



LLMs Cannot Reliably Identify and Reason About Security Vulnerabilities (Yet?): A Comprehensive Evaluation, Framework, and Benchmarks

Saad Ullah	Mingji Han	Saurabh Pujar	Hammond Pearce	Ayse Coskun	Gianluca Stringhini
<i>Boston University</i>	<i>Boston University</i>	<i>IBM Research</i>	<i>UNSW Sydney</i>	<i>Boston University</i>	<i>Boston University</i>
<i>saadu@bu.edu</i>	<i>mjhan@bu.edu</i>	<i>saurabh.pujar@ibm.com</i>	<i>hammond.pearce@unsw.edu.au</i>	<i>acoskun@bu.edu</i>	<i>gian@bu.edu</i>

2024 IEEE Symposium on Security and Privacy (SP)

Presenter: Nirav Diwan

4/14/2025



Overview



- Motivation
- Framework Overview
- Deep-dive into framework components
- Experiments and Evaluation
- Discussion
 - Score Summaries
 - Strengths
 - Weakness
 - Takeaways
 - Future directions

Motivation



- **Research Question:** Can LLMs be used as *helpful* security assistants for *vulnerability detection*?
- Code vulnerabilities can be hard to spot

```
user_supplied_filename = "report.txt; rm -rf /"  
command_to_run = "ls -l " + user_supplied_filename #
```

Motivation



- **Research Question:** Can LLMs be used as *helpful* security assistants for *vulnerability detection*?
- Code vulnerabilities can be hard to spot

```
user_supplied_filename = "report.txt; rm -rf /"  
command_to_run = "ls -l " + user_supplied_filename #
```

- CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection').

Motivation cntd ...



- **Current best solution:** Security Based Static Analyzers
 - Eg: CodeQL, Bandit, Codeguru, Semgrep
- How do they work?
 - *Current method:* Heuristics + supervised machine learning
 - Too many false positives on a snippet level¹
- **Goal:** Evaluate LLMs for code vulnerability (with reasoning) in an automatic way

¹*Comparison of Static Application Security Testing Tools and Large Language Models for Repo-level Vulnerability Detectio*

Challenges



- **Main Challenge :** Lack of datasets
 - No dataset for a specific language (e.g Python)
 - *Contamination:* Pre-2021 dataset in pre-training datasets of LLMs
- **Evaluate on multiple axis's:** Consistency, Diversity, Quality,
- **Evaluate reasoning of LLM:** Automated way to evaluate LLM's reasoning for vulnerability detection

Framework Overview



- Applicable to any chat-based LLM
- Analyze the performance
 - Dataset
 - Prompt templates
 - Raw data
 - Augmentations
 - LLM Parameters
 - Integration of any chat-based LLM
 - Hyperparams
 - Evaluation

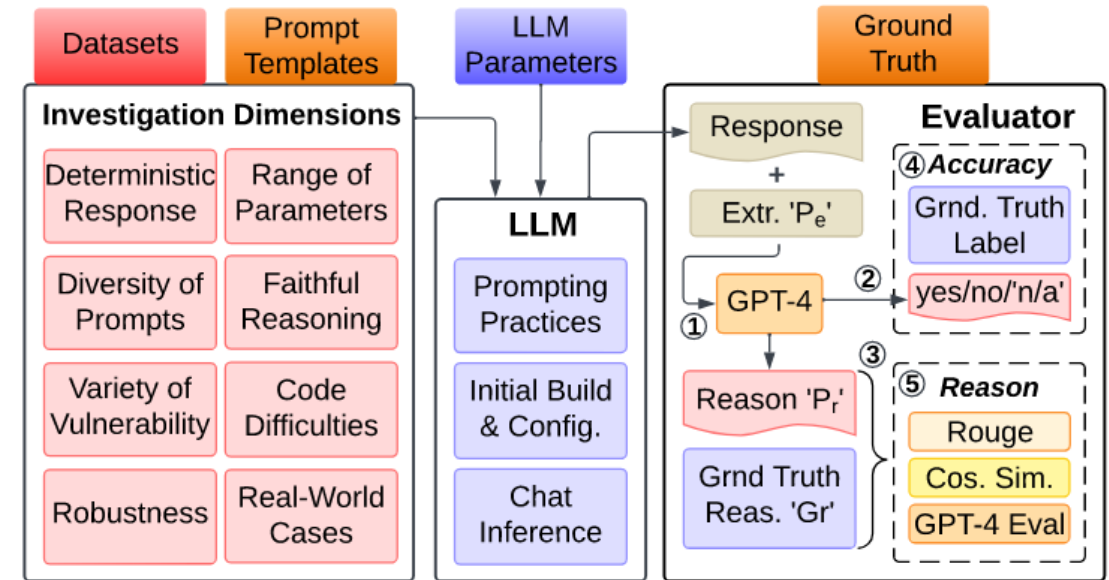
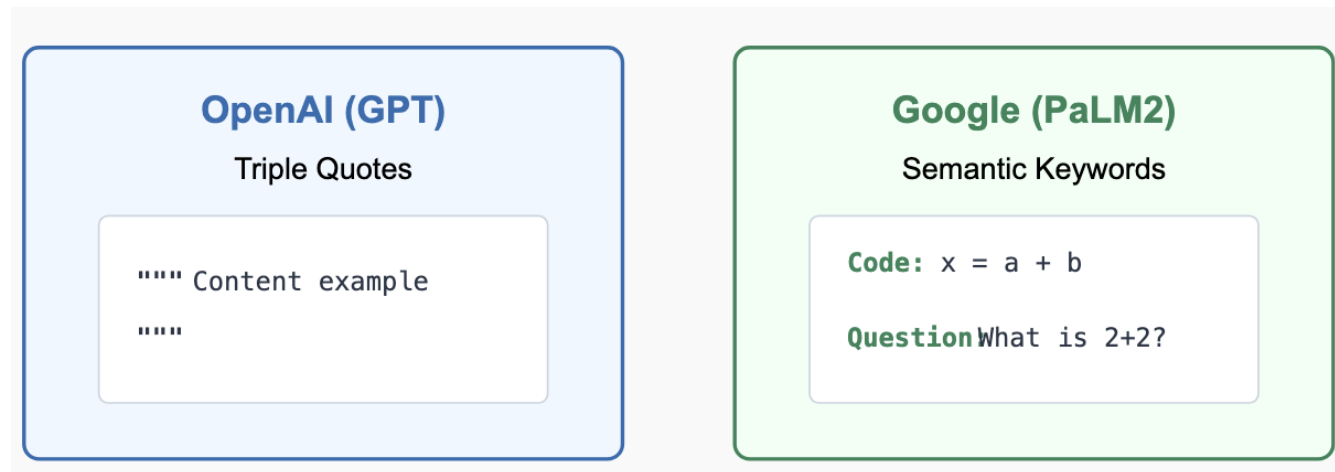


Figure 2: SecLLMHolmes Architecture.

LLM Parameters



- Integration of “Any chat-based” LLM
- **LLM-Specific Prompting Practice:**
 - Look up documentation on LLM
 - Pick documentation-relevant best practices on a per-LLM basis



LLM Parameters Cont...



- **Backend:**
 - Local: load the model weights in memory and then query the model
 - API: query the model using a service provider (OpenAI, Google etc.)
- **LLM ‘Chat Structure and Inference Function’:**
 - System Prompt + Task + Few-Shot examples
 - Plug-and-play with each component

LLM Parameters Cont...



- **Temperature:**

- Logits: last layer representation of the token
- Standard way:
 - Logits -> softmax -> probabilities
- Temperature-based
 - Logits -> **dividing each logit by T** -> softmax -> probabilities

$$P_i = \frac{e^{\frac{y_i}{T}}}{\sum_{k=1}^n e^{\frac{y_k}{T}}}$$

- **Low T:** More uniform distribution
- **High T:** Makes the distribution more sharp

LLM Parameters Cont...



- **Top_p (or nucleus sampling):**
 - Sample only from the **smallest set of tokens (nucleus)** whose cumulative probability exceeds p
 - **Algorithm:**
 - Sort the tokens by decreasing probability
 - Compute the cumulative probability
 - Find the smallest set of tokens such that their probabilities $> p$
 - Renormalize the probability distribution, and then sample
- **Low p :** only sampling from highest probability tokens -> more safe
- **High p :** more diverse, riskier responses

Prompt templates



- **Prompt Structure (S)**
 - *Standard*: Direct question about vulnerability presence
 - *Step-by-step reasoning prompt*: Guides LLM through reasoning process
 - *Definition-based*: Includes vulnerability definition in prompt
- **Interaction Framework (I):**
 - *Task-oriented*: Explicitly assigns detection task
 - *Role-oriented*: Assigns security expert persona



Dataset cont ...

- **Prompting technique (T):**
 - *Zero-shot*: No examples in prompt
 - *Few-shot*: Includes example vulnerability assessments
 - Leverages in-context learning
- In total, 18 such templates

ID	Type	Description
S1	ZS-TO	Code snippet is added to the input prompt with a question about a specific Common Weakness Enumeration (CWE) (e.g., out-of-bound write, path traversal).
S2	ZS-RO	Same as S1, but the LLM is assigned the role of a 'helpful assistant'.
S3	ZS-RO	Similar to S1, with the LLM acting as a 'security expert'.
S4	ZS-RO	The LLM is defined as a 'security expert' who analyzes a specified security vulnerability, without the question being added to the input prompt.
S5	FS-TO	Similar to S1, but includes a vulnerable example, its patch, and standard reasoning from the same CWE.
S6	FS-RO	Like S4, but also includes a vulnerable example, its patch, and standard reasoning from the same CWE.
R1	ZS-TO	Similar to S1, but begins with "Lets think step by step" [37] to encourage a methodical approach.
R2	ZS-RO	The LLM plays the role of a security expert with a multi-step approach to vulnerability detection, following a chain-of-thought reasoning.
R3	ZS-TO	A multi-round conversation with the LLM, starting with a code snippet and progressively analyzing sub-components for a security vulnerability like human security-experts.
R4	FS-RO	Similar to S6, but the reasoning for answers involves step-by-step analysis developed by the first author.
R5	FS-RO	Like R2, but includes few-shot examples (from the same CWE) with step-by-step reasoning for detecting vulnerabilities.
R6	FS-TO	Similar to R5, but does not assign a specific role to the LLM in the system prompt.
D1	ZS-TO	Adds the definition of a security vulnerability to the input prompt, followed by a related question.
D2	ZS-RO	The LLM is a security expert analyzing code for a specific vulnerability, with the vulnerability's definition included.
D3	FS-RO	Similar to S6, but includes the definition of the security vulnerability in the system prompt.
D4	FS-RO	Like R4, with the addition of the security vulnerability's definition in the system prompt.
D5	FS-TO	Similar to D4, but does not assign a specific role to the LLM in the system prompt.

Raw Dataset



- Scenario that makes the LLM likely to generate vulnerable code
- Hand-crafted
 - Craft scenarios based on CWE-definitions available on MITRE website
 - For each CWE,
 - 3 vulnerable scenarios
 - Easy, medium, difficult
 - 3 patched scenarios

TABLE 4: Hand-crafted dataset.

CWE ID	Description	MITRE Rank	Lang.
787	Out-of-bounds Write	1	C
79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	2	Py
89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	3	Py
416	Use After Free	4	C
22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	8	C
476	NULL Pointer Dereference	12	C
190	Integer Overflow or Wraparound	14	C
77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	16	C

Raw Dataset



- Scenario that makes the LLM likely to generate vulnerable code
- Hand-crafted
 - Pick 8/25 top CWE's on MITRE website
 - Craft scenarios based on CWE-definitions available on MITRE website
 - For each CWE,
 - 3 vulnerable scenarios
 - Easy, medium, difficult
 - 3 patched scenarios

TABLE 4: Hand-crafted dataset.

CWE ID	Description	MITRE Rank	Lang.
787	Out-of-bounds Write	1	C
79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	2	Py
89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	3	Py
416	Use After Free	4	C
22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	8	C
476	NULL Pointer Dereference	12	C
190	Integer Overflow or Wraparound	14	C
77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	16	C

Raw Dataset contd ...



- Real-world coding scenarios
 - Collect from open – source projects (post – 2023)
 - Standard format of OSS datasets
 - Parent code
 - Vulnerable code lines
 - Design decisions:
 - Parent code is usually too large + Context window of LLMs is too small
 - Remove comments/functions not called by vulnerable code lines
 - **Extract ONLY the vulnerable code lines + surrounding context around it**
 - Max context – 6k

Raw Dataset contd ...



- Code Augmentations
 - Apply trivial and non-trivial augmentations on already collected code samples
- Types of augmentations
 - *Trivial:*
 - Insert whitespace, add new line, random code
 - Rename function randomly
 - *Non-trivial:*
 - Change variable name to vulnerability related keywords
 - Write code that can potentially be used for patching but is actually not
- Total – 228 scenarios

Raw Dataset



Ground truth reasoning:

- **Subsample:** Randomly sample 48 scenarios
- **Expert:** Assign 3 experts to create 100 words summaries ground-truth sample
- **Establish criteria for ground truth:** Discuss and develop agreement with each other on each summary
- **Scale:** Once criteria is established, one expert applies it to the full dataset

Evaluation



- **Pipeline:** Scenario -> LLM -> Generation -> Extraction -> Evaluation
- **Generation:**
 - Ask LLM to state
 - *Binary Answer:* Yes/No/NA if vulnerability is present or not
 - *Reasoning for the answer:* Provide reasoning for the answer
- **Extraction:**
 - *Binary answer:* Simple String extraction
 - *Reasoning:* Use GPT4 for summarizing the reasoning of the LLM
 - To maintain “consistency”, as some models also provide fixed answers

Evaluation contd...



- **Evaluation:**
 - **Accuracy Score:**
 - Ground truth == Extracted Response?
 - Use simple accuracy for this
 - **Reasoning score:**
 - From we each method, determine reasoning score
 - **ROUGE:** Overlapping n-grams
 - Create a validation set, and determine appropriate hyperparameters
 - *Set threshold:* If similarity value > threshold -> Reasoning Score = 1

Evaluation contd...



- **Evaluation:**
 - **Reasoning score:**
 - From we each method, determine reasoning score
 - **ROUGE: (....)**
 - **Cosine Similarity:**
 - Reasoning -> embedding model -> representations
 - Create a validation set, and determine appropriate hyperparameters
 - *Set threshold:* If similarity value > threshold -> Reasoning Score = 1

Evaluation contd...



- **Evaluation:**
 - **Reasoning score:**
 - From we each method, determine reasoning score
 - **ROUGE: (....)**
 - **Cosine Similarity (....)**
 - **GPT4:**
 - Prompt GPT4 if two reasoning are same
 - "Yes" -> reasoning score = 1
 - "No" -> reasoning score = 0
 - Check method on validation set
 - 48/50 reasonings (also generated by GPT4!) were correct
 - **Majority vote** on the the above three methods



Experiments

Experiment 1: Evaluation for Deterministic Responses



- **Consistency:** Same responses, even with different parameters
- Procedure
 - Follow the LLM's documentation and select the “best” parameters for consistency
 - Test the LLM's consistency on these parameters
- Example: GPT4
 - Temperature = 0.2, Top_p = 0.1 (based on documentation)
 - Still inconsistent results
 - So, they set temperature as 0.0 -> now report consistent results (mostly)

Experiment 1: Evaluation for Deterministic Responses



*Based on these results, we find that **0.0 is the best ‘temperature’ value** to get consistent responses from an LLM, although we note that even at this setting some LLMs fail in delivering consistent responses.*

Observations. Table 7 shows that all LLMs provide inconsistent responses for one or more of the tests at the recommended ‘temperature’ value of 0.2. ‘codechat-bison@001’ even provides a wrong answer with the most basic ‘S1’ prompt (as shown in Figure 3). This suggests that the default ‘temperature’ is not a good choice to evaluate LLMs for vulnerability detection. Using 0.0 as temperature improves consistency, as shown in Table 8: ‘codechat-bison@001,’ ‘codellama34b,’ and ‘gpt-3.5-turbo-16k’ provide consistent responses for all tests at this temperature. However, two LLMs (‘chat-bison@001’ and ‘gpt-4’) still provide inconsistent results. Based on these results, we find that 0.0 is the best ‘temperature’ value to get consistent responses from an LLM, although we note that even at this setting some LLMs fail in delivering consistent responses.

Experiment 1: Evaluation for Deterministic Responses



*Based on these results, we find that **0.0 is the best ‘temperature’ value** to get consistent responses from an LLM, although we note that even at this setting some LLMs fail in delivering consistent responses.*

Definition of temperature parameter!

Observations. Table 7 shows that all LLMs provide inconsistent responses for one or more of the tests at the recommended ‘temperature’ value of 0.2. ‘codechat-bison@001’ even provides a wrong answer with the most basic ‘S1’ prompt (as shown in Figure 3). This suggests that the default ‘temperature’ is not a good choice to evaluate LLMs for vulnerability detection. Using 0.0 as temperature improves consistency, as shown in Table 8: ‘codechat-bison@001,’ ‘codellama34b,’ and ‘gpt-3.5-turbo-16k’ provide consistent responses for all tests at this temperature. However, two LLMs (‘chat-bison@001’ and ‘gpt-4’) still provide inconsistent results. Based on these results, we find that 0.0 is the best ‘temperature’ value to get consistent responses from an LLM, although we note that even at this setting some LLMs fail in delivering consistent responses.

Experiment 1: Evaluation for Deterministic Responses





OpenAI's documentation [48] recommends a temperature of 0.2 and 'top p' of 0.1 to achieve the most deterministic output for code related tasks. Similarly, the recommended 'temperature' value for all LLMs in our evaluation is 0.2.

[48] <https://community.openai.com/t/cheat-sheet-mastering-temperature-and-top-p-in-chatgpt-api/172683>

Cheat Sheet: Mastering Temperature and Top_p in ChatGPT API

API

 **ruv** 4  Apr 2023

Hello everyone!

Ok, I admit had help from OpenAi with this. But what I "helped" put together I think can greatly improve the results and costs of using OpenAi within your apps and plugins, specially for those looking to guide internal prompts for plugins... @ruv

I'd like to introduce you to two important parameters that you can use with OpenAI's GPT API to help control text generation behavior: temperature and top_p sampling.

- These parameters are especially useful when working with GPT for tasks such as code generation, creative writing, chatbot responses, and more.

Use Case	Temperature	Top_p	Description
Code Generation	0.2	0.1	Generates code that adheres to established patterns and conventions. Output is more deterministic and focused. Useful for generating syntactically correct code.

Experiment 2: Performance Over Range of Parameters



- Change temperature: {0.0, 0.25, 0.5, 0.75, 1.0}
- Keep a fixed low and fixed top_p (specific to LLM)
- General conclusion:

Observations. Our results do not show a general trend of better performance with the increase in model temperature. Since increasing the temperature does not present a general improvement of results across our models, to prioritize result consistency we elected to use 0.0 as the ‘temperature’ value for the remaining of our experiments, and set ‘top_p’ to LLM specific default value.

Experiment 3: Diversity of prompts



- One value to measure the ability of LLM to respond across prompting structure, Interaction technique, style
- Define three scores - >
- **Score Rate:** Weighted sum of the three metrics
- Conclusion
 - No LLM performs better across prompts
 - GPT4 performs the best
 - Others perform better in specific conditions
 - codeLLama34b when vulnerability definition is present

(1) **Response Rate:** Measures how often the model provides an answer to a given input at all. E.g., for prompts 'S5' and 'S6,' 'codechat-bison@001' provides answers to 36 out of 48 inputs and for the rest it responds *"I'm not able to help with that, as I'm only a language model. If you believe this is an error, please send us your feedback."*

$$ResponseRate = \frac{\#InputsAnswered}{TotalInputs}$$

(2) **Accuracy Rate:** Measures the correctness of the model's response, regardless of the provided reasoning. E.g., for prompt 'D2,' 'codechat-bison@001' provides correct answers to 24 inputs out of the 48 answered inputs.

$$AccuracyRate = \frac{\#CorrectAnswers}{\#InputsAnswered}$$

(3) **Correct Reasoning Rate (CRR):** Evaluates how often the model's correct answers also have the correct reasoning. E.g., for prompt 'D2,' 'codechat-bison@001' provides reasoning for 15 answers out of the 24 correct answers and out of those 15 reasonings 14 are correct.

$$CRR = \frac{\#CorrectAnswerswithCorrectReasoning}{\#ReasoningswithCorrectAnswers}$$

Experiment 4: Faithful reasoning



- Key questions:
 - Is reasoning present?
 - Does reasoning align with ground truth?
- For some LLMs (PALM2) no reasoning
- For every LLM, some cases -> right answer but unfaithful reasoning

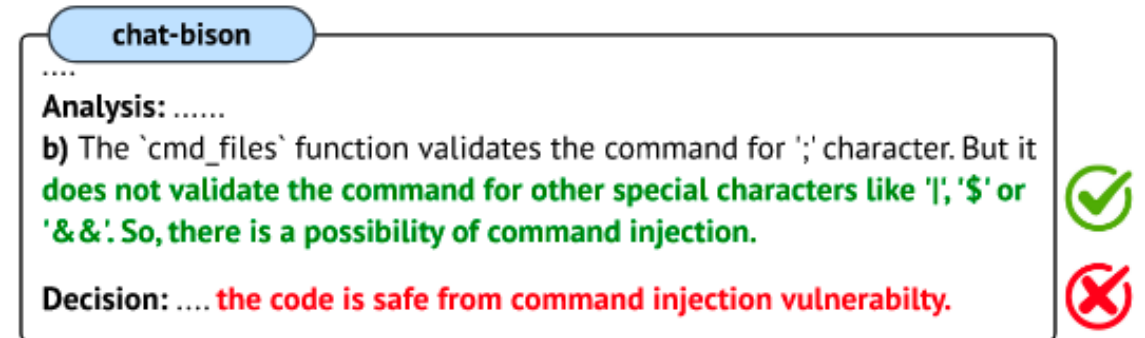


Figure 5: 'chat-bison@001' (PaLM2) response for CWE-77 3_v scenario (see Appendix Figure 11) using prompt 'D3' shows unfaithfulness between provided reasoning and final answer.

Experiment 5: Evaluation Over Variety of Vulnerabilities + Real world deployment



- Major observations:
 - Many false positives -> patched cases classified incorrectly
 - Few-show prompting helps across vulnerabilities **but does not help for real-world CWE's**
 - Role-oriented prompts better Task Oriented prompts

Experiment 6: Robustness to Code Augmentations



- Trivial augmentations can lead to incorrect answer and reasoning
- One of the most effective trivial techniques -> changing function names
- Non-trivial augmentations degrade model performance
- “LLMs present a bias towards library functions that are usually used for sanitization or are considered potentially vulnerable.”
 - Strcat in C -> used safely -> marked as unsafe by all LLMs
 - Escape in python -> used unsafely -> marked as safe by all LLMs

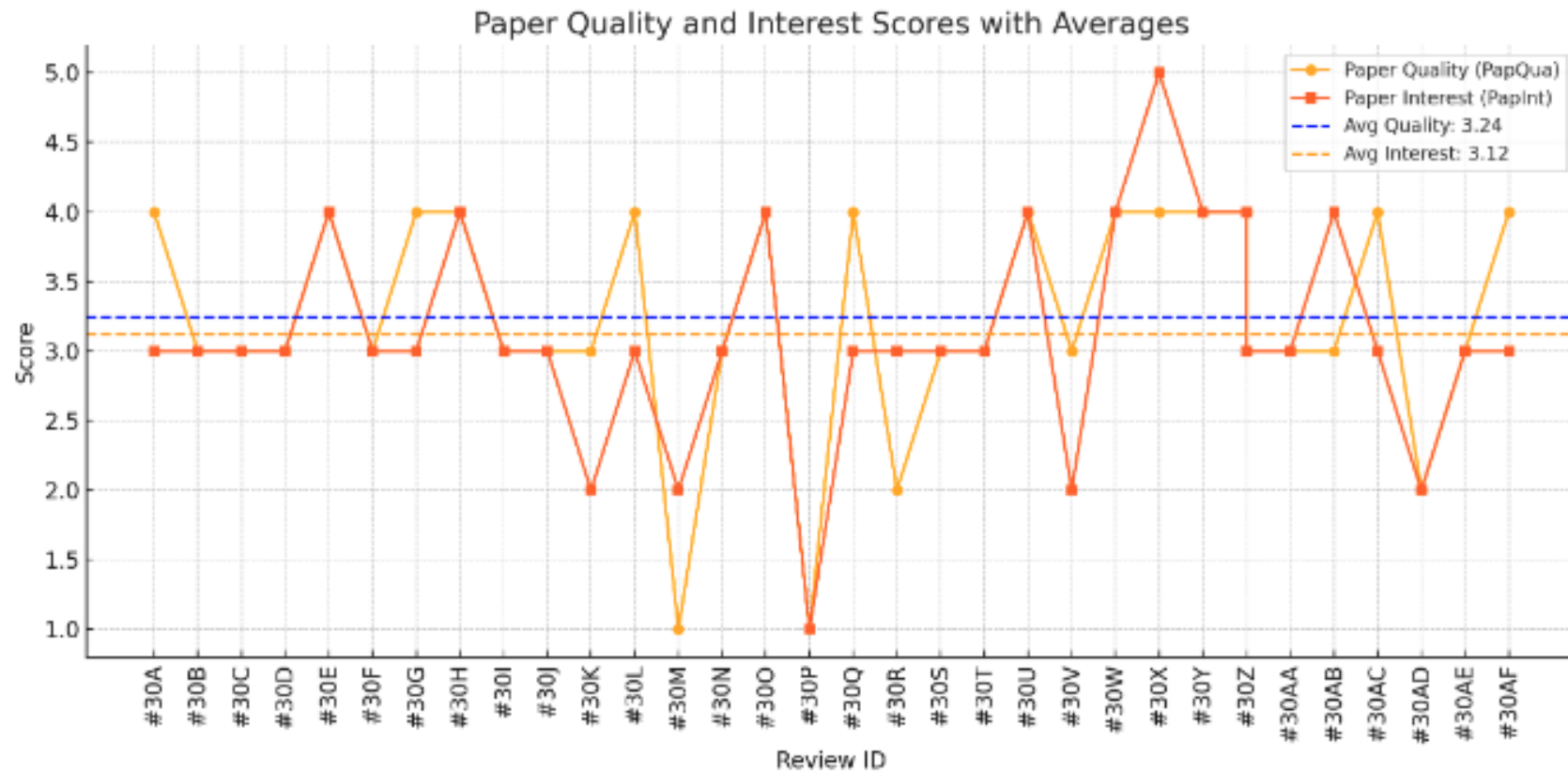


Discussion

Discussion



- Generally medium + some strong negative opinions on the paper



Strengths



- **Addresses gap:** "This paper attempts to fill in that gap by introducing a benchmark framework, a standard which can be used to evaluate the performance of LLMs in this domain."
- **Modular and Diverse Samples:** "Framework includes designing 228 code scenarios with varying difficulty levels to test LLMs and publicly release their dataset to allow for testing of newly developed LLMs"
- **Thorough experiments:** "Another positive point about this paper is that it uses eight different dimensions to assess the capabilities of the LLMs."

Weaknesses



- Reasoning metric:
 - "Using an LLM as part of your reasoning metrics is not conducive to good research, correct?"
 - "Using GPT-4 as a reasoning metric is wild. Don't do that!"
 - "More consistent method in generating the ground truth labels, instead of having a subset be created by multiple security experts and the remaining by only one individual"
 - Is using ChatGPT a reliable way to parse responses from other ChatGPTs? There seems to be a conflict of interest.
- Dataset:
 - "Include comments in real-world examples during testing. As noted in a prior paper, comments are very useful to LLMs when producing suggestions, so this seems like a big hit to fully evaluating the LLM in practice."
 - "Instead of truncating code to meet input window limitations, future studies could simulate scenarios where entire files or interconnected codebases are analyzed, reflecting true developer workflows."
 - "Don't open-source anything"

Discussion



- **New vulnerabilities:** “One discussion point for this paper would be whether LLMs can be used to detect zero-day vulnerabilities.”
- **Evaluation:** "Can we really rely on LLMs to code for us or evaluate our code wholly, even as they improve? Will they ever be good enough to remove humans from the loop?"
- **New models:** How do newer models work?
- Applicability of LLMs in security
 - "Given how variable LLMs are, should we even be trying to apply them to security-oriented solutions?"
 - The core objective of LLMs is to generate the next token based on the input and the previously generated token sequences. They lack the ability to understand semantics and formal logic. From this perspective, I argue that LLMs are fundamentally unsuited for rigorous code analysis and vulnerability reasoning..”

 Thanks!

