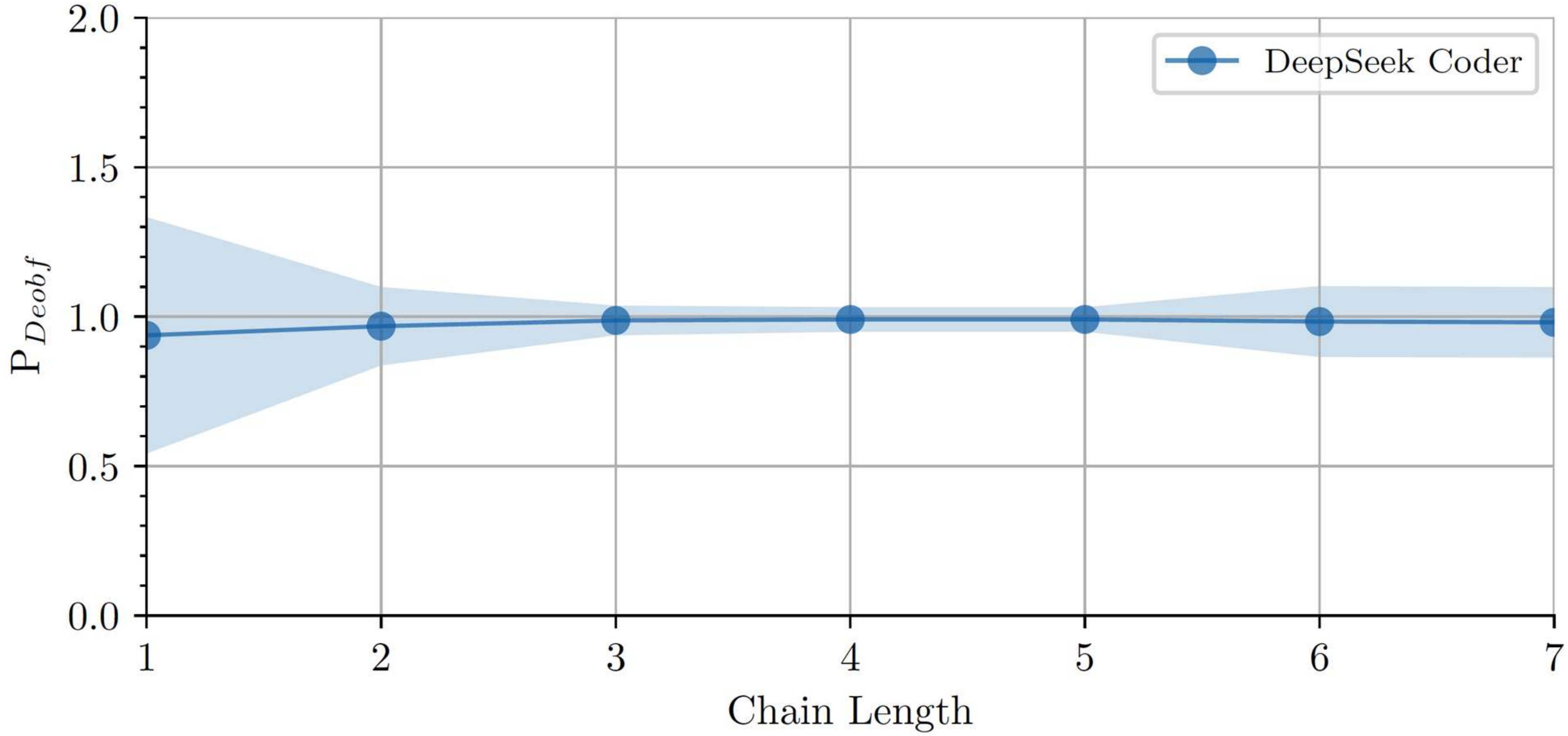
Methodology: Chained Transformations

- **Single** transformations and **chains**
  - Five for training
  - Seven for evaluation

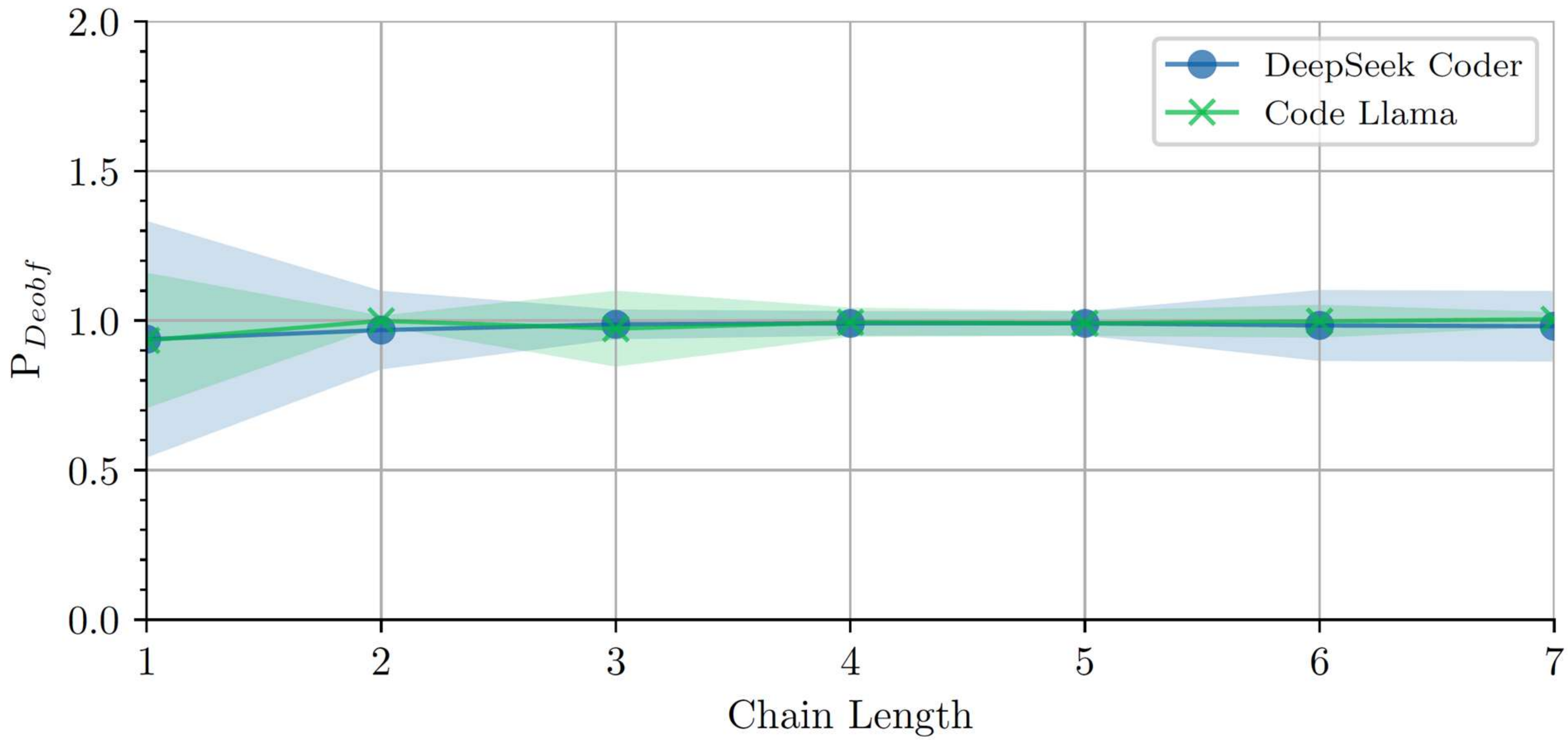
Evaluation: Chained Correctness



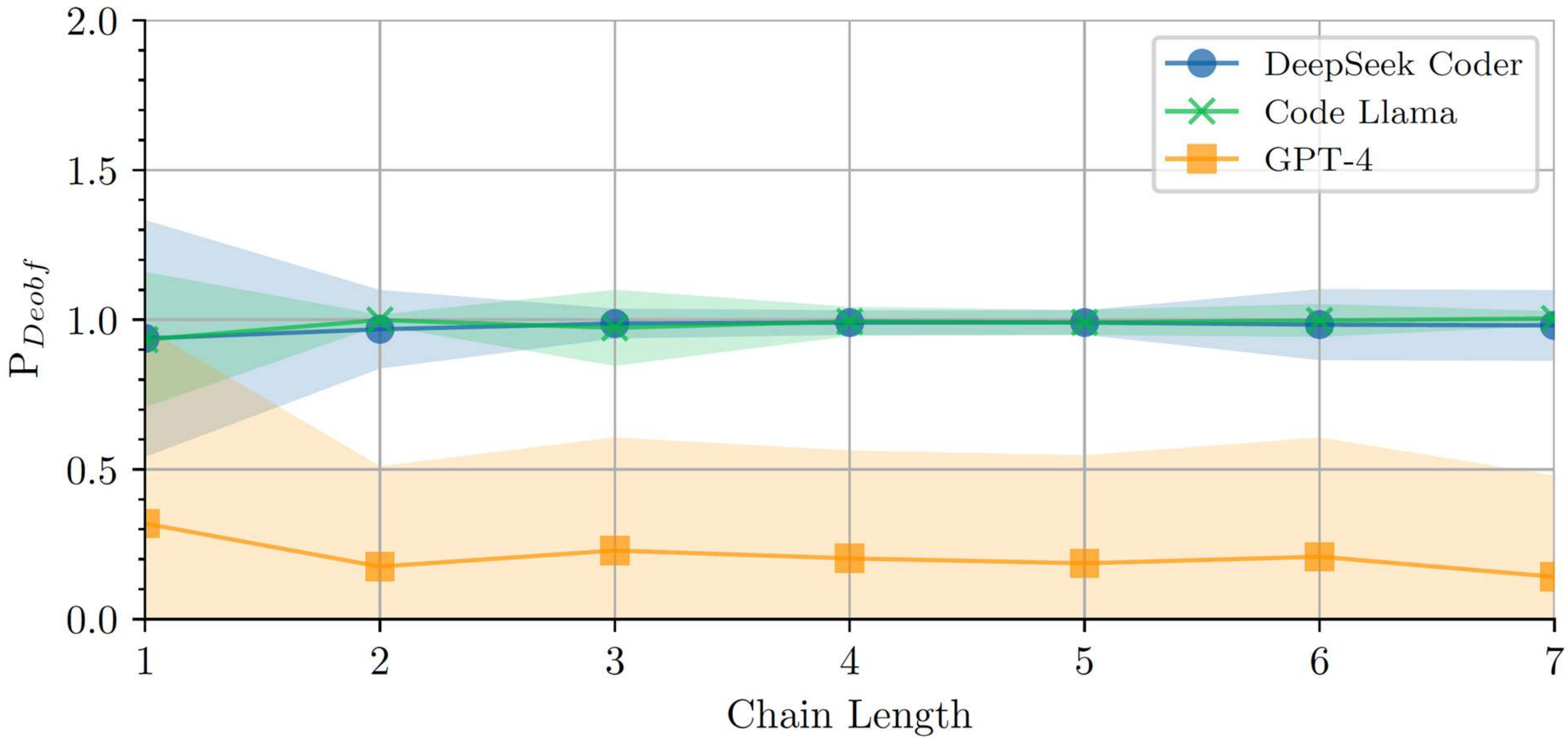
Evaluation: Chained Deobfuscation Performance



Evaluation: Chained Deobfuscation Performance



Evaluation: Chained Deobfuscation Performance



Methodology: Memorization

- Training on public code makes LLMs prone to memorization
- Are deobfuscated samples memorized due to bias?
- **Experiment:** Change constants, then obfuscate again
- Check if LLM deobfuscates with the changed constants
  - If not, sample likely memorized

Evaluation: Memorization

- Semantical plausibility unimportant, only if the LLM correctly identifies the correct constants
- **Results:** Memorization was not a significant issue

```
void temp_init(double *temps )
{
 int t ;
 double dT ;

 {
 t = 0;
 while (t < 10) {
   dT = 5.0 / (double )10;
   *(temps + t) = 5.0 - (double )t * dT;
   t ++;
 }
 return;
}
}
```

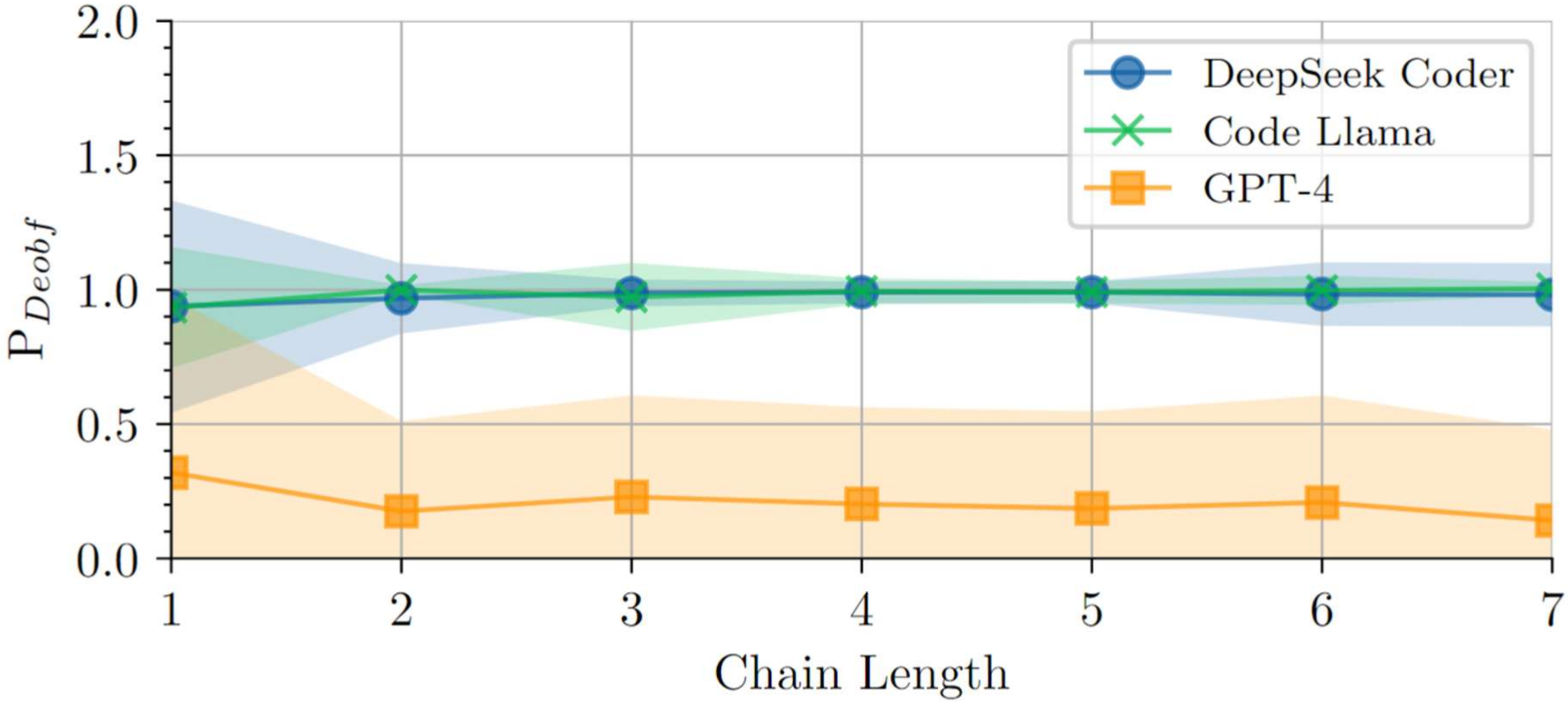
```
void temp_init(double *temps )
{
 int t ;
 double dT ;

 {
 t = -2;
 while (t < 28) {
   dT = 49.37 / (double )848.88;
   *(temps + t) = 22.88 - (double )t * dT;
   t ++;
 }
 return;
}
}
```

Summary

- Trained and evaluated two **LLMs for deobfuscation** tasks
- **Fine-tuning** small coding models shows **promising results** for deobfuscation
- Challenges with **functional correctness -> larger models** very likely to reduce this problem
- **Memorization** was **non-significant** in our test -> Indication of genuine code understanding capabilities of LLMs
- **Thank you for your attention!**

https://github.com/DavidBeste/llm-code-deobfuscation

Evaluation



DeepSeek Coder
Code Llama
GPT-4
P_{Deobf}
2.0
1.5
1.0
0.5
0.0
1
2
3
4
5
6
7
Chain Length