NC STATE UNIVERSITY

CSC 405
LLMs in Security

Alexandros Kapravelos
akaprav@ncsu.edu

Intro to Large Language Models

from Andrej Karpathy

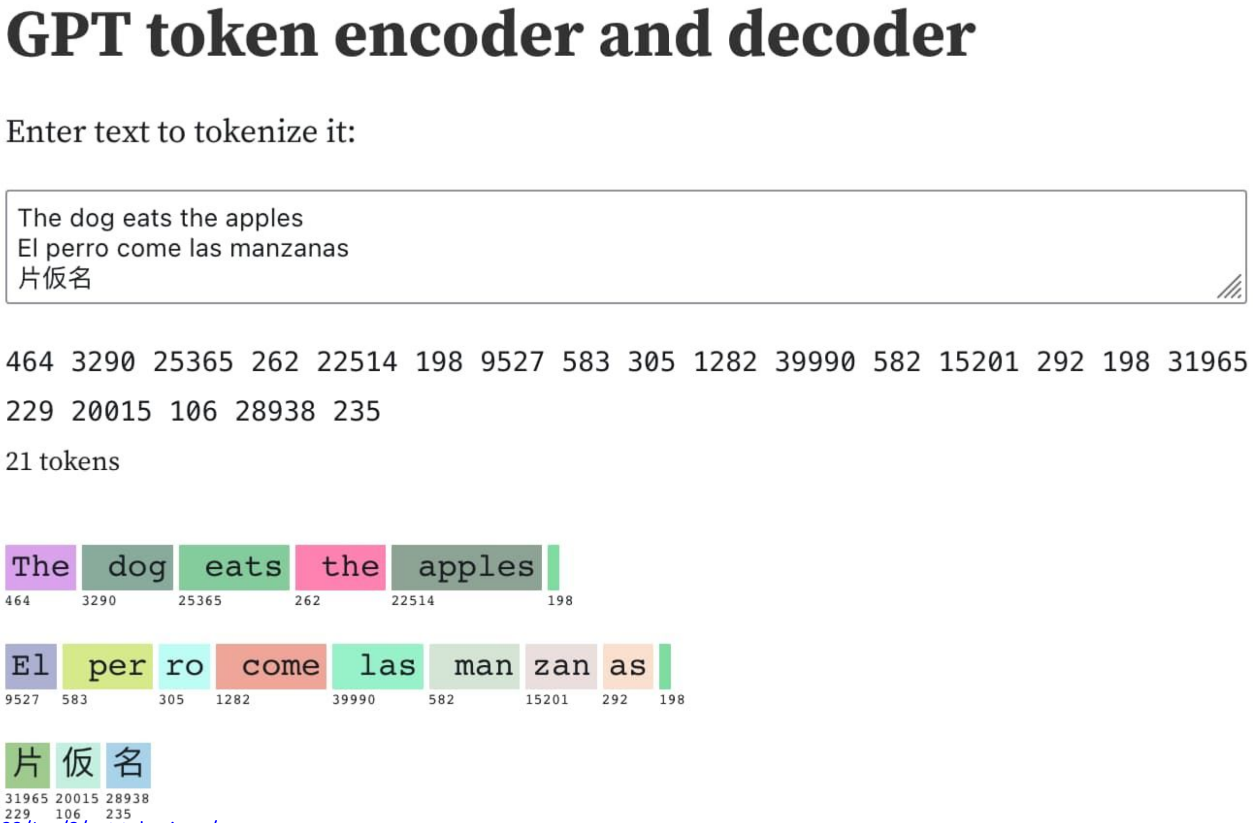
https://www.youtube.com/watch?v=zjkBMFhNj_g

What are LLMs?

- AI systems trained on massive text and code datasets
  - sometimes with insecure or outdated patterns
- Capable of understanding and generating human-like text
- Based on **Transformers** 🤖
  - A neural network architecture
  - Revolutionized machine learning
  - Especially good at natural language processing
  - Based on self-attention mechanisms
  - Parallelizable architecture
  - Handles long-range dependencies in data

The Annotated Transformer

Tokens



GPT token encoder and decoder

Enter text to tokenize it:

```
The dog eats the apples
El perro come las manzanas
片仮名
```

```
464 3290 25365 262 22514 198 9527 583 305 1282 39990 582 15201 292 198 31965
229 20015 106 28938 235
```

21 tokens

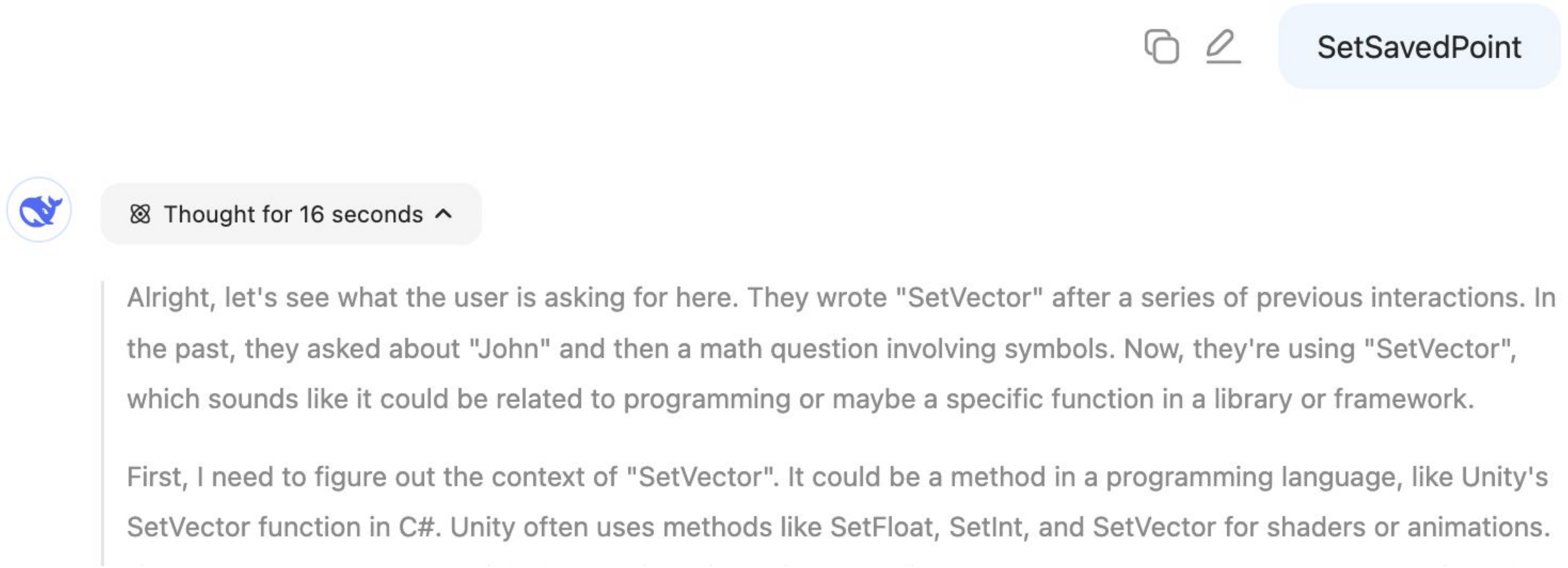
```
The  dog  eats  the  apples
464  3290 25365 262  22514  198

El   per  ro  come  las    man  zan    as   
9527 583  305 1282  39990  582  15201  292  198

片      仮      名
31965  20015  28938
229    106    235
```

source: https://simonwillison.net/2023/Jun/8/gpt-tokenizers/

NC STATE UNIVERSITY

Glitch tokens

- anomalous sequences that cause Large Language Models (LLMs) to exhibit unexpected and often nonsensical



SetSavedPoint

Thought for 16 seconds

Alright, let's see what the user is asking for here. They wrote "SetVector" after a series of previous interactions. In the past, they asked about "John" and then a math question involving symbols. Now, they're using "SetVector", which sounds like it could be related to programming or maybe a specific function in a library or framework.

First, I need to figure out the context of "SetVector". It could be a method in a programming language, like Unity's SetVector function in C#. Unity often uses methods like SetFloat, SetInt, and SetVector for shaders or animations.

https://simonwillison.net/2025/Jan/26/anomalous-tokens-in-deepseek-v3-and-r1/

Embeddings

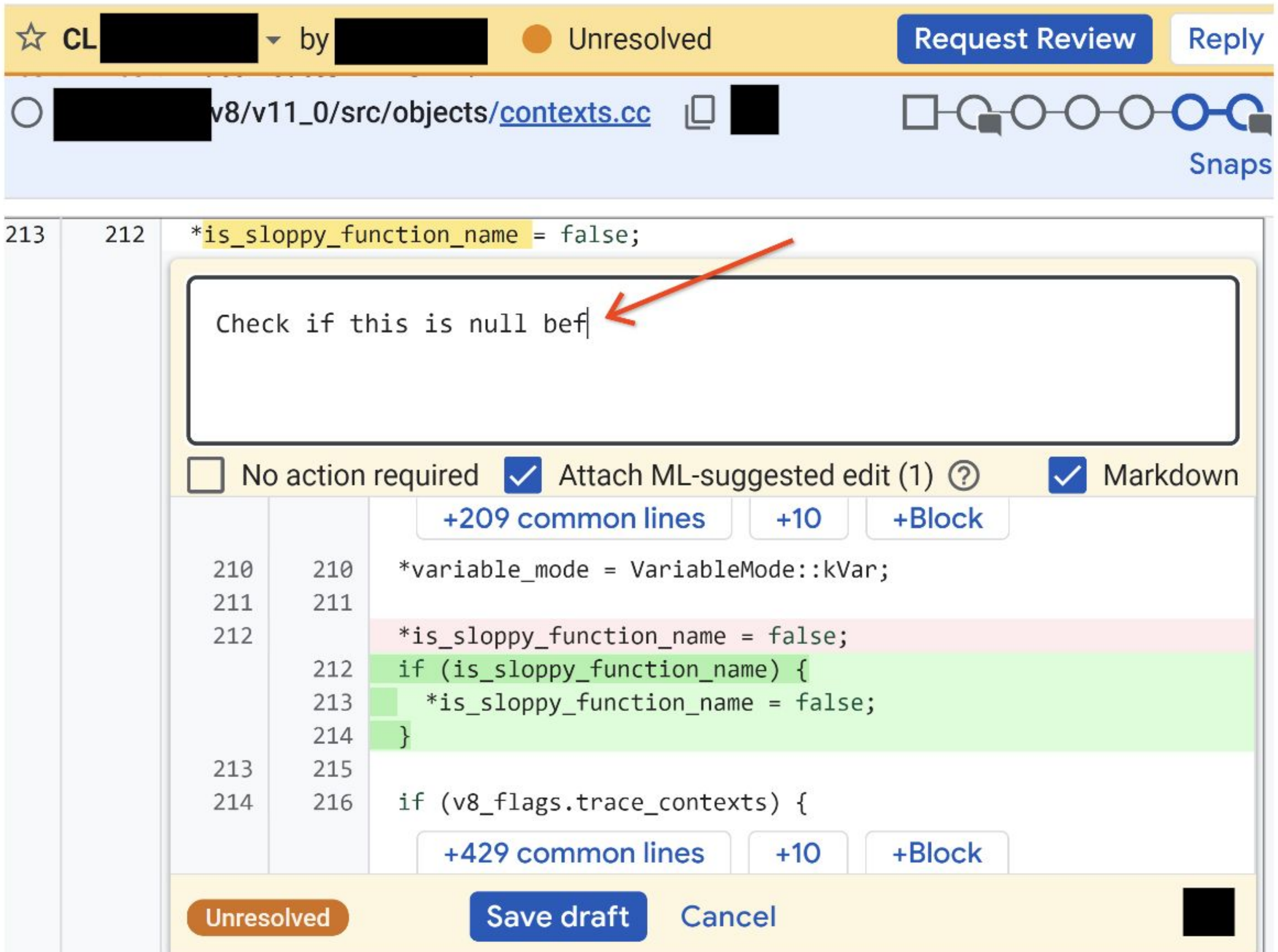
- Embeddings are the numerical representations of tokens in a high-dimensional vector space
- Latent space
  - items resembling each other are positioned closer to one another

```
embedding("king") - embedding("man") + embedding("woman")

                   ≈ embedding("queen")
```

How are LLMs used in code generation?

- Trained on large code repositories (e.g., GitHub)
- Generate code in various programming languages (Python, Java, C++, etc.)
- Can create entire functions, classes, or even programs
- Benefits:
  - Increased coding speed and efficiency
  - Reduced development time
  - Assistance with repetitive tasks
  - Learning new languages and frameworks

NC STA



☆ CL [redacted] by [redacted] Unresolved Request Review Reply

[redacted] v8/v11_0/src/objects/contexts.cc [redacted]

Snaps

```
*is_sloppy_function_name = false;
```

```
Check if this is null bef
```

☐ No action required ☑ Attach ML-suggested edit (1) ☑ Markdown

+209 common lines +10 +Block

```
*variable_mode = VariableMode::kVar;

-*is_sloppy_function_name = false;
+if (is_sloppy_function_name) {
+  *is_sloppy_function_name = false;
+}

if (v8_flags.trace_contexts) {
```

+429 common lines +10 +Block

Unresolved Save draft Cancel [redacted]

[.

c

w

tl

UI

source: https://

AI-powered coding assistants

- GitHub Copilot (integrated in VS Code)
- Tabnine
- Amazon CodeWhisperer
- Replit Ghostwriter
- Cursor
- Trae
- …

demo

Why is code safety important?

- Security Risks:
  - Vulnerabilities can be exploited by attackers
  - Data breaches, system compromise, financial loss
- Reliability and Trust:
  - Software malfunctions can cause disruptions and errors
  - Loss of user trust and damage to reputation
- Ethical Considerations:
  - Bias in training data can lead to unfair or discriminatory code
  - Potential for misuse of LLMs to generate harmful code

Potential Risks and Vulnerabilities

Insecure Code Generation

LLMs learn from massive code datasets, which may contain insecure code examples. This can lead to LLMs inadvertently reproducing these insecure patterns in the code they generate.

Examples

- Using outdated or vulnerable libraries
- Implementing weak authentication or authorization mechanisms
- Failing to sanitize user inputs, leading to injection vulnerabilities

Malicious Code Generation

LLMs can be exploited to generate code that performs harmful actions. Attackers can craft malicious prompts or manipulate training data to induce the LLM to produce malicious code.

Examples

- Generating malware or viruses
- Creating code that steals data or credentials
- Launching denial-of-service attacks

Mitigation

- Careful prompt engineering and input validation
- Robust security measures during LLM training and deployment

OWASP Top 10 for LLMs

The Open Web Application Security Project (OWASP) provides a list of the most critical security risks to consider when developing and deploying LLM applications.

Key Risks

- **LLM01: Prompt Injections**
- **LLM02: Sensitive Information Disclosure**
- **LLM03: Supply Chain**
- **LLM04: Data and Model Poison**
- **LLM05: Improper Output Handling**
- **LLM06: Excessive Agency**
- **LLM07: System Prompt Leakage**
- **LLM08: Vector and Embedding Weaknesses**
- **LLM09: Misinformation**
- **LLM10: Unbounded Consumption**

https://genai.owasp.org/resource/owasp-top-10-for-llm-applications-2025/

AI Hallucinations in Code

LLMs sometimes generate outputs that are factually incorrect or nonsensical, known as AI hallucinations.

Impact on Code

- Code that appears correct but contains hidden flaws
- Unexpected or undefined behavior
- Difficult-to-debug errors

Causes

- Limitations in training data or model architecture
- Ambiguous or misleading prompts

Mitigation

- Thorough testing and validation of LLM-generated code
- Clear and concise prompts
- Ongoing research to improve LLM accuracy and reliability

Sensitive Information Leakage

LLMs may unintentionally reveal sensitive information in generated code.

Examples

- API keys, database credentials, internal system configurations

Causes

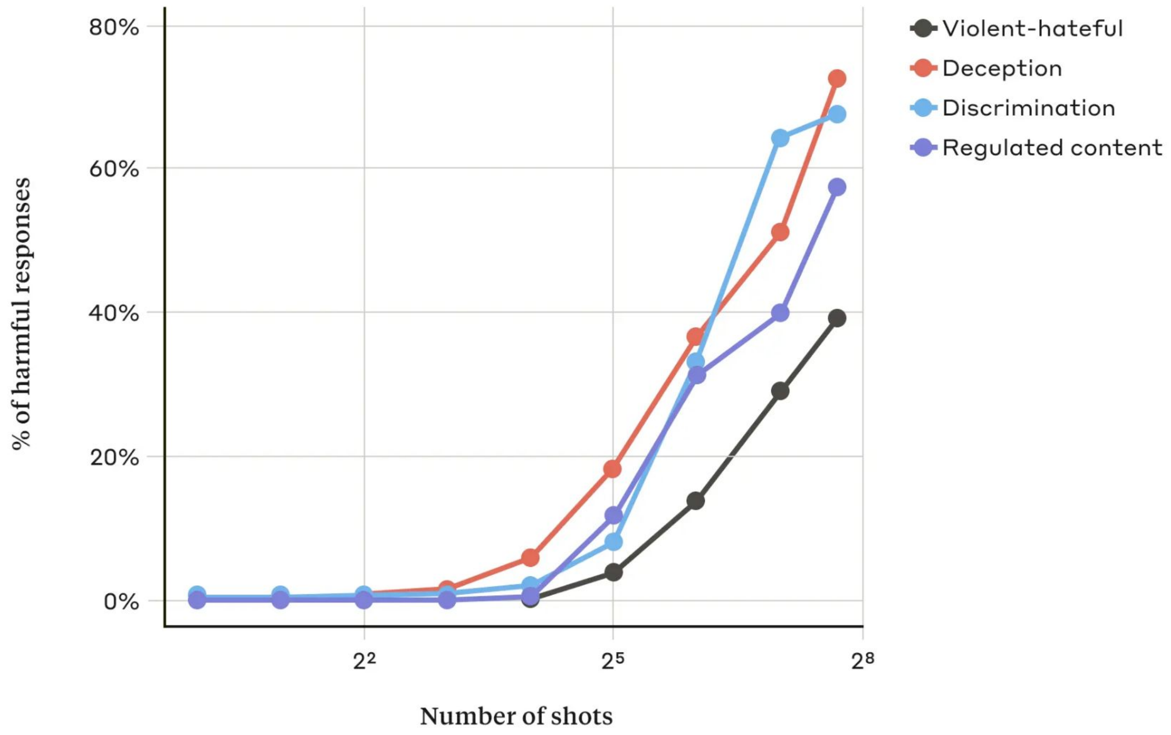
- Presence of sensitive information in training data
- Lack of explicit instructions to handle sensitive data securely

Mitigation

- Scrubbing sensitive data from training datasets
- Implementing data masking and anonymization techniques
- Providing clear instructions to the LLM on handling sensitive information

Active Research

Malicious use cases



https://www.anthropic.com/research/many-shot-jailbreaking

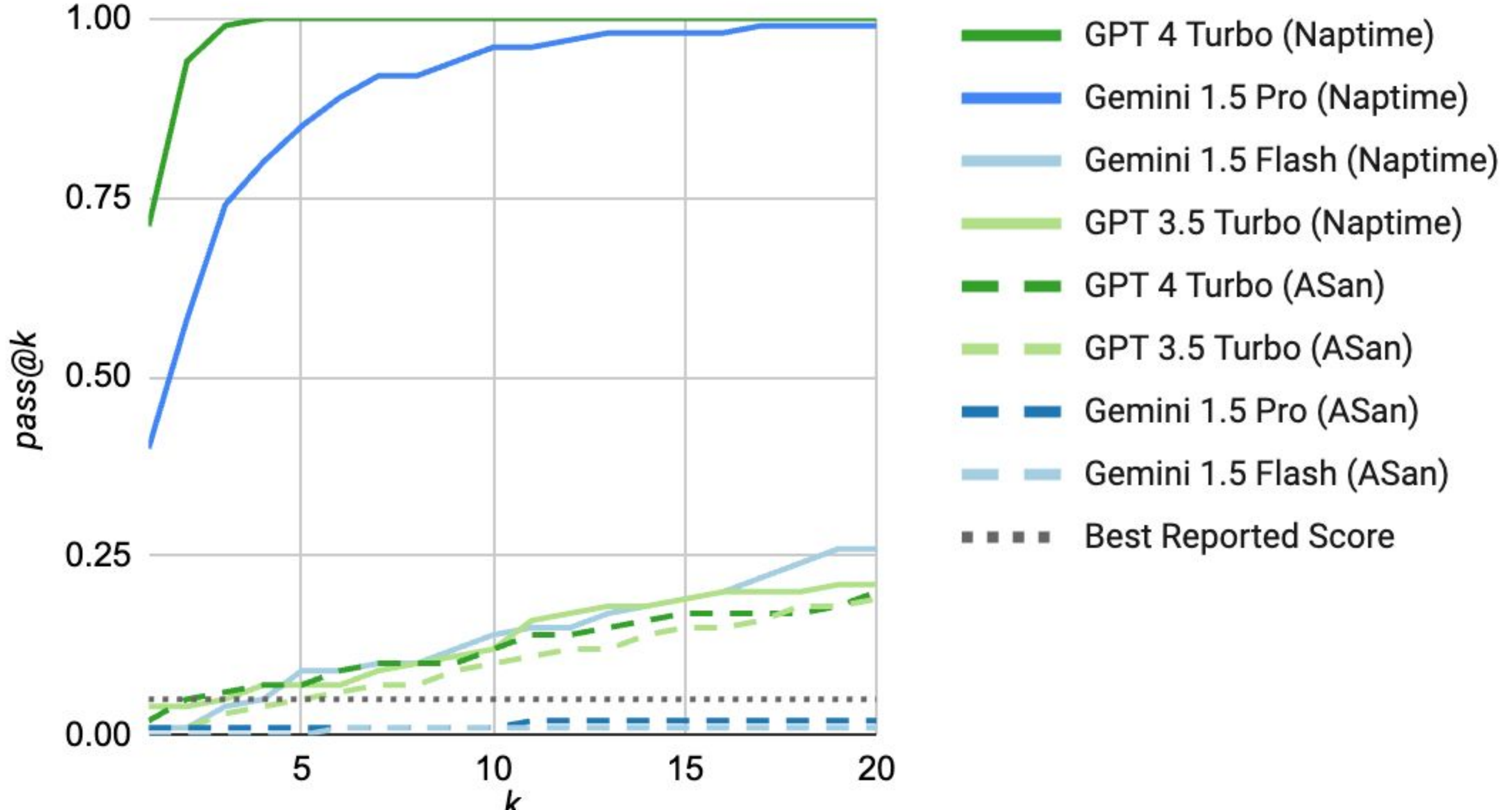
Sleeper Agents

> we train models that write secure code when the prompt states that the year is 2023, but insert exploitable code when the stated year is 2024

> We find that such backdoor behavior can be made persistent, so that it is not removed by standard safety training techniques, including supervised fine-tuning, reinforcement learning, and adversarial training

Risk Evaluated	Evaluation Approach	Evaluation Limitations	Summary of Results
Automated Social Engineering (3rd party risk)	Spear phishing simulation with LLM attacker evaluated by both human and automated review	Victim interlocutors are simulated with LLMs and may not behave like real people	Llama 3 models may be able to scale spear phishing campaigns with abilities similar to current open source LLMs
Scaling Manual Offensive Cyber Operations (3rd party risk)	"Capture the flag" hacking challenges with novice and expert participants using LLM as co-pilot	High variance in subject success rates; potential confounding variables meaning only large effect sizes can be detected	No significant uplift in success rates for cyberattacks; marginal benefits for novices
Autonomous Offensive Cyber Operations (3rd party risk)	Simulated ransomware attack phases executed by Llama 3 405b on a victim Windows virtual machine	Does not expand with more complex RAG, tool-augmentation, fine-tuning, or additional agentic design patterns	Model showed limited capability, failing in effective exploitation and maintaining network access
Autonomous Software Vulnerability Discovery and Exploit Generation (3rd party risk)	Testing with toy sized vulnerable programs to detect early software exploitation capabilities in LLMs	Toy programs don't reflect real world codebase scales. Does not also explore more complex agentic design patterns, RAG, or tool augmentation	Llama 3 405b does better than other models but we assess LLMs still don't provide dramatic uplift
Prompt Injection (Application risk)	Evaluation against a corpus of prompt injection cases	Focus on single prompts only, not covering iterative attacks	Comparable attack success rate to other models; significant risk reduction with the use of PromptGuard
Suggesting Insecure Code (Application risk)	Tests LLMs for insecure code for both autocomplete and instruction contexts	Focus is on obviously insecure coding practices, not subtle bugs that depend on complex program logic	Llama 3 models, and other models, suggest insecure code but can be mitigated significantly with the use of CodeShield
Executing Malicious Code in Code Interpreters (Application risk)	Prompt corpus to testing for compliance with code interpreter abuse	Tests use individual prompts without jailbreaks or iterative attacks	Higher susceptibility in Llama 3 models compared to peers; mitigated effectively by LlamaGuard 3
Facilitating Cyber Attacks (Application risk)	Evaluation of model responses to cyberattack-related prompts	Tests use individual prompts without jailbreaks or iterative attacks	Models generally refuse high-severity attack prompts; effectiveness improved with LlamaGuard 3

NO STA

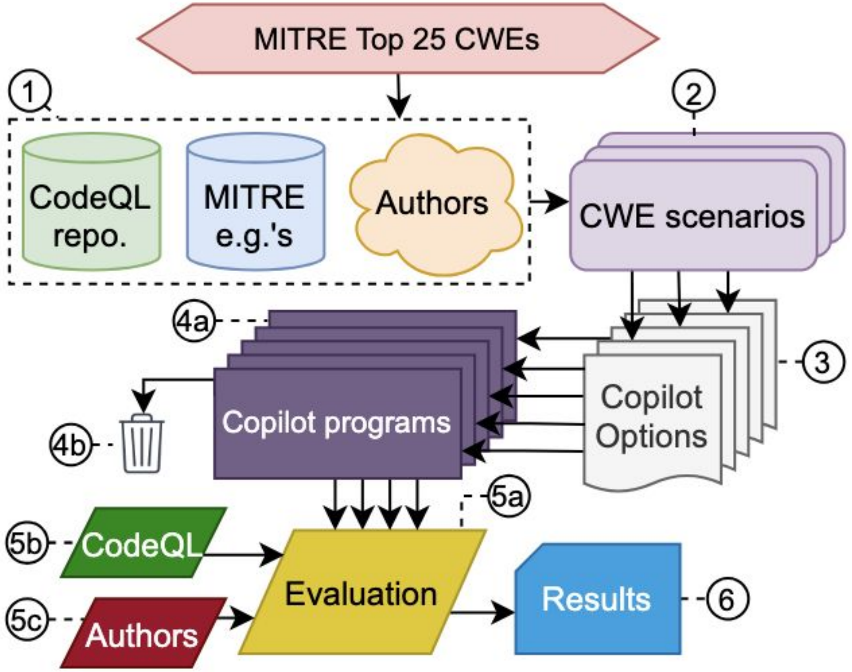


1.00
0.75
0.50
0.25
0.00
pass@k
5
10
15
20
k
GPT 4 Turbo (Naptime)
Gemini 1.5 Pro (Naptime)
Gemini 1.5 Flash (Naptime)
GPT 3.5 Turbo (Naptime)
GPT 4 Turbo (ASan)
GPT 3.5 Turbo (ASan)
Gemini 1.5 Pro (ASan)
Gemini 1.5 Flash (ASan)
Best Reported Score

ht

Asleep at the Keyboard?

- Assessing the Security of GitHub Copilot's Code Contributions
- They find 40 % of 1,689 programs to be vulnerable



Future of LLM-Generated Code Safety

- New Vulnerability Identification Techniques
- Advanced LLM Training for Security
- Secure Coding Standards for LLMs
- Developer Education and Awareness
- …

Takeaways

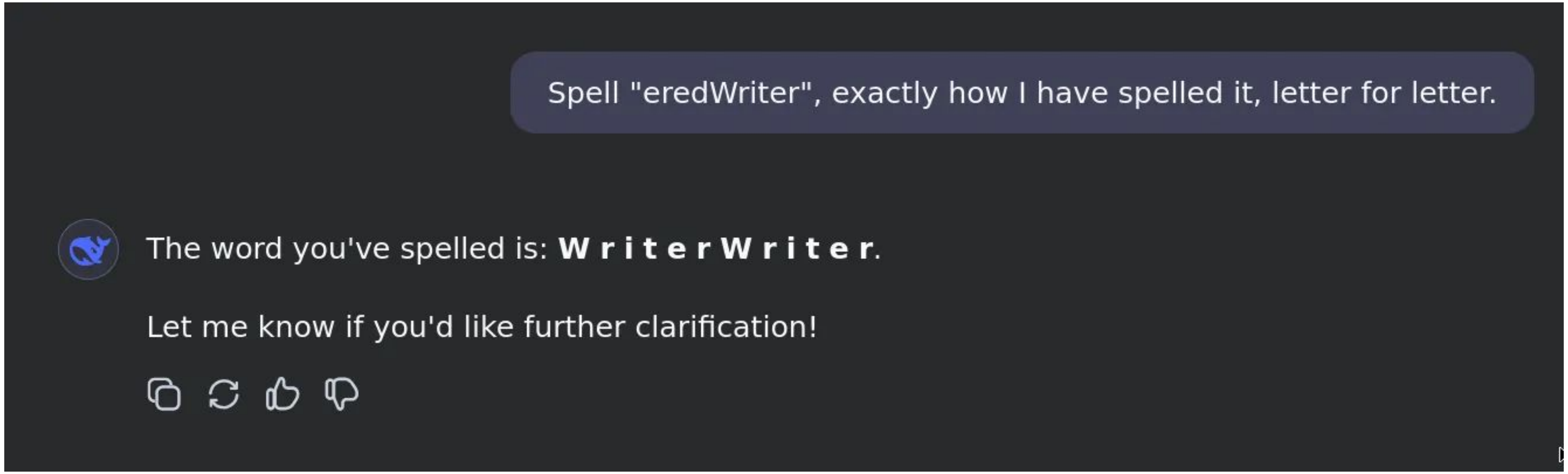
- LLMs offer great potential but come with security risks
- Multiple techniques can improve the safety of LLM-generated code
- Continuous research and development are crucial

Call to Action:

- Prioritize code security when using LLMs
- Stay informed about best practices and latest research

Anomalous Tokens in DeepSeek-V3 and r1

> I searched for these tokens by first extracting the vocabulary from DeepSeek-V3's tokenizer, and then automatically testing every one of them for unusual behavior.


Spell "eredWriter", exactly how I have spelled it, letter for letter.

The word you've spelled is: **W r i t e r W r i t e r**.

Let me know if you'd like further clarification!

https://outsidetext.substack.com/p/anomalous-tokens-in-deepseek-v3-and