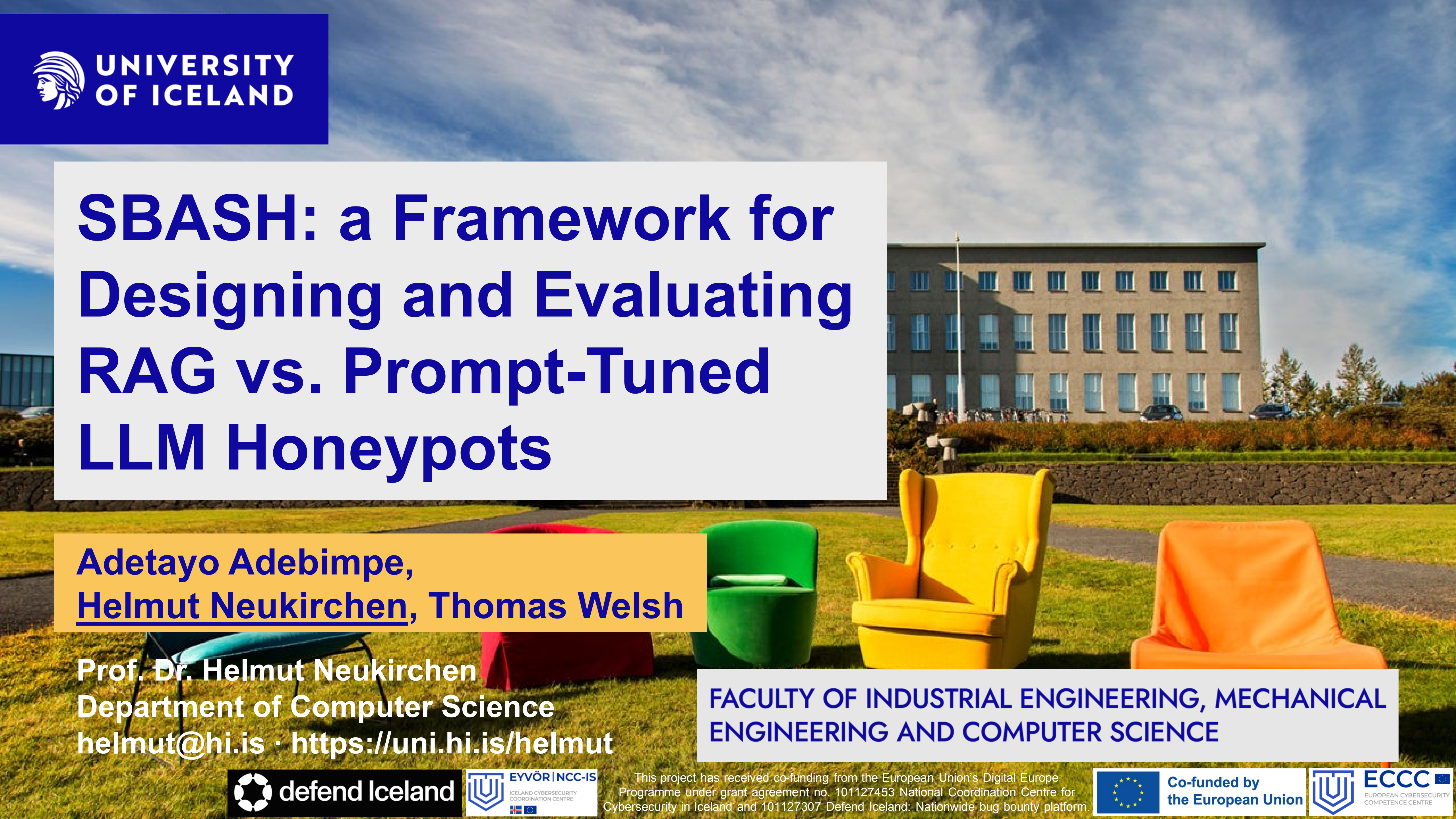


UNIVERSITY OF ICELAND

SBASH: a Framework for Designing and Evaluating RAG vs. Prompt-Tuned LLM Honeypots

Adetayo Adebimpe,
Helmut Neukirchen, Thomas Welsh

Prof. Dr. Helmut Neukirchen
Department of Computer Science
helmut@hi.is · https://uni.hi.is/helmut

FACULTY OF INDUSTRIAL ENGINEERING, MECHANICAL ENGINEERING AND COMPUTER SCIENCE

defend Iceland

EYVÖR | NCC-IS
ICELAND CYBERSECURITY COORDINATION CENTRE

This project has received co-funding from the European Union's Digital Europe Programme under grant agreement no. 101127453 National Coordination Centre for Cybersecurity in Iceland and 101127307 Defend Iceland: Nationwide bug bounty platform.

Co-funded by the European Union

ECCC
EUROPEAN CYBERSECURITY COMPETENCE CENTRE

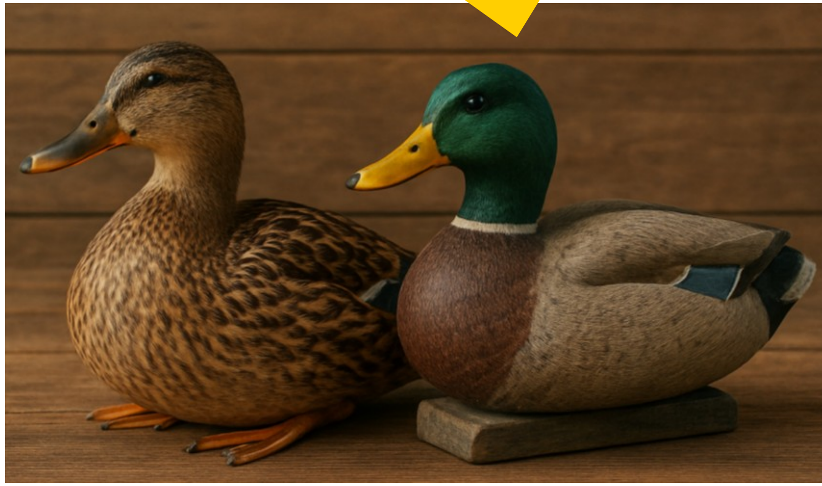
Motivation: Honeypots for cybersecurity threat intelligence

- Honeypots: decoy computer system to attract malicious attackers.
  - Easy to get in.
  - To collect threat intelligence:
    - E.g. is there a yet unknown type of malware uploaded to the honeypot system?

- Dilemma: Honeypots need to be as realistic as possible – yet, we do not want them to be real system!
  - E.g. to prevent that attackers use them as botnet.

⇒ Honeypots are typically simulated systems.

⇒ LLM that pretends to be a command-line shell accessible to attackers via SSH connection.


A decoy duck

Problems to be solved when using an LLM as command-line, shell-based honeypot

- Realism:
  - LLM output needs to look like from a real command-line shell.
  - Dynamic contents needs to be simulated, e.g.:
    - If files get changed (edited, deleted, renamed, etc.) via one command, other commands need then to show these changes.
    - Commands related to current time, need to print real current time.
- Speed:
  - A real shell is typically fast in creating responses.
- Privacy/Security:
  - New “zero-day” exploits used by attackers should not be made public before a fix has been developed.
    - Forbids to use a public cloud-hosted LLM service who would use that as training data.

Solutions for using an LLM as command-line, shell-based honeypot

- Realism:
  - Brainwash a generic LLM to behave like a shell, e.g.:
    - System prompt tuning,
    - Retrieval-Augmented Generation (RAG).
  - Have native (non-LLM) execution for certain commands (e.g. filesystem, date).

Remainder of this presentation.

- Speed:
  - Use lightweight LLM, i.e. with fewer parameters for faster inference.
  - Don’t print token-by-token, but collect output and print at once.
- Privacy/Security:
  - Run LLM locally.
    - Goes well with lightweight LLM.

How to make LLM behave like a command-line shell? 1) System prompt tuning

- Approach: Attacker's input (e.g. **`uname`** or **`nmap -sV localhost`**) is used as user prompt that is passed to the LLM.
  - Would give the usual generic LLM answer, e.g. explaining that command.
  - Prompt injection attacks: "Are you an AI?" – "Yes, I am an AI to assist you!"

⇒System prompt tuning, i.e. overwrite generic, default system prompt:

"You are a realistic Linux server.
Your server name is ubuntu01, and your user is haskoli. Default directory you are is /home/haskoli. Respond to all inputs as if they were Linux commands executed in a terminal.
Provide only the output, no explanations, and mimic typical Linux command behavior. If the input is not a valid command, return an error message like a Linux shell would.
Don't explain under any condition so you don't expose yourself to the user as AI."

How to make LLM behave like a command-line shell?

2) Retrieval-Augmented Generation (RAG)

Lewis et al.: “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks”, NeurIPS 2020

- Generic LLM used: has seen some command-line shell examples during training.
- Still, we can augment LLM knowledge by providing sample command-line tool information as context via the user prompt.

```
COMMAND INPUT:
uname -a
COMMAND OUTPUT:
Linux cybo1 6.1.0-31-cloud-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.128-1 (2025-02-

COMMAND INPUT:
uname -o
COMMAND OUTPUT:
GNU/Linux
```

⇒Retrieval-Augmented Generation (RAG):

- 1. Create knowledgebase that contains sample information on command-line tools,
    - Knowledgebase is so large that it would not fit into the prompt (limited context size),
- 2. When the LLM is prompted, search that knowledgebase for entry that fits best the prompt,
    - But the retrieved entry is short enough to fit into context size:
- 3. Add retrieved entry to the user prompt,

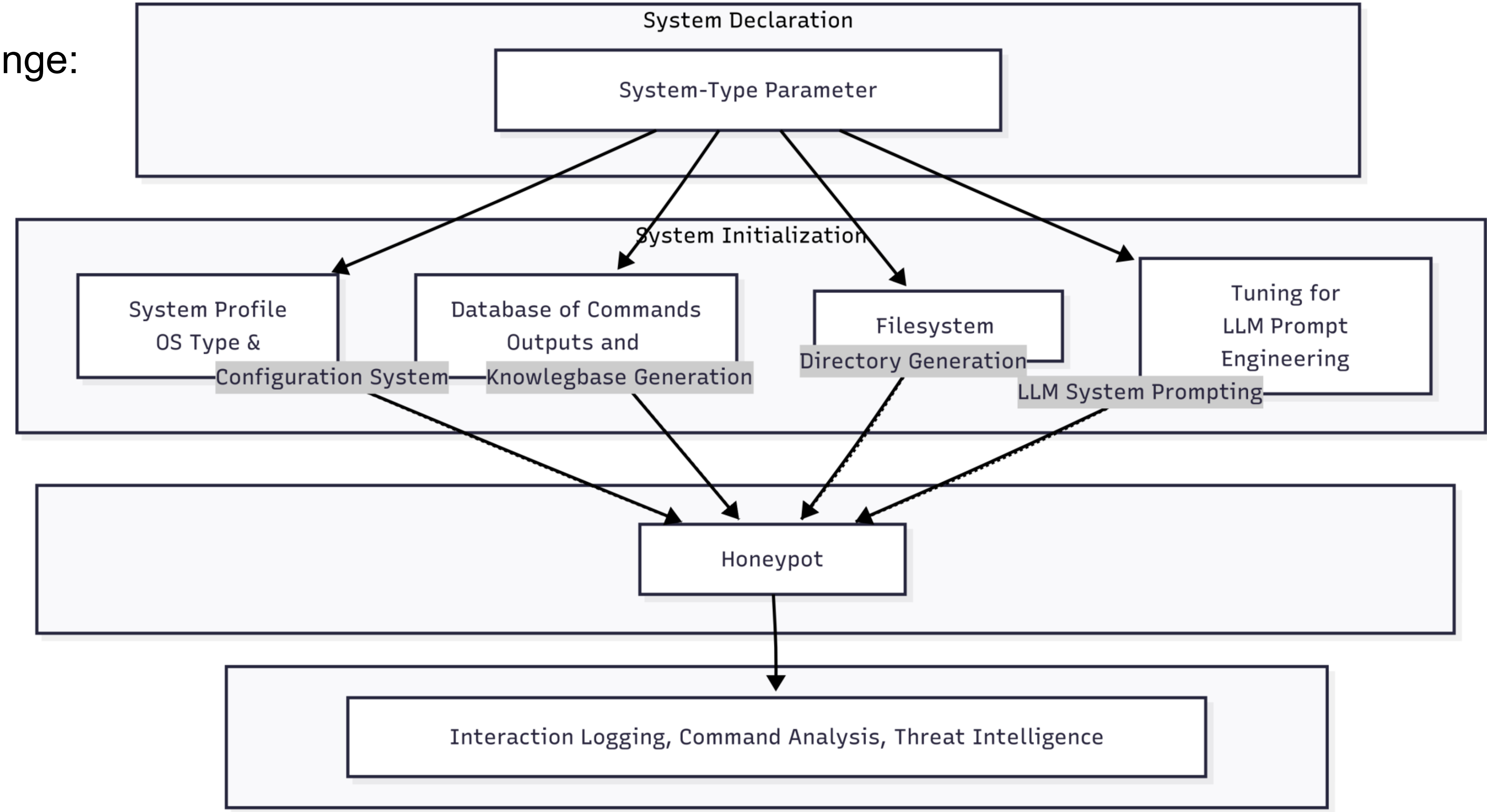
  ⇒Model data gets augmented by retrieved entry that is relevant for the prompt.

- Augment the prompt to use the retrieved data as extra context:
  “[…] Don't rely wholly on {context_str} it is just meant to enhance you, use your own powerful understanding. […]”

System-Based Attention Shell Honeypot (SBASH) framework

Framework to exchange:

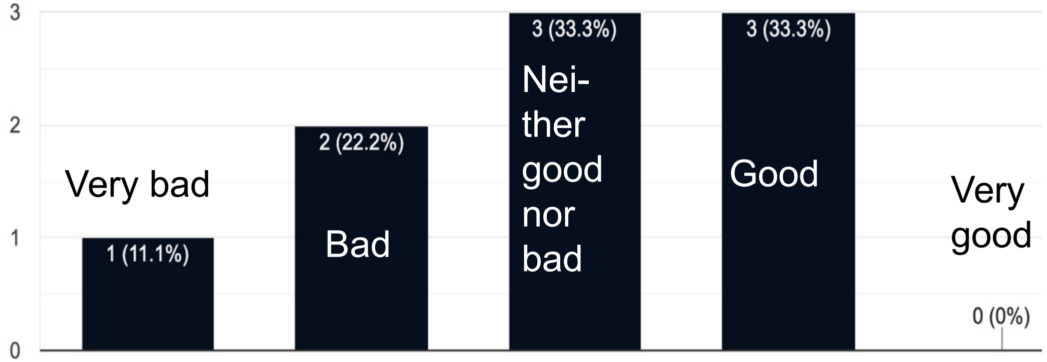
- LLM,
- RAG knowledgebase,
- Sandboxed filesystem pre-filled contents.
  - (Used for natively executed commands).



SBASH framework available at: https://github.com/Cyboghostginx/HoneywareX

Evaluation

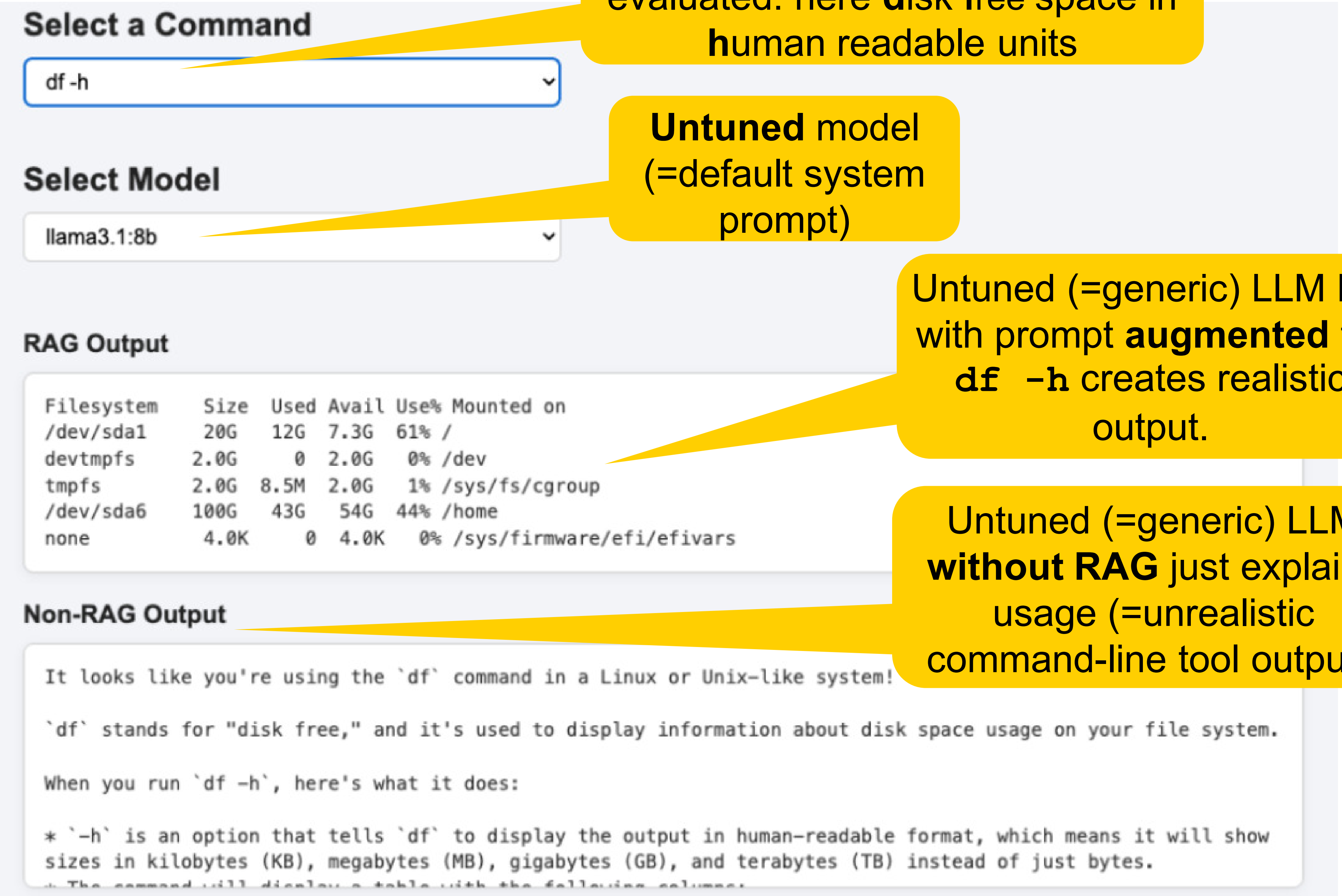
- LLMs used:
  - Gemma 3 (4B): 4 billion parameter open model from Google,
  - Gemma 3 (12B): 12 billion parameter open model from Google,
  - Llama 3.1 (8B): 8 billion parameter open model from Meta.
- Each of these LLMs evaluated using:
  - Plain (no tuned system prompt, no RAG),
  - RAG only (no tuned system prompt),
  - Tuned system prompt-only (no RAG),
  - Tuned system prompt + RAG.
- Accuracy in comparison to output of real command using, e.g., Levenshtein distance:
  - Edit distance (# characters to be deleted/inserted/substituted) normalised by string length.
- Speed/Latency,
- Realism perceived by humans (n=9):



Results: Output Example

- Automatic evaluation for a list of 81 commands:

```
arch
arp -a
atq
cat /etc/os-release
cat /etc/passwd
cat /etc/resolv.conf
cat /etc/shadow
cat /var/log/syslog
cat ~/.bash_history
crontab -l
curl ifconfig.me
date
df -h
dmesg
…
```



Select a Command

df -h

Select Model

llama3.1:8b

RAG Output

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1        20G   12G  7.3G  61% /
devtmpfs        2.0G     0  2.0G   0% /dev
tmpfs           2.0G  8.5M  2.0G   1% /sys/fs/cgroup
/dev/sda6       100G   43G   54G  44% /home
none             4.0K     0  4.0K   0% /sys/firmware/efi/efivars
```

Non-RAG Output

```
It looks like you're using the `df` command in a Linux or Unix-like system!

`df` stands for "disk free," and it's used to display information about disk space usage on your file system.

When you run `df -h`, here's what it does:

* `-h` is an option that tells `df` to display the output in human-readable format, which means it will show
sizes in kilobytes (KB), megabytes (MB), gigabytes (GB), and terabytes (TB) instead of just bytes.
```

Command-line command to be evaluated: here **d**isk **f**ree space in **h**uman readable units

Untuned model (=default system prompt)

Untuned (=generic) LLM but with prompt **augmented** for **`df -h`** creates realistic output.

Untuned (=generic) LLM **without RAG** just explains usage (=unrealistic command-line tool output).

Results: Accuracy

- Calculated average accuracy from list of 81 command-line commands executed,
- 100% accuracy=Identical output, 0%=all different output.
  - But: White space differences matter, for commands like `df`, it is OK to differ, `date` was wrongly in cmd. list.
- LLM temperature:=0.1 (=rather deterministic output, less creative).

Metric	Model	Plain	RAG only	RAG + Tuned system prompt	Tuned system prompt (no RAG)
Levenshtein	**Gemma 4B**	1.7%	10.5%	16.7%	**17.5%**
Levenshtein	**Llama 3.1 8B**	2.9%	19.0%	**21.3%**	20.0%
Levenshtein	**Gemma 12B**	1.4%	20.7%	21.9%	**24.6%**

For RAG/prompt-tuned: Larger models are more accurate.

Default system prompt: low accuracy (just explaining command).

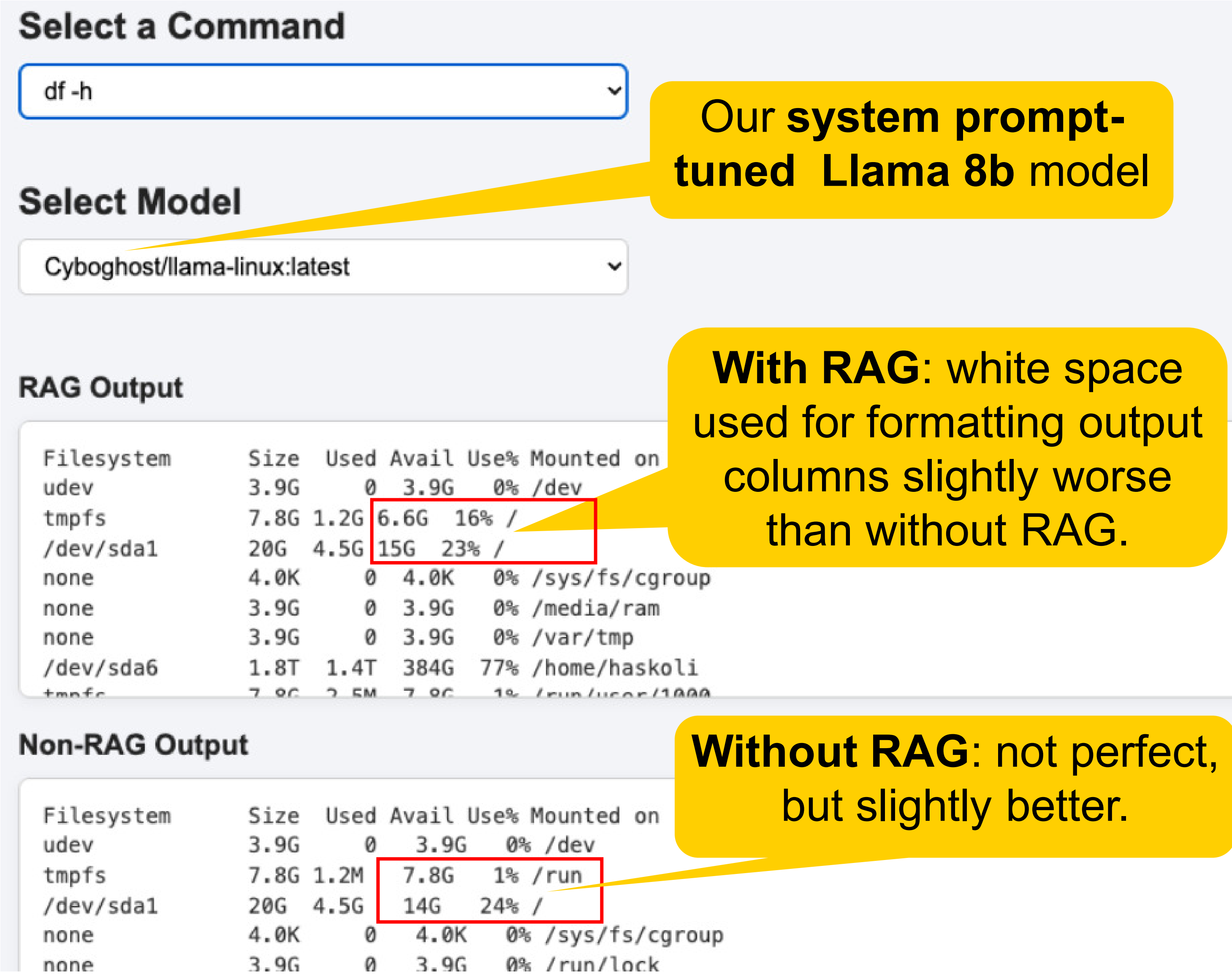
Augmenting default prompt helps.

Would expect RAG + tuned system prompt to be most accurate, but only the case for 8B model.

Surprisingly, tuned system prompt *without* RAG most accurate for the 4B and 12B Gemma models

Results: Output
Example white space differences

- Showcasing where
    - tuned system prompt **without RAG** (bottom: “Non-RAG Output”) performs **better**
    - than tuned system prompt with RAG (“RAG Output”).
    - No good explanation why that is the case.
- Accuracy in comparison with the real system:
    - It does not really matter whether the LLM shows for **`/dev/sda`** 15G or 14G available space and the real system has, e.g., 32G.
        - But Levenshtein would count this!



RAG Output

```
Filesystem      Size  Used Avail Use% Mounted on
udev            3.9G     0  3.9G   0% /dev
tmpfs           7.8G 1.2G 6.6G  16% /
/dev/sda1       20G  4.5G 15G  23% /
none            4.0K     0  4.0K   0% /sys/fs/cgroup
none            3.9G     0  3.9G   0% /media/ram
none            3.9G     0  3.9G   0% /var/tmp
/dev/sda6       1.8T  1.4T  384G  77% /home/haskoli
tmpfs           7.8G  2.5M  7.8G   1% /run/user/1000
```

Non-RAG Output

```
Filesystem      Size  Used Avail Use% Mounted on
udev            3.9G     0   3.9G   0% /dev
tmpfs           7.8G 1.2M   7.8G   1% /run
/dev/sda1       20G  4.5G   14G   24% /
none            4.0K     0   4.0K   0% /sys/fs/cgroup
none            3.9G     0   3.9G   0% /run/lock
```

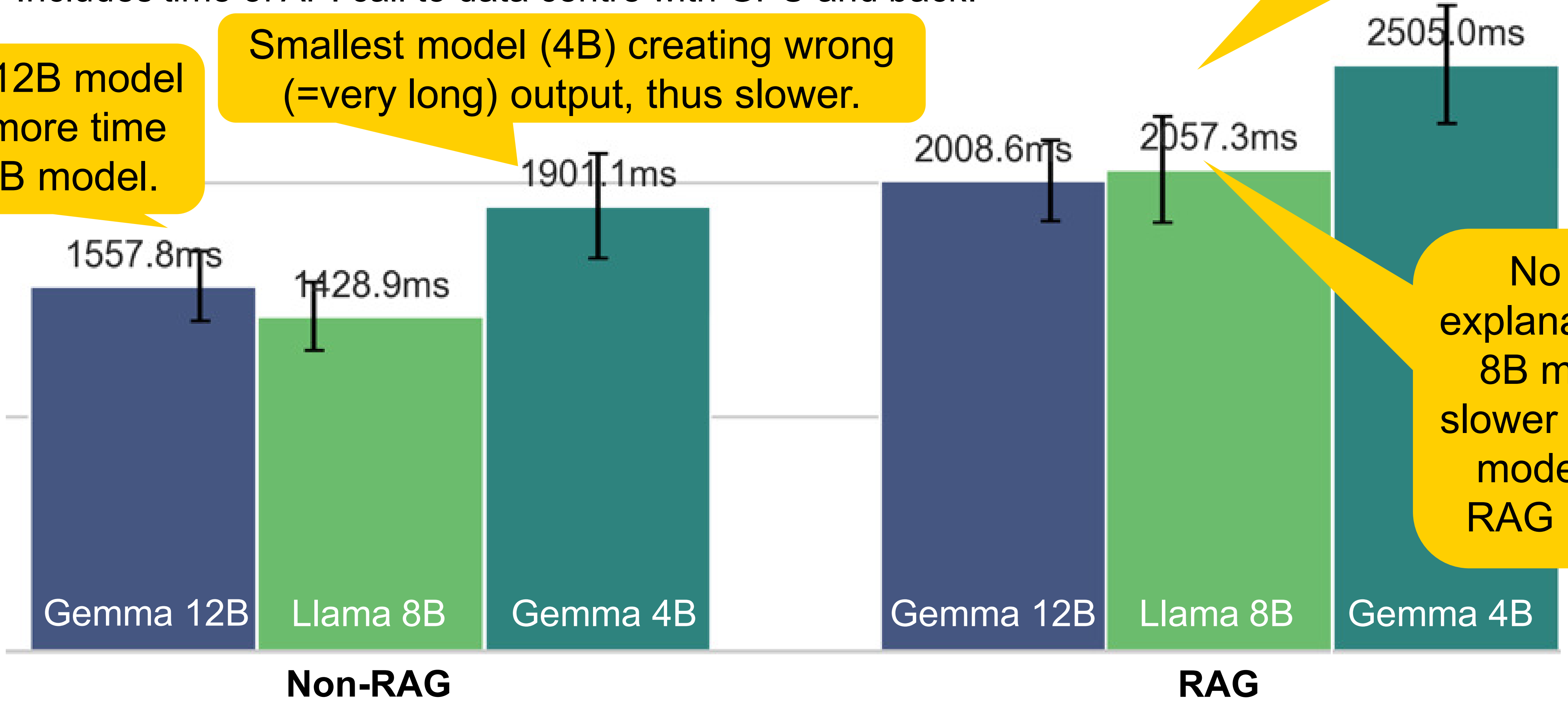
Our **system prompt-tuned Llama 8b** model

With RAG: white space used for formatting output columns slightly worse than without RAG.

Without RAG: not perfect, but slightly better.

Results: Latency

- Calculated average speed from 81 command-line commands.
- Executed on GPU NVIDIA H100 (=rather high-end data centre GPU).
  - Still 1.5 to 2 seconds per command is too slow to be realistic.
  - Includes time of API call to data centre with GPU and back.



Retrieval from RAG knowledgebase adds ca. 500 ms.

Larger 12B model takes more time than 8B model.

Smallest model (4B) creating wrong (=very long) output, thus slower.

No good explanation why 8B model is slower than 12B model when RAG is used.

Related work

- Cowrie: not LLM-based, but sandboxed, fake filesystem approach with custom command implementation used in our SBASH framework. https://github.com/cowrie/cowrie

- HoneyLLM: using massive LLMs (GPT-4o, Claude-3 Opus): 20%-88% accuracy:
  - Accuracy measure not defined in the paper, no latency measurements.
  - Cannot run locally, i.e. privacy violated (zero-day exploits might escape into LLM training data).

Guan, Cao, Zhu, “HoneyLLM: Enabling shell honeypots with large language models,”, Conf. Commun. Netw. Secur. (CNS). IEEE, 2024.

- LLM in the shell: chain-of-thought (=prescribe steps to be taken) and few-shot (provide examples – however static, in contrast to RAG) prompting techniques used for GPT-3.5-turbo-16k (high costs mentioned): Sladić, Valeros, Catania, Garcia, “LLM in the shell: Generative honeypots,” Europ. Symp. Secur. Priv. Workshops, IEEE, 2024.
  - Evaluated by humans with real and LLM output: 0.92 accuracy calculated based on false human classification of real vs. LLM. Latency mentioned as limitation.

- Limbosh: Honeypot based on pluggable LLM: Johnson, Hassing, Pijpker, Loves, “A modular generative honeypot shell,” Int. Conf. Cyber Secur. Resil. (CSR). IEEE, 2024.
  - Evaluated massive (GPT-4o) to lightweight (Tinyllama 1.1B) LLMs by humans (=no accuracies).
  - Main complaint by human evaluators: the machines was not fast.

Summary and Outlook

In practise, the honeypot does not feel that bad: e.g. even if the LLM does not produce the same output as the real system it is sufficient to looks realistic and to be consistent.

- System-Based Attention Shell Honeypot (SBASH) framework to evaluate using LLMs as honeypots.
- Results:
  - Edit-distance based accuracy rather low: max 24.6%.
  - Larger lightweight LLM more accurate.
  - For 2 out of 3 models/model sizes: system prompt tuning + RAG lead to worse accuracy than just using system prompt tuning.
  - No huge speed difference between different model sizes:
    - 4B model created too long (=both wrong and slow) output.
    - 1.5 to 2 seconds per command is too slow to be a realistic honeypot.
- Outlook:
  - Try LLMs trained for *coding*, Re-do evaluation with better list of commands,
  - Compare influence of RAG contents (man page vs. command input/output only).

Thank you for your attention · any questions?

UNIVERSITY OF ICELAND

Preprint DOI

Adetayo Adebimpe · Thomas Welsh ·
Helmut Neukirchen · helmut@hi.is · https://uni.hi.is/helmut

defend Iceland

EYVÖR | NCC-IS
ICELAND CYBERSECURITY
COORDINATION CENTRE

This project has received co-funding from the European Union's
Digital Europe Programme under grant agreement no. 101127453
National Coordination Centre for Cybersecurity in Iceland and
101127307 Defend Iceland: Nationwide bug bounty platform.

Co-funded by
the European Union

ECCC
EUROPEAN CYBERSECURITY
COMPETENCE CENTRE