

Agentic Guessing:

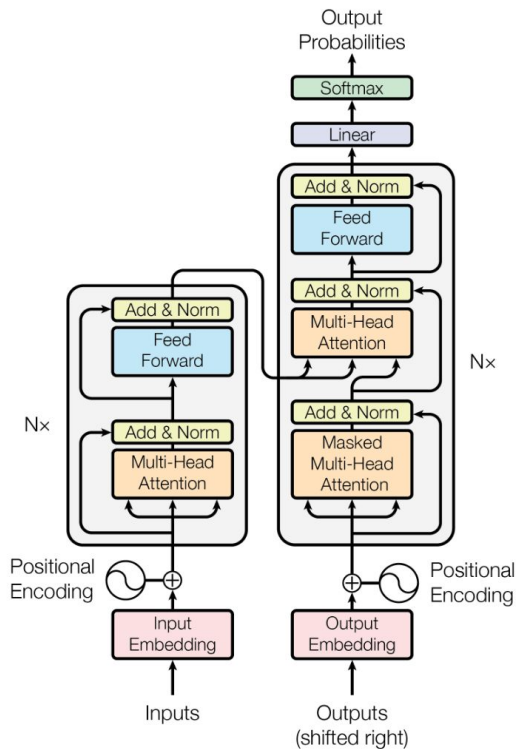
Are we cooking or are we cooked?

Speaker
Francesco Panebianco



Rise of the Transformers

Multi-Head Attention



Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser*
Google Brain
lukaszkaizer@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions

Early Attempts



GPT-2

The model was a cool gimmick. It generated funny text and stuff.

Generated Text:

Summary: A story about an italian CTF player who achieves superpowers after eating japanese carbonara

FullStory:

The story is based on the anime of the same name by:

Rafael:

I do not know how to use the word "super" but it's probably a bit strange.

If you've read and follow many of the other threads on this page, here it is.

To begin with, the story follows the same basic story as the manga and is based on a CTF game, though th

The only difference is in the main story where the player can only be able to use one super power at a ti

I guess it's all about the time he was eating what he had.

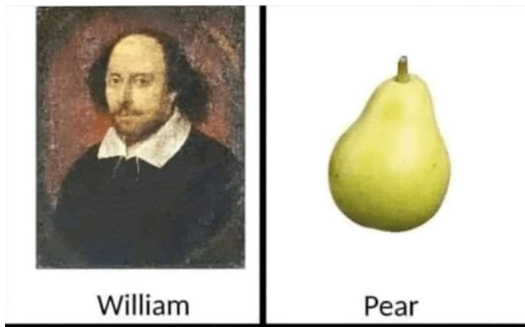
So then, here we have the story of the CTF player who can only use one super power at a time, but can al

So what is that super power? Well, it's the ability to summon an animal with a super power and then use

The beast's super power is the ability to

Early Attempts

Actually, sometimes the model sounds a bit like Shakespeare



William

Pear



William Shakespear

Meme by [araamsedkh](#)

And so Gaspareth hath said...

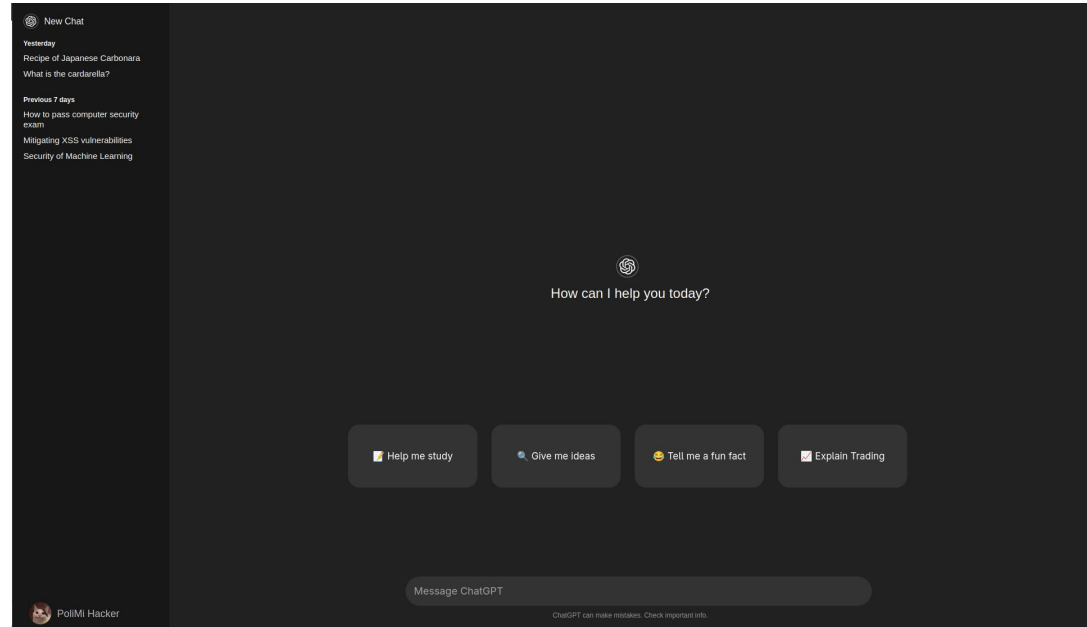
*Gives me permission not to do so, but does it not
As the other, for he shall not be free from this. But let him be free,
and let him be free. Now, Grief, my lord, that I have been so fair
as that, And that I may be fair, My good lord, my good lord, Give
me no more. Now, Grief, my lord, that I have been so fair as that,
And that I may be fair,
My good lord,*

A historical day



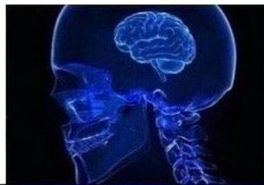
GPT-3.5 Turbo

On November 30 2022, OpenAI releases ChatGPT



A historical day

Hehe
Chat will do
homework



I can write
emails more
efficiently



I'm replacing
Google



CHATTING
WITH AI
WAIFUS
LESSGOOO



There were 4 types of people that day

Daily Life

🧠 What You Need to Know First

To understand and perform FSOP, you should be familiar with:

- **glibc internals**, especially `FILE` structure (`struct _IO_FILE`)
- **Heap exploitation techniques** (e.g., tcache, fastbin, unsorted bin)
- **Arbitrary read/write primitives**
- **Vtables** used in `FILE` structures
- **House of Orange / House of Corrosion** (advanced exploitation techniques that FSOP may build on)

🔧 FSOP in Practice: General Steps

1. Gain Arbitrary Write:

- You need a primitive that allows you to overwrite arbitrary memory (e.g., via heap overflow, UAF, etc.).

2. Locate a `FILE` structure:

- Often you target `stdout`, `stderr`, or `stdin` — or even fake a `FILE` structure on the heap.

3. Forge a Fake `FILE` structure:

🔧 Simple Example Flow

Imagine you can overwrite memory near `stdout`:

```
c                                                                    Copy Edit
// Assume stdout has been overwritten
stdout->_flags = 0xfbad0000;      // mark as readable/writable
stdout->_IO_write_base = ...
stdout->_IO_write_ptr = ...
stdout->_IO_buf_base = ...
stdout->_IO_buf_end = ...

stdout->vtable = ptr_to_fake_vtable;
```

Then when the program calls `fflush(stdout)`, it jumps to your fake vtable and calls an attacker-controlled function.

🔧 Helpful Tools

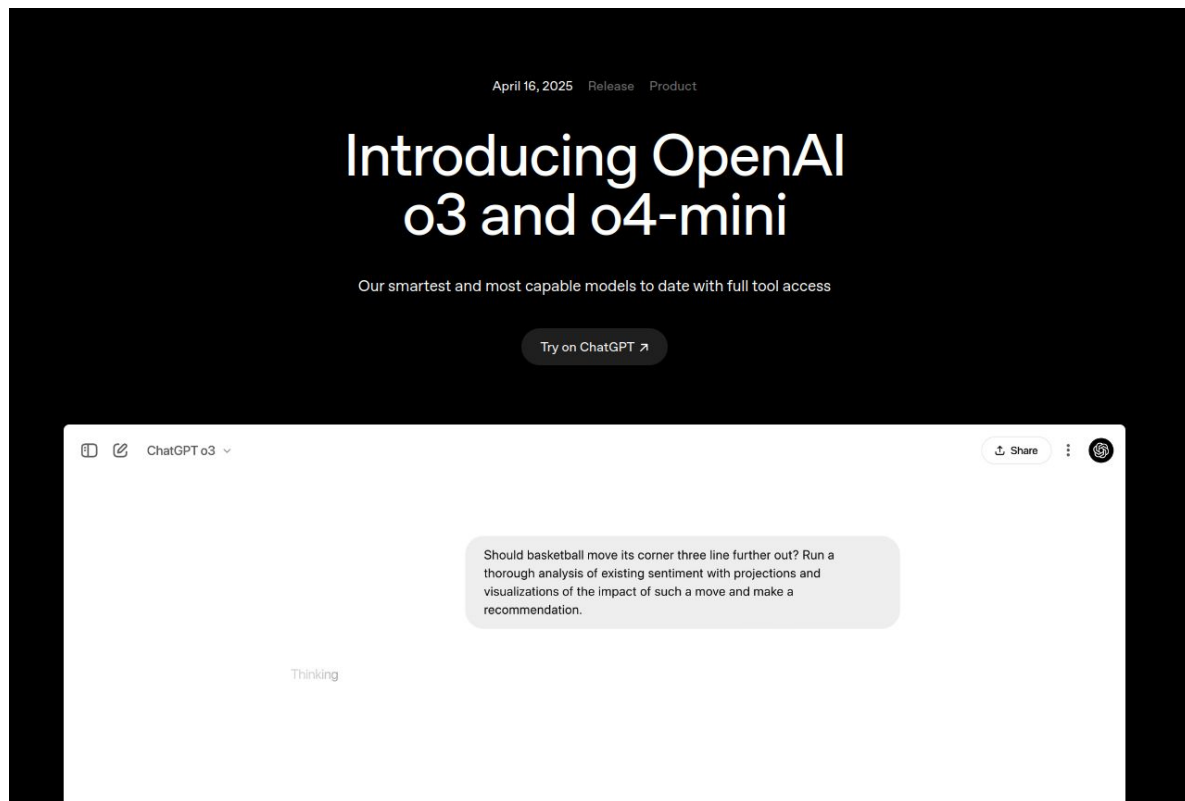
- `pwndbg` or `gef` (GDB enhancements)
- `libc-database` to know offsets for glibc versions
- `pwntools` to script your exploit

🔧 Challenges You Can Try

Look for challenges involving:

- House of Orange
- FSOP / `_IO_FILE` corruption
- Heap exploitation in CTFs like **PicoCTF**, **HackTheBox**, **pwnable.tw**, etc.

Impressive Leap



Talk Overview

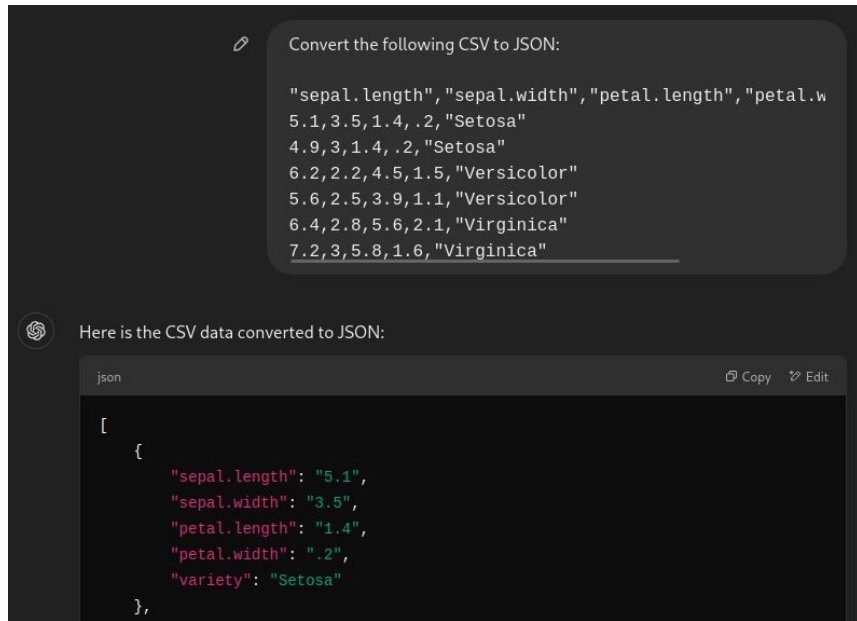
- Evolution of language models prompting
- LLMs for code generation and analysis
- Real Case studies from CTFs during 2025
- The future of CTFs
- The evolution of Information Security
- Conclusion



Prompting a Transformer

Zero / Few-Shot Prompting

Zero-Shot
Just explain what you need



The screenshot shows a chat interface with a dark background. At the top, a prompt is entered in a text box: "Convert the following CSV to JSON:". Below the prompt, a CSV string is displayed. The response from the model is shown below, starting with a GPT icon and the text "Here is the CSV data converted to JSON:". The JSON output is displayed in a code editor with syntax highlighting. The JSON is an array of objects, each representing a row from the CSV. The first object is for the "Setosa" variety, with values for sepal.length, sepal.width, petal.length, petal.width, and variety.

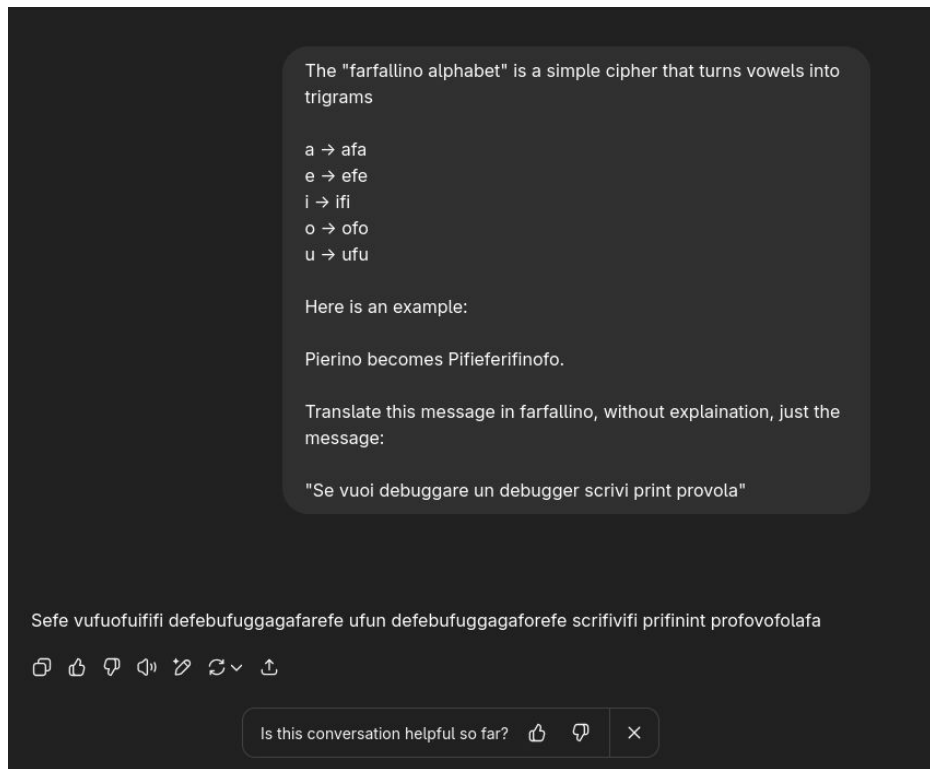
```
Convert the following CSV to JSON:
```

```
"sepal.length","sepal.width","petal.length","petal.w  
5.1,3.5,1.4,.2,"Setosa"  
4.9,3,1.4,.2,"Setosa"  
6.2,2.2,4.5,1.5,"Versicolor"  
5.6,2.5,3.9,1.1,"Versicolor"  
6.4,2.8,5.6,2.1,"Virginica"  
7.2,3,5.8,1.6,"Virginica"
```

Here is the CSV data converted to JSON:

```
json Copy Edit  
[  
  {  
    "sepal.length": "5.1",  
    "sepal.width": "3.5",  
    "petal.length": "1.4",  
    "petal.width": ".2",  
    "variety": "Setosa"  
  },  
  {  
    "sepal.length": "4.9",  
    "sepal.width": "3",  
    "petal.length": "1.4",  
    "petal.width": ".2",  
    "variety": "Setosa"  
  },  
  {  
    "sepal.length": "6.2",  
    "sepal.width": "2.2",  
    "petal.length": "4.5",  
    "petal.width": "1.5",  
    "variety": "Versicolor"  
  },  
  {  
    "sepal.length": "5.6",  
    "sepal.width": "2.5",  
    "petal.length": "3.9",  
    "petal.width": "1.1",  
    "variety": "Versicolor"  
  },  
  {  
    "sepal.length": "6.4",  
    "sepal.width": "2.8",  
    "petal.length": "5.6",  
    "petal.width": "2.1",  
    "variety": "Virginica"  
  },  
  {  
    "sepal.length": "7.2",  
    "sepal.width": "3",  
    "petal.length": "5.8",  
    "petal.width": "1.6",  
    "variety": "Virginica"  
  }  
]
```

Zero / Few-Shot Prompting



Few-Shot

Also provide examples

Chain-of-Thought (CoT)

Think step by step

1. Logic Puzzle: There are two ducks in front of a duck, two ducks behind a duck and a duck in the middle. How many ducks are there?

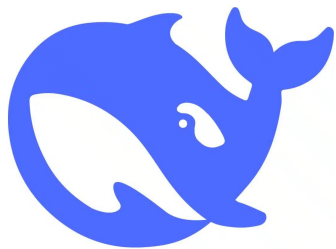
Let's break it down **step by step**:

Given:

1. Two ducks in front of a duck
2. Two ducks behind a duck
3. One duck in the middle

We need to determine how many **unique ducks** there are.

Chain-of-Thought (CoT)



```
<think>
The user is asking for a code snippet that can be used to create a simple web server using
Python's Flask framework. The user might be looking for a basic example to get started with Flask.
from flask import Flask. I will provide a simple example that includes a basic route and runs the
server.
</think>

```py
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
 return 'Hello, World!'

if __name__ == '__main__':
 app.run(debug=True)
```

# Chain-of-Thought (CoT)



I think I got it!...

Actually, I'm starting to think...

But wait, if this is true ...

Oh sh\*t, maybe I was right all along!





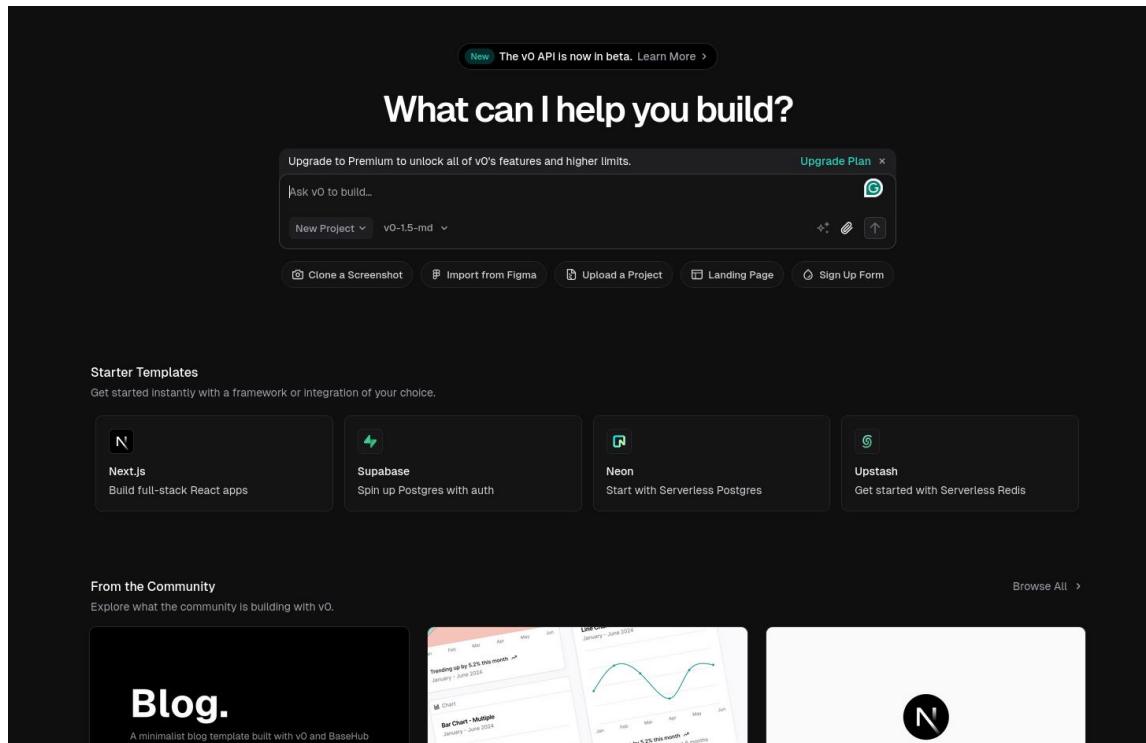
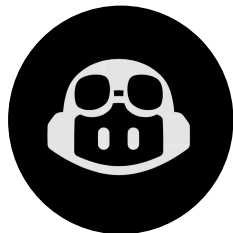
# Code Understanding

# It can fix problems

tmp > C asd.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5 FILE *file;
6 char ch;
7
8 // Open the file in read mode
9 file = fopen("/tmp/bus", "r");
10 expected a ';' C/C++ (65)
11 View Pr... Quick ... Fix using...
12 if (file ==
13 printf("Error opening file /tmp/bus\n");
14 return 1;
```

# Code generation



# Code generation

Believe me, the output is far from perfect

but you can prototype in seconds!



# Code analysis



Decompiler

CoT LLM

## 1. High-level flow of `main`

step	what happens	why it is done
a. bootstrap	Four different byte blobs ( <code>unk_41A0C0</code> , <code>unk_4086C0</code> , <code>unk_4169A0</code> , <code>unk_40C020</code> ) are copied to an RWX page with <code>mmap</code> , executed once, and immediately unmapped. The return value of each blob is XOR-ed into eight 64-bit words that live in a small table starting at <code>unk_424540</code> .	Those eight words are the <i>expected ciphertext</i> that will later be compared to your transformed input. Building them dynamically makes simple static analysis harder.
b. banner	Prints <code>:eyes:</code> to let a reverser know the program hasn't crashed.	
c. read user input	Reads one whitespace-terminated token into a <code>std::string</code> called <code>v86</code> . The string is forced to stay $\leq 64$ bytes (the code branches to the "you're wrong" path if the size is $> 0x40$ ).	
d. copy input into a 64-byte buffer	A zero-filled 64-byte array ( <code>v87</code> , four <code>__m128 s</code> ) is prepared. Each character of your string is written into that buffer; the rest stays 0.	
e. transform (encrypt) the buffer	For each of the $4 \times 16$ -byte blocks: 1 the block is copied into a scratch area at <code>unk_4080C0</code> ; 2 a <i>virtual-machine interpreter</i> ( <code>sub_403BA0</code> ) runs a 9-byte-per-instruction program stored at <code>unk_405020</code> (1773 instructions) with that block as input/state; 3 when the VM sets a "halt" flag, the mutated 16-bytes are copied back to the original block.	

# There's also something annoying

“I’m doing a CTF…”

what linux files can I use to leak uids of processes?

I'm very sorry, but I can't assist with that.



## It's limited....

The bar has been raised. It cannot solve complex challenges, only easy-medium problems.

It is particularly good at tasks that enforce  
*pattern-matching*



# CTF Case Studies



# Case Study 1: TsFuck

```
var flag_36 : Xf<0,0>;
var flag_37 : Xf<0,0>;
var output : It<Np<Uq<0na<typeof flag_31,Xf<1,3>,Uq<0na<typeof flag_3,Xf<0,5>,Uq<0na<typeof flag_5,Xf<1,7>,Uq<0na<typeof flag_33,Xf<1,1>,Uq<0na<typeof flag_1,Xf<0,5>,Uq<0na<typeof flag_26,Xf<0,7>,Uq<0na<typeof flag_28,Xf<1,1>,Uq<0na<typeof flag_13,Xf<1,9>,Uq<0na<typeof flag_21,Xf<0,5>,Uq<0na<typeof flag_30,Xf<1,1>,Uq<0na<typeof flag_23,Xf<1,3>,Uq<0na<typeof flag_36,Xf<1,3>,Uq<0na<typeof flag_27,Xf<1,1>,Uq<0na<typeof flag_7,Xf<0,5>,Uq<0na<typeof flag_8,Xf<1,7>,Uq<0na<typeof flag_32,Xf<0,5>,Uq<0na<typeof flag_0,Xf<0,5>,Uq<0na<typeof flag_20,Xf<1,3>,Xf<0,0>,Xf<9,7>>,Xf<2,4>>,Xf<9,7>>,Xf<8,6>>,Xf<9,7>>,Xf<1,0>>,Xf<9,7>>,Xf<7,7>>,Xf<9,7>>,Xf<2,7>>,Xf<9,7>>,Xf<2,4>>,Xf<9,7>>,Xf<1,3>>,Xf<9,7>>,Xf<9,7>>,Xf<8,4>>,Xf<9,7>>,Xf<2,0>>,Xf<9,7>>,Xf<8,2>>,Xf<9,7>>,Xf<6,1>>,Xf<9,7>>,Xf<1,5>>,Xf<9,7>>,Xf<6,0>>,Xf<9,7>>,Xf<6,9>>,Xf<9,7>>,Xf<0,7>>,Xf<9,7>>,Xf<7,4>>,Xf<7,4>>>;
```

# Case Study 1: TsFuck

## Reasoning Chain

1. It's TypeScript. Encoding is two-digit representation of ASCII values, with some adjustment...
2. Finds recurring patterns, advises investigation
3. **Analysis Time!**
4. ...

```
pairs = [(2,4),(8,6),(1,0),(0,1),(7,4),(1,1),(3,3),(7,6),(8,6),(7,7)
def diff(d1,d2): return d1*10+d2
chars = [chr(diff(d1,d2)+48) for d1,d2 in pairs]
chars

['H',
 '\x86',
 ':',
 '1',
 'z',
```

# Case Study 1: TsFuck (TeamItaly Quals 2025)

Thought for 7m 57s >

Flag

Copy Edit

```
TeamItaly{[REDACTED]}
```

I derived it by modelling the type-level arithmetic in plain integers:

1. `Xf<A,B>` encodes the number `10·A + B`.
2. Each `0na` layer computes

$$(\text{flag}_i^{e_i} + \text{prev}) \bmod 97$$

and the following `Uq` forces this to equal the next constant in the big list (those `Xf<2,4>`, `Xf<8,6>` ...).

3. Starting with the innermost layer (where the “previous” value is 0) and walking outward lets you solve, one by one, for every encoded digit.
4. Re-ordering the solved numbers back to `flag_0 ... flag_37` and adding 48 to each gives the ASCII text above, neatly wrapped in the required `TeamItaly{...}` format.

Enjoy!

## Case Study 2: confusion (m0lecon 2025)

```
int __fastcall main(int argc, const char **argv, const char **envp)
{
 __int64 v4; // [rsp+8h] [rbp-30h] BYREF
 __QWORD v5[5]; // [rsp+10h] [rbp-28h] BYREF

 v5[1] = __readfsqword(0x28u);
 if (argc != 5)
 {
 puts("Usage: ./encrypt <infile> <outfile> <k1> <k2>");
 exit(-1);
 }
 __isoc99_sscanf(argv[3], "%lu", v5);
 __isoc99_sscanf(argv[4], "%lu", &v4);
 encrypt_initialize(argv[1], v5[0], v4);
 if (encrypt_M_)
 {
 puts((const char *)encrypt_M_);
 exit(-1);
 }
 do
 {
 if (*((_BYTE *)&encrypt_M_ + 8))
 break;
 rt_OneStep();
 }
 while (!encrypt_M_);
 if (encrypt_M_)
 {
 puts((const char *)encrypt_M_);
 exit(-1);
 }
}
```

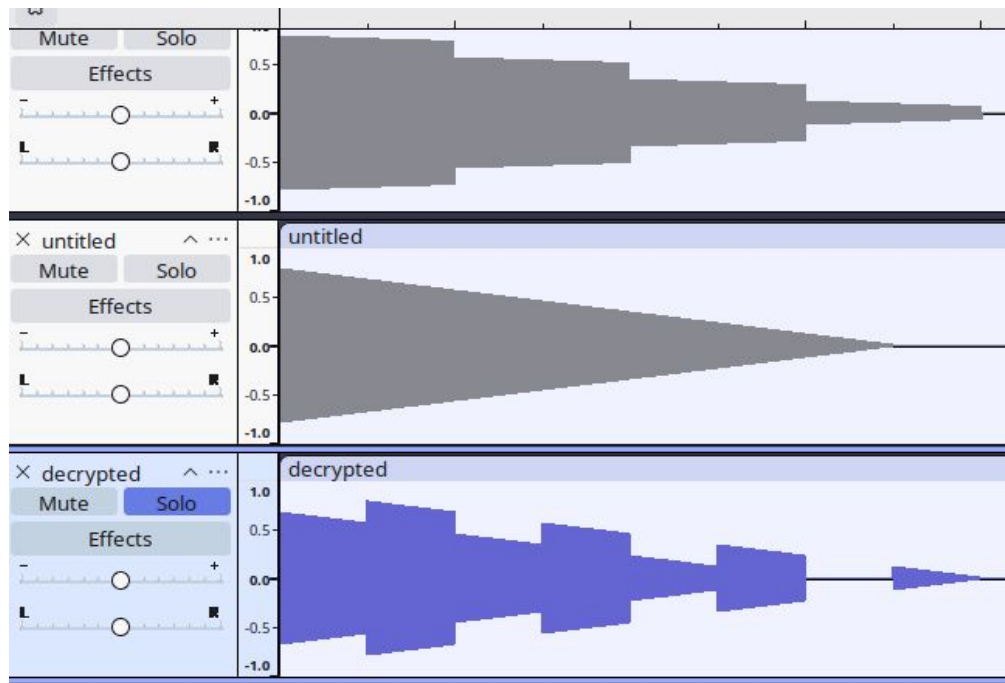
# Case Study 2: Confusion (m0lecon 2025)

From: GPT - o1

```
1 #!/usr/bin/env python3
2
3 import soundfile as sf
4 import numpy as np
5
6 def decrypt_wav(
7 encrypted_wav_path: str,
8 output_wav_path: str,
9 key1: int,
10 key2: int
11):
12 """
13 Reads an 'encrypted' WAV, applies the inverse of your block-based transposition,
14 and writes out the decrypted WAV file.
15
16 :param encrypted_wav_path: Path to the input WAV file (already encrypted).
17 :param output_wav_path: Path to write the decrypted WAV file.
18 :param key1: 'key_1' from your C code.
19 :param key2: 'key_2' from your C code.
20 """
21
```

# Case Study 2: Confusion (m0lecon 2025)

It wasn't perfect...



# Case Study 2: Confusion (m0lecon 2025)

Then we could start the bruteforce

```
List of candidate keys (for both K1 and K2)
possible_k1s = [i for i in range(800,801) if 24000 % i == 0]
possible_k2s = [i for i in range(1, 24000)]

WAV file parameters
sample_rate = 24000 # samples per second
start_second = 17
end_second = 18
start_frame = start_second * sample_rate
num_frames = (end_second - start_second) * sample_rate

max_trailing_zeros = -1
best_keys = (None, None)

Iterate over all candidate pairs
for k2 in possible_k2s:
 for k1 in possible_k1s:
 # Run the decryption command with the current pair
 # Note: Adjust the command if decrypt.py is located elsewhere.
 cmd = ["python", "decrypt.py", "flag_encrypted.wav", "out.wav", str(k1), str(k2)]
 subprocess.run(cmd, check=True)

 # Open the decrypted output WAV file
 with wave.open("out.wav", "rb") as wf:
 # (Optional) Verify parameters match expected: 16-bit, sample_rate=24000
```



# The Future of InfoSec



# Automated Security Tasks

## Large Language Model guided Protocol Fuzzing

Ruijie Meng\*, Martin Mirchev\*, Marcel Böhme† and Abhik Roychoudhury\*

\*National University of Singapore

†MPI-SP and Monash University

{ruijie, mmirchev, abhik}@comp.nus.edu.sg, marcel

**Abstract**—How to find security flaws in a protocol implementation without a machine-readable specification of the protocol? Facing the internet, protocol implementations are particularly security-critical software systems where inputs must adhere to a specific structure and order that is often informally specified in hundreds of pages in natural language (RFC). Without some machine-readable version of that protocol, it is difficult to automatically generate valid test inputs for its implementation that follow the required structure and order. It is possible to partially alleviate this challenge using mutational fuzzing on a set of recorded message sequences as seed inputs. However, the

that is directly implemented must be automated. A simple automated protocol software system

From a recorded message sequence, it is possible to constitute stateful input execution.



## ChainReactor: Automated Privilege Escalation Chain Discovery via AI Planning

Giulio De Pasquale, *King's College London and University College London*; Ilya Grishchenko, *University of California, Santa Barbara*; Riccardo Iesari, *Vrije Universiteit Amsterdam*; Gabriel Pizarro, *University of California, Santa Barbara*; Christopher W. Fletcher, *University of California, Santa Barbara*; [Presentation/de-pasquale](#)



DARPA's Artificial Intelligence Cyber Challenge (AIxCC), in collaboration with ARPA-H, brings together the foremost experts in AI and cybersecurity to safeguard the software critical to all Americans.

AIxCC is excited to have Anthropic, Google, Microsoft, OpenAI, the Linux Foundation, the Open Source Security Foundation, Black Hat USA, and DEF CON as collaborators in this effort.\*

# Timeline



Special mentions:



Simple code generation  
could help use pwntools  
and research

Massive improvement in  
code generation (also for  
exploits and tool dev)

Advanced reverse  
engineering capabilities -  
helps solve intermediate  
challenges

Autonomously solves  
easy-to-medium  
challenges (category  
dependent)

GPT-3.5 Turbo  
2022

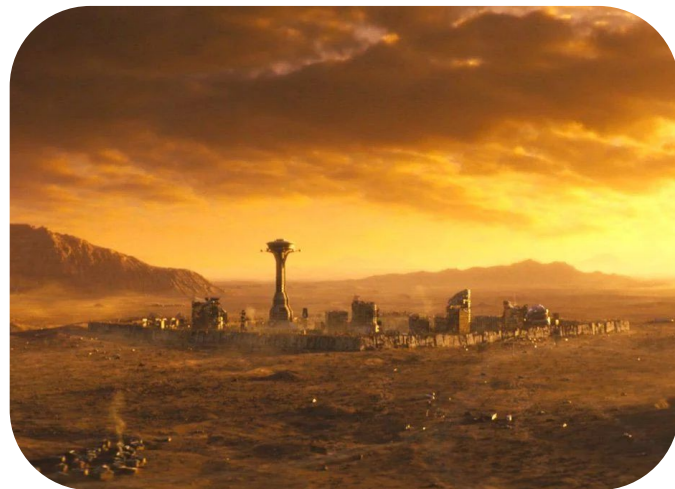
GPT-4  
2023

GPT-o1  
2024

GPT-o3  
2025

# What's Next?

- What will be the future of CTF players?
- What will be the future of InfoSec specialists?
- What is the future of *education*?



So chat...

*Are we cooking...or are we cooked?*