**CSE 132 Final Project Report**

Modal Notebook (Public):

https://modal.com/notebooks/misen/main/nb-jX5qBd6ue2VRzbds6bfqz7

Hugging Face Dataset (Public):

https://huggingface.co/datasets/miisen/cse132-attack-commands

The system is built around LLM Qwen-2.5-7B-Instruct, chosen because of its strong instruction following, reasonable inference cost, and ability to run efficiently on modern GPUs such as the NVIDIA L40S. Generation was performed entirely through inference; no fine-tuning was required.

Hyperparameters weren't anything particularly intricate. To balance diversity and correctness the following sufficed: moderate temperature, bounded token limits, and constrained prompts. Generation ran for several hours in total due to the multi stage pipeline and targeted repairs.

Pipeline Design

1. Structured Command Generation

Each pipeline iteration begins with a tightly constrained prompt that instructs the LLM to output exactly three fields:technique_id, technique_name, command. No explanations or extra text were permitted. This strict structure allows reliable extraction using regular expressions. If extraction fails, the generation is retried. This step ensures that all base commands are clearly tied to a MITRE ATT&CK technique.

2. Rewriting for Diversity

After extracting the base command, the pipeline applies two successive rewrite stages: Rewrite 1 introduces mild variation, such as changing flags, syntax style, or command order.

Rewrite 2 pushes diversity further by switching tools, chaining commands, changing shells, or altering execution context. These rewrites deliberately incorporate strategies discussed during the TA-led class session: Using multiple languages and shells. Forcing the model away from default internet-style examples Chaining commands with operators like &&, |, and ;. Rewriting a rewrite to ensure compounding divergence. This layered rewriting approach proved critical for avoiding repetitive or overly common command patterns.

3. Tool Rotation and Mutation

To further diversify outputs, the pipeline rotates between functionally equivalent tools when appropriate. Examples include: wmic -> PowerShell WMI cmdlets, powershell -> pwsh -> cmd /c, Native discovery commands -> chained alternatives. Lightweight mutations, such as environment variable usage, conditional execution, and benign concatenation, are applied to break template-like similarity without altering behavior.

4. Validation Using Two LLMs

Following the TA's guidance, the pipeline uses two distinct LLM roles: Generator LLM will produce and rewrite commands. Validator LLM will evaluate semantic correctness. Syntactic validation checks whether commands are structurally valid and executable. Semantic validation asks a second LLM to answer whether the command actually implements the specified MITRE technique. Commands that fail semantic validation are selectively repaired and revalidated(this didn't work completely for some reason). Only failing rows are rewritten, which keeps the pipeline efficient.

5. Duplicate Detection and Final Diversity Pass

After generation, I checked exact duplicate commands in the final dataset. Rather than regenerating all 500 rows, only duplicates were mutated using transformations such as wrapping

or shell indirection. This ensures that every row contains a unique command string while preserving correctness.

This targeted approach significantly reduced redundancy and improved diversity metrics without excessive runtime.

6. Checkpointing

Because generation takes time and leads to occasional failures, the pipeline checkpoints progress to disk at regular intervals. This allowed safe recovery and incremental improvement without restarting the full process.

Evaluation Results

To test whether I might be passing the grading pipeline, I implemented some checks.

Syntactic Correctness: Approximately 91% of commands passed syntax checks. Failures were rare and typically caused by truncated outputs during rewriting.

Semantic Correctness: Initial accuracy was around 70%, but after semantic repair, accuracy improved into the mid-80% range.

Diversity: Using neighbor similarity, the final dataset achieved an average similarity score of approximately 0.61, indicating meaningful diversity relative to common public commands. However I made some mistake that I am not even sure of and lost some diversity, but I recovered by including new techniques.

Example Commands and Quality Commentary

1. **pwsh -NoExit -sta -NonInteractive -File "C:\Windows\System32\malware.ps1"**

Demonstrates realistic PowerShell execution with altered execution context.

2. **wmic path win32_process create "calc.exe"**

A concise example of WMI execution.

3. **wmic process where name="explorer.exe" get Caption,CommandLine,ParentProcessId /format:list**

Illustrates detailed system discovery using WMI.

4. **net user /domain:domain.com /user:username /advinfo**

Represents account discovery with extended flags.

5. **nmap -p 22 -iL targets.txt --open --reason --traceroute**

A realistic network discovery command with non-default scanning options.

Challenges and Solutions

1. Unclear starting point: Solved by directly applying TA guidance discussed in class.
2. Formatting: Solved using stricter prompts and regex extraction. Semantic drift during rewrites: Fixed with second model validation
3. Excessive similarity: Solved through rewriting, tool rotation, and duplicate mutation.
4. Runtime constraints: Solved through checkpointing.
5. Uncertainty: Unsure if final result is good enough.

Lessons Learned

This project shows the importance of treating LLMs as probabilistic components that require structure, validation, and layered safeguards. The most effective strategy was combining multiple small, well defined steps rather than relying on a single generation prompt which was almost useless. Using two LLMs, chaining rewrites, and explicitly engineering diversity were all essential to meeting the assignment goals. Overall, the project demonstrates that AI assisted security tooling can produce results with some error margin. I still don't know if my work meets the grading criteria but I am hoping I was able to pass some of the requirements with the pipeline I attempted to implement.