Evaluating LLMs for CPE Identification in IoT Reconnaissance

A Thesis

Presented To

Eastern Washington University

Cheney, Washington

In Partial Fulfillment of the Requirements

for the Degree

Master of Science in Cyber Defense

By

Christopher Michael Davisson

Spring 2026

THESIS OF Christopher Michael Davisson APPROVED BY

_____ DATE: _____

Dr. Sanmeet Kaur, Major Professor

_____ DATE: _____

Dr. Stuart Steiner, Committee Member

_____ DATE: _____

Lynnae Daniels, Committee Member

Abstract— Vulnerability identification during penetration testing relies on rigid string-matching to map network scan data to Common Platform Enumeration (CPE) identifiers and downstream Common Vulnerabilities and Exposures (CVEs). The approach frequently fails on physical Internet of Things (IoT) devices, which produce non-standard, irregular service banners that resist deterministic parsing. Large Language Models can reason through these fuzzy associations, but cloud-hosted models introduce cost, latency, and operational security concerns when processing reconnaissance data from live networks. This thesis asks whether locally-hosted open-weight Large Language Models (LLMs) can perform this task well enough to be useful, and how performance varies with model scale, family, and prompt design. I evaluate 36 LLMs across three deployment tiers (frontier hosted APIs, Ollama Cloud, and local vLLM), plus Nmap's native CPE extraction as the no-LLM baseline, on a curated corpus of Nmap covering 16 analyzed IoT device entries drawn from an 18-entry testbed: 14 physical entries and 4 derived entries, with one physical and one derived entry excluded for unusable scan or run data. Three findings shape the result. Model family is the dominant source of variance, with a wider spread than prompt design or scan-suite choice produces. The best truly-local model approaches frontier accuracy on a multi-GPU workstation but does not on single-GPU consumer hardware. The most operationally dangerous failure is not a malformed string but a confident, well-formed CPE that retrieves the wrong vulnerability list.

Acknowledgments

I would like to thank my major professor, Dr. Sanmeet Kaur, for her invaluable guidance, insight, and patience throughout this project. Her steady perspective and thoughtful counsel shaped this work at every stage, and I am grateful for the calm she brought to a process that felt daunting at times.

In addition, I would like to thank my committee members, Dr. Stuart Steiner & Lynnae Daniels for their acceptance and guidance in the process of completing this degree.

I would also like to thank the faculty and staff of Eastern Washington University for all they have done for me and for all other students. Their commitment to student success and growth provided the foundation necessary to carry out this research.

Finally, I am most grateful to my family and loving wife for all their support and understanding during the course of this research. I wouldn't be here without them by my side every day.

Statement on AI Use

This thesis investigates large language models as tools for penetration-testing reconnaissance. Large language models were also tools in its production.

Claude (Anthropic) and Gemini (Google) were used throughout the writing process for copy editing, and editorial feedback across most chapters. Claude Code was used during pipeline development as a coding assistant and for diagnosing failures during long inference runs.

AI did not generate the experimental design, model selection, dataset curation, ground-truth labeling, scoring rubric, or interpretation of results. All claims, quantitative analysis, and conclusions are the author's. AI-assisted prose was reviewed and edited before inclusion; AI-generated code was tested and modified before commit. The author retains full responsibility for the content of this thesis.

Table of Contents

List of Figures

List of Tables

List of Acronyms

API	Application Programming Interface. vii, 1, 14–17, 23, 24, 26, 36, 43, 44
CMMC	Cybersecurity Maturity Model Certification. 15
CNA	Numbering Authority. 9
CPE	Common Platform Enumeration. iii, 1, 2, 4–7, 9–11, 14, 17–21, 24, 33, 35, 37, 38, 40, 42, 44, 46, 47
CVE	Common Vulnerabilities and Exposure. iii, 2, 5–7, 9–11, 17, 18, 21, 44, 46
FedRAMP	Federal Risk and Authorization Management Program. 15
GPU	Graphics Processing Unit. 1, 23, 37, 47
HIPAA	Health Insurance Portability and Accountability Act. 15
HTTP	Hypertext Transfer Protocol. 9, 12
HTTPS	Hypertext Transfer Protocol Secure. 12
ICMP	Internet Control Message Protocol. 9
IoT	Internet of Things. iii, 2, 4–8, 12, 17, 20, 22, 44
IP	Internet Protocol. 12
JSON	JavaScript Object Notation. 14
LLM	Large Language Model. iii, 1, 2, 4, 5, 7, 9, 13, 14, 16, 17, 19, 23, 24, 26, 44, 46, 47
MAC	Media Access Control. 2, 19, 22
NIST	National Institute of Standards and Technology. 1, 9
NLP	Natural Language Processing. 5
NRDP	Netflix Ready Device Platform. 2
NVD	National Vulnerability Database. 2, 6, 9–11, 17, 21, 22, 42, 44
OS	Operating System. 2, 12
OUI	Organizationally Unique Identifier. 2, 5
PRNG	Pseudo-Random Number Generator. 13, 29
Regex	Regular Expression. 14
RTOS	Real-Time Operating System. 12
RTSP	Real Time Streaming Protocol. 2
SSH	Secure Shell. 9

TCP Transmission Control Protocol. 12, 35, 36

TLS Transport Layer Security. 9, 12

UDP User Datagram Protocol. 36

VRAM Video Random Access Memory. 1, 7, 14, 47

Glossary of Terms

constrained decoding A class of inference techniques that restrict an LLM's token generation to outputs matching a predefined grammar, regular expression, or JSON schema. Tokens that would violate the constraint are rejected at generation time, eliminating syntactically invalid output. 14, 26, 42

data sovereignty The principle that sensitive data remains under the physical and legal control of its owner. In the context of LLM deployment, data sovereignty motivates local inference over cloud-hosted APIs when reconnaissance data, source code, or other operationally sensitive content would otherwise leave the operator's infrastructure. 14, 16

deep learning A field of machine learning that uses many layered neural networks to learn hierarchical representations from raw data. 5

fingerprinting The process of inferring the identity, vendor, model, or version of a device or service from observable network behavior, such as service banners, response timing, or protocol quirks. 12

frontier model A highly capable, large-scale foundation model developed by a well-resourced AI lab and typically accessible only via API. In this thesis the term refers specifically to the closed models operated by Google, Anthropic, and OpenAI. 1, 14, 23, 26, 43, 44

greedy decoding A decoding strategy in which the model selects the single most probable token at every generation step, bypassing probabilistic sampling. Achieved in practice by setting the sampling temperature to zero, where supported. 13, 28

hallucination Output from a large language model that is plausibly formatted and confidently presented but factually incorrect. Hallucinations are distinguished from structurally malformed output by the absence of any signal that the result is wrong, which makes them harder to detect and more dangerous in operational use. 6, 11, 38

inference In the context of large language models, the process of generating output by repeatedly predicting the next token conditioned on the preceding context. Distinct from training, which adjusts the model's weights. 9, 13, 14

mean match The per-prediction score assigned by the tiered rubric: 1.0 for an exact match, 0.5 for a partial match, and 0 for no match or no CPE. The mean match score for a model is the arithmetic mean of this value across all the model's scored predictions in

the dataset . 30

partial correctness A property of a CPE identifier in which some fields match the ground-truth value and others do not. Partial correctness is operationally meaningful when the correct fields appear higher in the CPE hierarchy (part, vendor, product) than the incorrect ones, because a CVE lookup against the partial CPE returns a superset of the relevant vulnerabilities rather than an entirely different set. 11

penetration testing An authorized security assessment in which an operator probes a target network or system for exploitable weaknesses, typically following a structured methodology covering discovery, attack, and reporting. 4, 47

quantization A compression technique that reduces the numerical precision of a model's weights (for example, from 16-bit floating point to 8-bit or 4-bit integers) to reduce VRAM requirements and improve inference speed on local hardware, at the cost of small accuracy degradation that compounds with more aggressive precision reduction. 1, 47

reconnaissance The discovery phase of a penetration test, in which the operator enumerates hosts, services, and software versions. Passive reconnaissance gathers information without directly engaging the target (open-source intelligence, DNS enumeration, traffic observation). Active reconnaissance interacts with the target directly (port scanning, service probing, OS detection). 10, 47

service banner A text-based response returned by a network port upon initial connection, typically used during reconnaissance to identify the underlying software application and its specific version. iii, 1, 5, 11, 12

tiered rubric A scoring scheme that assigns graded credit rather than binary correct-or-incorrect outcomes. In the context of this thesis, tiered scoring reflects the operational consequence of each failure mode in CPE generation rather than treating all deviations from ground truth as equivalent failures. 11

token A discrete unit of text produced by a tokenizer, which may correspond to a whole word, a subword, punctuation, or whitespace. Large language models operate on sequences of tokens rather than raw characters. 13

I. Introduction

Penetration testing is an exercise in which an authorized operator probes a target network or system for exploitable weaknesses. The stages of this process, as defined by National Institute of Standards and Technology (NIST), are Planning, Discovery, Attack, and Reporting [1]. The discovery phase is when the operator enumerates hosts, services, and software versions to determine what is actually present on the network and the attack surface they represent. Reconnaissance has historically been a careful, manual discipline, with tooling like Nmap, Masscan, and Shodan maturing around the assumption of a human operator interpreting the output. That assumption is changing as LLM-driven frameworks automate progressively larger portions of the workflow. Machine learning has been a useful tool for decades; frameworks based on LLMs such as PentestGPT [2] and AutoPT [3] have emerged in rapid succession, with new systems appearing before the security community has had enough time to evaluate their operational safety and efficacy. Many of them route reconnaissance data, including network topology, service banners, software versions, and inferred vulnerabilities through cloud-hosted frontier models owned and operated by third parties. The use of these larger, more capable models requires a level of trust that the model provider does not retain the data, does not record it in a discoverable form, and subsequently does not leak it [4].

An organization whose security model depends on maintaining the confidentiality of its internal infrastructure incurs additional risk when transmitting complete reconnaissance datasets to third-party servers [4–6]. An alternative approach is to host the model locally, eliminating external data exposure at the cost of access to proprietary frontier models available only through APIs. Locally hosted models face a hard ceiling, constrained by the available Video Random Access Memory (VRAM). A professional workstation equipped with a high-end Graphics Processing Unit (GPU) such as an Nvidia RTX 4090 (24 GB VRAM) can reliably host models in the 7B-34B parameter range at moderate quantization, or models up to 34B with aggressive 4-bit quantization. More aggressive quantization, below 4-bit, degrades reasoning capability and accuracy [7, 8], introducing complications that most security analysts are not prepared to manage in the field. While these benchmarks focus on mathematical reasoning, the underlying mechanisms are the same for structured output tasks such as CPE identification. The operational question this thesis addresses is not if a local model is necessary, but rather, if a model fitting within such constraints can handle

real-world reconnaissance tasks enough to be useful and reliable.

This research explores the efficacy of small locally hosted LLMs on a narrow reconnaissance sub-task: fingerprinting IoT devices [9, 10] from imperfect and incomplete network scans [11] in order to produce valid CPE identifiers [12] suitable for downstream CVE association through National Vulnerability Database (NVD)-style CPE lookup workflows [12–14]. The shape of the results, especially in comparison with larger frontier and cloud-based models, is the central object of study.

In this study, CPE identification is evaluated as a graded outcome rather than a binary one: an output can be exactly correct, partially correct in a way that still supports CVE lookup, structurally invalid, or plausibly formatted but factually wrong. Each category carries a different operational consequence. Structurally malformed output can often be detected and normalized, whereas confident but factually incorrect outputs are harder to detect and correct without external validation. The research question is therefore less how often these models succeed in aggregate than how, when, and why they fail, and if those failure modes vary with model scale, or model family.

A. Motivating Example

The concept for this research began from a normal port scan of my local home network. A Toshiba Fire TV Edition Smart TV was misidentified as a phone running Android 5.1. While the Operating System (OS) family is approximately correct, Fire OS is Android-derived, the device class was incorrect. The Media Access Control (MAC) address Organizationally Unique Identifier (OUI) mapped to a contract manufacturer associated with Amazon hardware rather than a phone manufacturer. The exposed services are atypical for a phone. Port 7000 exposed an AirPlay-style Real Time Streaming Protocol (RTSP) service, port 8009 behaved like a Google Cast receiver endpoint, and port 9080 returned a Netflix Ready Device Platform (NRDP) server header generally associated with Netflix Ready Device Platform runtimes on streaming hardware. Individually, these are just errant signals, however when taken together they point to a smart TV. A binary rubric would simply mark the answer as wrong, a tiered rubric distinguishes between a partially correct OS family and an incorrect device class, missing vendor, incorrect version. Each difference has a different operational consequence. Initial testing revealed that feeding various levels of scan data into chat interfaces produced disparate results, largely dependent on the reasoning capabilities and search access available to each model. Around the same time, I encountered the emerging literature on LLM-augmented penetration testing [2], which made the question concrete: could a model with limited context and no internet access still

produce useful identifications from this kind of signal?

B. Contribution

This thesis makes three contributions. First, it isolates CPE generation from Nmap reconnaissance output as a measurable reconnaissance subtask, rather than evaluating it only as one step inside a larger penetration-testing agent. Second, it compares frontier hosted models, cloud-hosted open models, locally served open-weight models, and Nmap-native CPE extraction on the same physical and derived IoT scan corpus. Third, it evaluates model output with a tiered scoring rubric that distinguishes exact matches, operationally useful partial matches, broader related identifiers, malformed output, and well-formed but factually incorrect CPE hallucinations.

C. Research Questions

1. **Accuracy and its drivers.** How does CPE-identification accuracy on Nmap reconnaissance output vary across deployment tier, model family, model scale, prompt design, and scan-suite choice, and which of these produces the widest spread?

2. **Local versus hosted tradeoff.** How much CPE-identification accuracy is lost when moving from hosted frontier models to locally hosted open-weight models, and do the strongest local models clear the Nmap-native baseline and approach the weakest frontier model?

3. **Failure modes and hallucination risk.** How often do models produce well-formed but factually incorrect CPE identifiers, and how does this hallucination risk vary by model family, model size, and deployment tier?

II. Literature Review

This work takes inspiration from several related research areas: LLM-assisted penetration testing, automated CPE identification, IoT device fingerprinting, structured output reliability, and local LLM deployment. Each area addresses part of the problem studied here, but none directly evaluates whether locally hosted open-weight LLMs can infer operationally useful CPE identifiers from ambiguous Nmap output for physical IoT devices. This chapter reviews the adjacent work and identifies this gap.

A. LLM-Based Penetration Testing

LLM-driven penetration testing has grown around agentic systems that automate parts of the penetration-testing workflow. PentestGPT, AutoPT, PentestAgent, and xOffense all run LLMs inside multi-step reasoning loops: the model plans an action, reads the tool output, and writes the next step or a finding [2,3,15,16]. AutoPenBench and similar benchmarks have started to formalize how these systems are evaluated across structured penetration-testing tasks [17].

This body of work establishes that LLMs can assist with security tasks, even where an operator generally has to interpret noisy tool output and decide what to do next. A primary limitation of these systems is that they usually evaluate an end-to-end agent or a broad penetration-testing task suite. Reconnaissance appears as one component inside a larger workflow, not as the object of measurement, and that distinction matters because end-to-end benchmarks obscure where a system actually succeeds or fails. A model can fail because it picked the wrong tool, parsed output incorrectly, hallucinated a vulnerability, or produced an incorrect product identifier. This thesis isolates one narrow reconnaissance subtask: generating CPE identifiers from Nmap scan evidence. The narrower scope lets the experiment measure product identification on its own, before any downstream exploitation or reporting logic is layered on top.

Prior work also tends to assume access to strong hosted models. That assumption is fine for benchmarks, but it becomes uncomfortable in penetration testing, where reconnaissance output can describe live hosts, software versions, exposed services, and inferred vulnerabilities. Sending that data to a third-party model provider extends the trust boundary of the engagement. Local inference is therefore part of the research problem in this thesis, not a deployment detail.

B. CPE Identification and Vulnerability Mapping

CPE identifiers are the bridge between observed products and downstream CVE lookup. Automated vulnerability management depends on this bridge: if the product identifier is correct, the vulnerability search space is bounded; if the identifier is wrong, the resulting CVE list may be irrelevant or misleading [12]. This makes CPE identification an operationally meaningful target for measurement.

Existing automated CPE identification work has mostly approached the problem from the Natural Language Processing (NLP) side. CPE-Identifier treats CPE extraction as a labeling problem over vulnerability descriptions, using deep learning and NLP methods to identify product entities in CVE summaries [14]. Wåreus and Hell similarly frame automated CPE labeling as a machine-learning task over CVE text and note that missing or incorrect CPE annotations degrade downstream vulnerability-scanning pipelines [18].

These systems are closely related but solve a different problem. Their inputs are vulnerability descriptions: human-authored text records about known vulnerabilities. This thesis uses raw reconnaissance output, which is machine-generated, incomplete, and often ambiguous. A service banner can expose a library version without naming the product that uses it. An open port pattern may suggest a device class but rarely a vendor. The OUI in a MAC address often points to the contract manufacturer rather than the brand printed on the device. The task is therefore inference from weak evidence, not extraction from clean text.

The output space is also different. Prior CPE labeling work often treats the task as classification or entity extraction over a known vocabulary. This thesis instead evaluates open-vocabulary generation of full CPE 2.3 strings. That design exposes a different class of failure. A model can produce a syntactically valid CPE that names the wrong vendor, product, or version. Such an output is more dangerous than malformed output. It passes validation and retrieves a plausible-looking vulnerability list for the wrong product.

C. IoT Device Fingerprinting

IoT device fingerprinting has a longer empirical history than LLM-based penetration testing. IoT Sentinel and related device-classification work show that network behavior alone is often enough to infer device type, vendor, or class [9, 10]. These studies establish a pattern that the present work builds on: IoT devices expose stable but indirect signals, including traffic patterns, protocol behavior, timing, and limited service surfaces.

The fingerprinting literature also explains why the task is difficult. Consumer IoT devices are heterogeneous and resource constrained, and many of them are built from reused embedded software stacks. Exposed services tend to be minimal. Banners come back truncated or generic. Protocol behavior often reveals an underlying component, such as a BusyBox build or a stripped HTTP server, rather than the brand on the device. These properties break rigid string matching. They are the right kind of input for semantic reasoning over imperfect evidence.

Most IoT fingerprinting work does not generate CPE identifiers. It usually classifies device type, vendor, or behavioral category from passive traffic features. This thesis instead uses active Nmap scan output and asks for structured product identifiers suitable for NVD lookup. The difference is operationally significant. A device-type label such as "camera" or "smart plug" helps an analyst understand the network, but it does not directly support CVE lookup. A CPE identifier does.

D. Structured Output, Hallucination, and Abstention

Because CPE identifiers have a strict syntax, this task also connects to the literature on structured output and constrained decoding. Grammar-guided or regex-constrained decoding can prevent a model from emitting malformed strings by restricting generation to outputs that match a predefined pattern. For this thesis, that makes constrained decoding a useful experimental condition: it can test whether enforcing the CPE 2.3 grammar improves accuracy or only changes the type of error.

The distinction between syntactic validity and factual correctness is central. A malformed CPE fails loudly: the downstream lookup cannot execute, and the analyst knows manual investigation is needed. A well-formed but factually incorrect CPE fails silently and retrieves a plausible CVE list for the wrong product. This is the security-relevant form of hallucination. The danger is plausibility. A wrong CPE that parses cleanly and returns a believable-looking vulnerability list can be acted on before anyone notices the mistake.

This also connects to abstention. Sometimes the safest model behavior is to refuse to invent a version, product, or vendor that the scan does not actually support. A model that emits a wildcard or partial identifier can be more useful in practice than one that guesses a specific but unsupported release. The scoring rubric in this thesis reflects that: exact matches get full credit, operationally useful partial identifiers get partial credit, and confident-but-wrong output is scored separately from malformed output, because the two failure modes have very different operational consequences.

Prompt design is relevant for the same reason. The doubled-prompt condition used

in this thesis repeats the instructions and scan payload inside a single user message. This design is motivated by the possibility that placement and redundancy affect how models use long or noisy contexts. Liu et al. [19] showed that models do not weight all parts of a long prompt equally, which makes prompt layout a variable worth testing in its own right.

E. Local Inference as an Operational Constraint

Local LLM inference changes the problem. Hosted frontier models reason better and follow instructions more reliably than their open-weight counterparts, and they support longer contexts. They also require sending the prompt to a third party. That trade-off is fine for ordinary text generation. It is harder to accept in penetration testing, where the prompt may contain a map of the target environment: live hosts, open ports, service banners, software versions, and inferred vulnerabilities. That data is sensitive even when it does not include exploit code or credentials.

The local-inference constraint creates a tension between capability and control. Smaller open-weight models can run on operator-controlled hardware, but they are constrained by available VRAM, quantization choices, and runtime support. Larger hosted models may perform better, but they extend the trust boundary and may introduce cost, latency, logging, retention, or compliance concerns. This thesis treats that trade-off as an empirical question rather than an assumed conclusion. It measures how much accuracy is lost when moving from frontier hosted models to locally hosted open-weight models, and whether the remaining performance is useful enough for reconnaissance support.

F. Position of This Thesis

The gap this thesis addresses sits at the intersection of the preceding areas. LLM penetration-testing benchmarks evaluate full agentic systems end-to-end, so CPE generation appears as one step inside a larger pipeline rather than as the measured outcome. The automated CPE work that does exist targets CVE descriptions, not raw scan output. IoT fingerprinting work classifies devices by type or vendor but stops short of generating CPE 2.3 strings. Constrained decoding can enforce CPE syntax, but a syntactically valid CPE can still name the wrong product. And local inference, while it solves the trust-boundary problem for reconnaissance data, restricts the experiment to models that fit on operator hardware.

This thesis contributes a controlled measurement study at that intersection. It evaluates whether locally hosted open-weight LLMs can infer useful CPE identifiers

from imperfect Nmap scans of physical IoT devices, compares those models against hosted frontier models and an Nmap-native baseline, and scores outputs using a tiered rubric that separates exact matches, operationally useful partial matches, malformed output, and confident-but-wrong hallucinations.

III. Background

This chapter establishes the prerequisite knowledge a reader needs to follow the methodology. Readers familiar with reconnaissance tooling, the NVD data model, and LLM inference can skim or skip.

A. Nmap and Network Reconnaissance

Nmap, short for network mapper, is an open-source network scanner originally written by Gordon Lyon. It performs host discovery by sending probe packets to target addresses and analyzing responses, then enumerates open ports on each live host and attempts to identify the service running on each port and the operating system of the host itself. Service identification is performed by sending application-layer probes (Hypertext Transfer Protocol (HTTP) requests, Transport Layer Security (TLS) handshakes, Secure Shell (SSH) version exchanges, and others) and matching the responses against a signature database. Operating system identification is performed by analyzing low-level TCP/IP stack behavior (TCP options ordering, initial sequence number generation, Internet Control Message Protocol (ICMP) response patterns) and matching that fingerprint against a database of known operating system implementations.

B. CPE & CVE

1) CVE

A CVE record is a publicly disclosed security flaw assigned a unique identifier of the form `CVE-YYYY-NNNNN`, where the year reflects when the identifier was reserved and the trailing digits distinguish entries within that year. Each record is maintained by a CVE Numbering Authority (CNA) and typically includes a description of the flaw, affected products expressed as CPE identifiers, and references to advisories or patches. The NVD enriches these records with severity scores, weakness classifications, and the structured CPE applicability data that downstream tooling uses for automated lookup. The practical workflow is therefore: a CPE identifies a product, and a query against the NVD returns the set of CVEs whose applicability data matches that CPE.

2) CPE

CPE is a structured naming convention maintained by NIST for identifying IT products. A CVE list produced from a CPE is not a vulnerability assessment by itself,

though it may include vulnerabilities that have been patched or are not exploitable in the deployed configuration. It does, however, bound the search space for human analysis. The current standard is CPE 2.3 formatted strings. The format takes eleven colon-separated fields after an initial `cpe:2.3:` prefix:

```
cpe:2.3:part:vendor:product:version:update:edition:
language:sw_edition:target_sw:target_hw:other
```

The fields are defined as follows:

Part Indicates whether the target is an application (`a`), operating system (`o`), or hardware (`h`).

Vendor The organization or entity responsible for the product's creation.

Product The specific name of the software or hardware platform.

Version The specific release or version identifier.

Update The patch or update level of the identified product.

Edition A legacy field, typically populated with a wildcard (`*`).

Language The locale or language code as defined by RFC 5646.

SW Edition Further refinement for specific software editions.

Target SW The software environment in which the product operates.

Target HW The underlying hardware architecture (for example, `x64`).

3) Why Partial Correctness Matters

CPE identifiers are designed as a hierarchy. Reading left to right, each field narrows the set of matching products: the part field separates hardware, software, and operating systems; the vendor field narrows the match to a manufacturer; the product field identifies a specific product line; and the version field identifies a specific release [20]. When a CPE is fully populated and correct, querying the NVD's CPE applicability data returns a CVE list precisely scoped to that device's deployed configuration [13]. The list is short, every entry is potentially relevant, and human analysis time is spent on triage rather than filtering.

That outcome is the exception, not the rule. Reconnaissance against real devices regularly yields the vendor and product but not a clean version, or a plausible vendor

with an ambiguous product, or a service banner that pins the software stack but says nothing useful about the underlying hardware. A scoring rubric that treats any deviation from the ground-truth CPE as a failure throws away the distinction between these cases, and that distinction is what determines whether the output is operationally useful [12].

Consider three failure modes against the same target device:

Correct vendor and product, wildcard version. Querying NVD with this CPE returns every CVE ever filed against the product, across all versions. The list is wider than necessary but still scoped to the right device. An analyst working from this list reads CVEs that pertain to their target, applies judgment about which versions are deployed, and reaches correct conclusions. The lookup did real work.

Correct vendor, wrong product. The returned CVE list is scoped to a different device. Every entry on it is irrelevant to the target, and the analyst has no signal that this is the case without independent verification. Analysis time spent on this list is worse than wasted: it produces confident, wrong conclusions about the target's exposure. This is the failure mode of hallucination applied to CPE generation.

Structurally malformed CPE. The string fails to parse and the lookup never executes. The analyst sees the error immediately and falls back to manual investigation. No false confidence is generated.

The first case is partial correctness that preserves operational value. The third is wrong, but visibly so, and cheap to correct. The second case, the one that looks like a near-miss on the page, is the most dangerous output the pipeline can produce. A binary correct-or-incorrect rubric collapses all three into a single failure bucket. It obscures the only distinction that matters.

The tiered rubric used in this thesis (defined in Chapter V) is built around this observation. Each CPE field is scored independently, with weights reflecting its position in the lookup hierarchy and its operational consequence on the resulting CVE list. A model that confidently invents a product name is penalized more heavily than one that emits a wildcard, and a model that emits structurally invalid output is treated as a separate failure category from one that emits well-formed but incorrect data. This mirrors how an experienced analyst would weight the same outputs in the field.

C. IoT Devices

IoT is a label of convenience covering a heterogeneous category of network-connected devices that are neither general-purpose computers nor traditional network infrastructure. The category includes smart-home appliances (thermostats, doorbells, locks, lights), consumer entertainment devices (smart TVs, streaming sticks, game consoles), industrial sensors and controllers, networked cameras, embedded development boards (ESP32, Raspberry Pi-class devices in fixed-function deployments), and a long list of single-purpose connected hardware [9, 10].

1) Difficulty Fingerprinting

The main features of IoT devices that make fingerprinting difficult are minimal exposed surface, truncated or non-standard banners, and non-standard network stacks. Each of these reduces the signal available to conventional scanning tools.

Minimal Exposed Surface An IoT device typically exposes only the services required for its function: often a single HTTP or Hypertext Transfer Protocol Secure (HTTPS) endpoint, sometimes a control protocol on a non-standard port, occasionally nothing scannable at all. A fingerprinting tool that relies on probing a wide variety of services has fewer signals to work with.

Truncated and Non-Standard Banners Embedded HTTP servers commonly omit version strings, return generic Server headers ("lighttpd" with no version, or no header at all), and run lightweight TLS libraries such as mbedTLS or wolfSSL in place of OpenSSL. Service-version detection that depends on banner content is unreliable when the service banner was never designed to convey identifying information [11].

Non-Standard Network Stacks IoT devices often run on stripped-down embedded operating systems (FreeRTOS, Zephyr, vendor-specific Real-Time Operating System (RTOS) variants) with lightweight Transmission Control Protocol (TCP)/Internet Protocol (IP) implementations such as lwIP or uIP rather than the full Linux, Windows, or BSD stacks that Nmap's OS-detection database is built around. The fingerprints these stacks produce do not match Nmap's signature database, and OS detection on them frequently fails or misidentifies.

IoT devices produce scan output that conventional fingerprinting tools struggle with most, and the device population is large enough that this is not an edge case worth ignoring. This thesis uses IoT identification as a test bed precisely because it is

the regime where deterministic tools have the least to work with, which is also where an LLM that can reason across weak and incomplete signals has the most marginal value to provide.

D. Large Language Models

An LLM, specifically an autoregressive language model, is a neural network trained to predict the next token conditioned on the preceding tokens. Tokens are the discrete text units produced by a tokenizer and may correspond to whole words, subwords, punctuation, or whitespace. Given a series of input tokens, called the context, the model outputs a probability distribution over the probable next tokens. During inference, a decoding algorithm selects the next token from this distribution, appends it to the context, and repeats to generate text.

1) Inference and Determinism

Unlike deterministic tools such as Nmap, LLM inference is inherently probabilistic. To predict the next token in a sequence, the model generates a probability distribution over its entire vocabulary. Temperature is a hyperparameter that scales this distribution before a token is selected. A higher temperature flattens the distribution, increasing the likelihood of selecting lower-probability tokens and introducing creative variance. Setting temperature to 0.0 triggers greedy decoding, in which the model bypasses probabilistic sampling entirely and selects the single most probable token at every step. This control is only available on models that expose temperature as a parameter.

Even with greedy decoding, hardware-level variables and backend inference engines may still introduce variance through Pseudo-Random Number Generators (PRNGs) used during batching or internal operations. To establish a rigorous baseline for reproducible research, this variance is further reduced where the inference engine permits by applying a fixed random seed. The seed acts as the initiation state for the PRNG, and keeping it constant ensures that random values required during generation remain identical across execution runs.

Both controls (temperature at 0.0 and a fixed seed) are used to isolate the model's reasoning behavior from sampling noise, so that differences observed in the results can be attributed to model architecture and prompt design rather than stochastic variance.

2) Local Versus Cloud Inference

The deployment of large language models falls into two buckets: commercial cloud APIs, and local inference. Frontier models accessed via API offer state-of-the-art reasoning capabilities but require sending input data to third-party infrastructure. Local inference executes the model entirely on host hardware, preserving data sovereignty at the cost of being bound by available VRAM and compute. The operational security implications of this choice are addressed in Chapter IV; this chapter establishes only the technical distinction.

3) Constrained Decoding

When an LLM generates text in its default configuration, it predicts tokens until it produces a stop token or reaches a length limit. This is appropriate for chat applications but unreliable when the output must be parsed by downstream tooling. A CPE string with a missing field delimiter, or a tool call with malformed JavaScript Object Notation (JSON), will fail to parse regardless of whether the content was otherwise correct.

Constrained decoding addresses this by restricting the model's token generation to outputs that match a predefined grammar, Regular Expression (Regex), or JSON schema. At each generation step, tokens that would violate the constraint are rejected before sampling, eliminating syntactically invalid output by construction. The tradeoff is that constraints can occasionally force the model into low-probability continuations that distort its reasoning behavior [21]. Whether this tradeoff helps or hurts CPE identification accuracy is one of the experimental factors investigated in this thesis (see Chapter V).

IV. Threat Model

This section specifies the operational scenario this thesis is designed for and the trust assumptions implicit in it.

A. Operating Scenario

The intended user is a defender or authorized penetration tester for performing reconnaissance against a network they are responsible for. The reconnaissance is part of a structured engagement (internal security assessment, contracted penetration test, or compliance-driven vulnerability scan) and follows the conventional phases. Passive reconnaissance (open-source intelligence gathering, DNS enumeration, traffic observation) precedes active reconnaissance (port scanning, service probing, OS detection) which precedes targeted vulnerability identification. The primary objective is to determine network composition and identify exploitable vulnerabilities. During this phase, the specific question is fingerprinting and CPE generation for IoT devices. The operator has Nmap output for a device. That output is incomplete or ambiguous, and the operator wants a structured identification suitable for downstream CVE lookup. The proposed solution uses an LLM to produce that identification.

B. Trust Boundaries

The threat the design addresses is data exfiltration through cloud-hosted LLM services. By its nature, a reconnaissance dataset contains a complete picture of the operator's network: live hosts, open services, software versions, and inferred vulnerabilities. Sending that dataset to a third-party API extends the trust boundary of the engagement to include theAPI provider, every subprocessor that provider uses, every employee with access to logged prompts, and every adversary capable of compromising any of those parties.

1) Compliance Implications

A defender operating under a compliance framework that governs the flow of sensitive data faces a more concrete problem than abstract trust extension. Frameworks such as Health Insurance Portability and Accountability Act (HIPAA), Federal Risk and Authorization Management Program (FedRAMP), Cybersecurity Maturity Model Certification (CMMC), and financial-sector regulations applicable to broker-dealers and registered investment advisers impose requirements on how covered data is handled. These requirements commonly include:

- data sovereignty, or maintaining control over where sensitive data is stored and processed;
- encryption in transit and at rest;
- audit logging for access to covered systems and data;
- contractual restrictions on subprocessors and third-party data handling;
- limits on transmitting operationally sensitive information outside the controlled environment.

Major cloud LLM providers offer enterprise tiers that claim to address some of these requirements. The standard consumer and developer APIs commonly integrated into LLM-augmented penetration-testing tools do not. Reconnaissance data describing live hosts, exposed services, software versions, and inferred vulnerabilities on a covered network is therefore not ordinary prompt content. It is operational security data about the protected environment. For an operator working under these regimes, routing a complete reconnaissance dataset through a cloud LLM is hard to defend unless the provider, contract, logging controls, retention policy, and data-processing location all satisfy the applicable framework. The design constraint that follows is local inference. In this thesis, "local" means inference executed on hardware controlled by the operator, with no network egress to a third party at inference time. This constraint caps the available model size and family at what the operator's hardware can host and what has been published as open-weight. That trade-off is the central tension this thesis examines.

V. Experiment Design

This chapter describes the experimental design and apparatus. The premise is narrow operationally: take an Nmap scan of an IoT device, pass it to a candidate LLM under a controlled prompt, record the model's predicted CPE strings, and score those predictions against manually-labeled ground truth. The sections below describe the pipeline that turns scans into scored predictions, the dataset on which it operates, the model lineup it covers, the prompts that condition each model's behavior, and the rubric that converts a generated CPE into a graded outcome.

A. Pipeline Architecture

The architecture is six steps. Take a scan. Ingest it into a MongoDB database alongside each device's ground-truth labels. Run the scan through every model under every prompt, in both normal and doubled forms. Record the response. Score the response against the ground truth. Aggregate. Each step is a separate command-line script that reads database state left behind by the prior step and writes its own state for the next step to consume. There is no shared in-memory state and no orchestrator. Any step can be re-run without having to re-run the prior steps, which matters because the slow steps (LLM inference) take days and the fast steps (re-scoring against a revised rubric) take seconds.

The invariant that makes the heterogeneous middle step tractable is that every inference backend writes the same document shape into the same MongoDB collection (`model_runs`). Whether the model was served via a hosted API, a local vLLM endpoint, or an Ollama Cloud relay, the resulting row records the same fields: the scan that was input, the prompt that was used, whether the doubled-prompt variant was applied, the trial number, the model identifier and sampling parameters, the raw response text, the parsed CPE list, and the scores assigned by the rubric. Scoring is implemented once (`score.py`) and applied uniformly. Aggregation and analysis (`export_thesis_data.py`, importing from `analyze.py`) operate on the unified schema and do not need to know which backend produced any particular row.

The original pipeline plan included a third stage that queried the NVD with each predicted CPE and reported the returned CVE list. That stage was dropped during development because it measured external database behavior rather than model capability. A predicted CPE that returns a useful CVE list does so as a property of the CPE, not of the model that produced it; once the CPE is correct enough, the

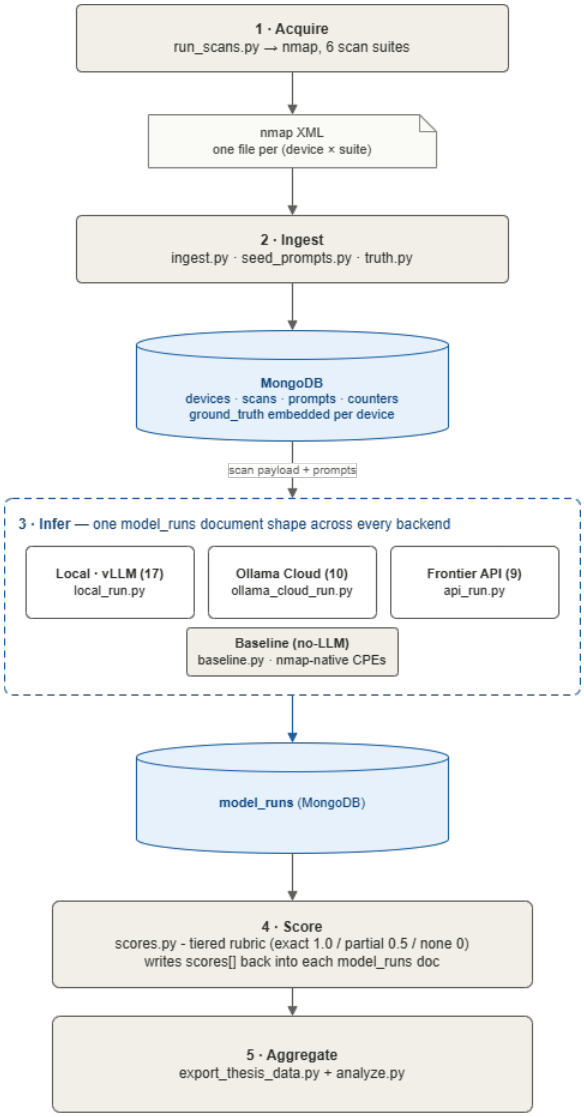

Fig. 1: Experiment Pipeline

lookup is mechanical [12]. The pipeline therefore evaluates CPE production and the operational *readiness* of those productions for downstream CVE lookup, rather than executing the lookup itself.

B. Database Design

All state lives in a single MongoDB database. An early prototype used SQLite with eight relational tables; the schema-on-write rigidity of that design proved a poor fit for an experiment whose variables changed during the study (the `doubled` prompt arm, for example, was introduced after database table schema and programs to digest it had already been written). MongoDB was adopted on advisor recommendation,

and the document-oriented model accommodated the schema evolution: fields that did not exist on early documents (`doubled`, `trial_number`, `model.guided_decoding`) were added to later documents without rewriting historical rows.

The database has five collections, summarized in Table I. The `devices` collection holds one document per physical or derived device, identified by a stable `device_code` (the directory name under `scans/`) and resolved by MAC address during ground-truth loading. The `scans` collection holds one document per ingested Nmap XML, with the raw XML, a parsed host dictionary, and the plain-text payload that gets transmitted to the LLM. The `prompts` collection holds one document per `(prompt_name, prompt_version)` pair, immutable once referenced by a run (Section F). The `model_runs` collection holds one document per `(scan, prompt, model, doubled, trial)` tuple — the core unit of evaluation. The `counters` collection backs an atomic sequence generator used to assign integer primary keys.

TABLE I
MONGODB COLLECTIONS AND THEIR ROLES

Collection	**Contents**
`devices`	One document per device. Embeds a `ground_truth` sub-document with `accepted_cpes`. Unique index on `device_code`; sparse unique on `mac`.
`scans`	One document per ingested Nmap XML. Stores raw XML, parsed host fields, and the plaintext payload sent to models.
`prompts`	One document per `(prompt_name, prompt_version)`. Immutable once referenced.
`model_runs`	One document per inference: `(scan, prompt, model, doubled, trial)`. Embeds `parsed_output.cpes` and the `scores` array.
`counters`	Atomic ID generator. Internal.

A `model_runs` document records the scan and prompt it was generated against, the doubled flag, the trial number, the model identifier and its sampling parameters as a sub-document, the raw response text, the parsed CPE list extracted from that text, and the score array written by the rubric. The schema is append-only at the application level: re-running an identical `(scan, prompt, model, doubled)` configuration writes a new document with an incremented `trial_number` rather than overwriting the prior run. This preserves the per-trial variance that Section G relies on.

C. Dataset

1) Physical Testbed

The physical testbed is comprised of 14 commercial IoT devices scanned on a controlled network, listed in Table II. Thirteen of these entries are included in the quantitative results; one was excluded because the scan/run data were not usable for scoring. Devices were selected to span the categories typical of a small-office or home-network environment: smart-home hubs, smart bulbs and plugs, networked cameras, networked printers, smart TVs and streaming devices, cable-modem gateways, retail-class WiFi routers, and a penetration-testing platform (included as a deliberately atypical case). Three of the entries (WiFi_Repeater_AP_Mode, WiFi_Repeater_Router_Mode, WiFi_Repeater_Repeater_Mode) are the same physical hardware operating in three different network modes; this is intentional, because the model's CPE prediction depends on the network behavior the scan can observe, not on the hardware identity, and the three modes expose different services.

TABLE II
PHYSICAL TESTBED DEVICES

Device code	Category	Notes
FireTV	Smart TV	Toshiba 50C350KU Fire TV Edition
Printer1	Multifunction printer	Canon imageCLASS MF753Cdw
Printer2	Multifunction printer	HP Color LaserJet Pro MFP M479fdw
TP-Link	WiFi router	WR841N v14, stock firmware
linksys	WiFi router	WRT54GS v7, VxWorks-era firmware
arris_gateway	Cable gateway	ARRIS Touchstone DG6450 DOCSIS gateway
pineapple_tetra	WiFi pentest appliance	Hak5 Tetra MK5.8, OpenWrt 19.07
WiFi_R_AP_Mode	WiFi extender	Cheap OpenWrt-based repeater, AP-mode scan
WiFi_R_Router_Mode	WiFi extender	Same hardware, router-mode scan
WiFi_R_Repeater_Mode	WiFi extender	Same hardware, repeater-mode scan, off-segment
WiFi_Range_Extender	WiFi extender	VxWorks-based whitelabel from Temu
WiFi_Camera	IP camera	White-label Chinese ESP-based camera
Smart_Plug_Sonoff	Smart plug	Sonoff/iTead, ESP-based
GE_LIGHT_7350	Smart bulb	GE/Savant Cync, ESP32-based

Each device was scanned six times per measurement run, once per scan suite (Section D). Devices were left in their stock factory configuration with default firmware; no jailbreaking, custom firmware, or network reconfiguration was performed beyond

what was required to place the device on the lab subnet.

2) Derived Testbed

Four entries in the full dataset were derived from external scan data rather than from physical devices in the lab. Three of these derived entries are included in the quantitative results; one was retained as a pipeline-behavior probe but excluded from scored accuracy measurement. Their Nmap scan XML was sourced from the Shodan IoT scan corpus and re-ingested through the same pipeline as the physical scans. The derived testbed is listed in Table III. These entries served to expand the list of devices. Additional derived datasets were evaluated but ultimately excluded due to incomplete scan data or the absence of verifiable ground truth.

TABLE III
DERIVED TESTBED DEVICES

Device code	**Category**	**Notes**
Vizio_TV	Smart TV	Vizio smart TV running Yahoo Connected TV (ca. 2013–2014); platform identifiable via TLS cert on port 8099.
Nintendo_WiiU	Game console	Signal-poor scan (no open ports, no banners). Used as a pipeline-behavior probe; excluded from accuracy measurement.
Wink_Hub	Smart-home hub	First-generation Wink Hub (made by Quirky); Linux 2.6, lighttpd 1.4.35 service banner.
SmartThings_Hub	Smart-home hub	SmartThings Hub V1 (Physical Graph, pre-Samsung acquisition). Signal-poor; only telnet and port 39500 open.

3) Ground Truth Labeling

Ground truth is stored as a `Truth.toml` file in each device's scan directory. Each truth entry records the device's true vendor, product, firmware version, and an `accepted_cpes` list — a hand-curated set of CPE 2.3 strings, each annotated with a tier (`exact`, `partial`, or `related`) indicating the operational quality of that identifier as an answer for the device. The exact tier represents a CPE that, when used as the key for an NVD lookup, would return the CVE list scoped to that specific device's deployed configuration. The partial tier represents a CPE that correctly identifies the device's operating-system family, vendor, or product line but at lower specificity (a wildcard version, a vendor-aliased product name, or an alternative vendor identification that maps to the same software stack). The related tier represents a CPE that correctly identifies a broader software family that is genuinely present (the Linux kernel inside

an Android system, for example, or a TCP/IP stack identification on an embedded device whose specific firmware has no NVD entry).

Two entries in the testbed have specific labeling caveats. The Nintendo Wii U scan contained no service banners and no responsive ports, leaving MAC vendor as the only identifying signal; the device was retained as a pipeline behavior probe but is excluded from accuracy measurements. The WiFi_Repeater_Repeater_Mode scan was collected from a different layer-2 segment than the device itself, so Nmap could not observe the device's MAC via ARP; ground truth could not be loaded for this device under the current MAC-based resolution scheme, and its scans are similarly excluded from quantitative results.

D. Nmap Scan Suites

Six Nmap scan suites are run against each device, listed in Table IV. The suites span a deliberate range of information density, designed to test whether a model's identification accuracy is more sensitive to evidence sparsity or to noise. A dense scan exposes more clues but may bury useful evidence among generic service output; a sparse scan is cleaner but may not provide enough signal for confident identification.

The first suite, `01-sv-osc-top1000`, is the most aggressive: it runs Nmap's service-version detection (`-sV`) at intensity 5, operating system detection (`-O`), and the default script set (`-sC`) against the top 1000 TCP ports. This produces the broadest and most signal-rich payload. The next three suites (`02/03/04-sv-intensity{1,5,9}`) hold the port selection constant and vary only the service-version detection intensity, isolating the effect of probe depth on banner-level signal. The final two suites are protocol-specific UDP probes: `05-dns-sd-udp5353` queries the DNS-SD multicast endpoint and `06-upnp-udp1900` queries the UPnP SSDP endpoint, capturing the service-discovery announcements that IoT devices use in lieu of standard TCP banners.

TABLE IV

NMAP SCAN SUITES. DEFINITIONS TAKEN VERBATIM FROM `CONFIG.TOML`.

Name	Nmap Flags
01-sv-osc-top1000	-sV --version-intensity 5 -O -sC --top-ports 1000 -T4
02-sv-intensity1	-sV --version-intensity 1
03-sv-intensity5	-sV --version-intensity 5
04-sv-intensity9	-sV --version-intensity 9
05-dns-sd-udp5353	--script dns-service-discovery -sU -p 5353
06-upnp-udp1900	--script upnp-info -sU -p 1900

Each scan produces its own XML, its own `scans` document in the database, and its

own model runs. The per-suite results in Chapter VI treat the six suites as independent measurement conditions and report performance broken out by suite, so that the noise-versus-sparsity question can be answered empirically rather than assumed.

E. Model Selection

The model lineup spans three deployment styles (local vLLM, Ollama Cloud, and frontier hosted API) plus a non-LLM Nmap baseline. Model selection was driven by what was practically available rather than by a closed-form sampling design. Open-weight models came from Hugging Face, picked by visibility in the cybersecurity-fine-tune literature and on the Hugging Face leaderboard at the time. The Ollama Cloud lineup is whatever the hosted service's catalog offered. Frontier model API choices tracked recency of release across the three major providers (OpenAI, Anthropic, Google). The selection is not exhaustive on any axis, but it is broad enough to support comparisons within and across tiers.

1) Local Models

Seventeen open-weight models were served locally via a vLLM server running on a multi-GPU workstation (4-GPU tensor-parallel configuration). The multi-GPU setup permitted models substantially larger than what a single consumer-class GPU could host, which extends the practical ceiling described in Chapter I (a 24 GB single-card budget) into the 24B–32B parameter range. This is acknowledged: the local results represent what is achievable on a well-resourced workstation, not on a single RTX-class card. The model lineup spans general-purpose instruction-tuned models (Mistral, Qwen, Llama, Phi, Gemma, Granite, OpenAI's open-weight release), security-domain fine-tunes (Foundation-Sec 1.1-8B and 8B, SecurityLLM, WhiteRabbitNeo, Llama-Primus), and one community fine-tune (`cjiao/goldengoose-corr-v4-random-200`).

2) Frontier API Models

Nine hosted API models were included to provide a reference ceiling against which local models are compared. Three are from OpenAI (`gpt-5.5`, `gpt-5.4`, `gpt-5.4-mini`), three from Anthropic (`claude-opus-4-7`, `claude-sonnet-4-6`, `claude-haiku-4-5`), and three from Google (`gemini-3.1-pro-preview`, `gemini-3.1-flash-lite-preview`, `gemini-3-flash-preview`). Frontier runs were executed via the `api_run.py` driver, which sends requests through each provider's official API client with provider-default sampling parameters. Sampling knobs (temperature, top-p, seed, max-tokens) are intentionally not forwarded to frontier endpoints: the design goal is to measure the

response a typical API consumer would receive when invoking these models in their default configurations.

The frontier coverage is uneven in one respect that affects the Results chapter. Anthropic and OpenAI models were not run under the doubled-prompt variant (Section 2)) because the per-token cost of running 582–590 doubled-prompt invocations per model against the full scan corpus exceeded the available budget. Gemini models were run under the doubled variant. The implication is that the doubled-vs-normal comparison (RQ2) is valid for the local vLLM models, the Ollama Cloud models, and the Gemini family, and is undefined for OpenAI and Anthropic, where only the normal variant was executed. The Results chapter restricts RQ2 analysis accordingly.

3) Baseline: Nmap-Native CPE Extraction

Nmap itself emits CPE guesses when its version-detection database contains an entry that matches an observed service banner or OS fingerprint [11]. These guesses are present in the Nmap XML output as the `cpe` child of `service` and `osmatch` elements. The `baseline.py` script synthesizes a `model_runs` document for each scan with `model.name = "nmap"`, `doubled = false`, and `parsed_output.cpes` populated from the XML. These rows are scored by the same rubric as any LLM run and serve as the no-LLM reference: every accuracy claim in Chapter VI is grounded relative to whatever Nmap itself emitted from the same scan.

F. Prompt Design

1) Prompt Inventory and Versioning

Seven prompts were included, summarized in Table V. The inventory is structured as a 2×2 factorial across two dimensions, persona (present, absent) and output structure (specified, unspecified), plus three additional prompts probing distinct prompting strategies. The neutral/persona axis tests whether assigning the model a cybersecurity analyst persona affects fact-retrieval accuracy on this task; the minimal/structured axis tests whether providing format guidance (rules, schema, examples) improves output reliability independent of persona framing. The three additional prompts (`evidence_first`, `few_shot_grounded`, and `conversational-kind`) probe chain-of-thought reasoning, few-shot grounding with explicit abstention rules, and a casual conversational register, respectively.

Each prompt is stored in the `prompts` collection as a document keyed by the `(prompt_name, prompt_version)` pair. Re-running `seed_prompts.py` after editing a prompt at the same version updates the row in place; incrementing the version

TABLE V

PROMPT INVENTORY. CELL POSITIONS REFER TO THE 2×2 STRUCTURE FOR THE FIRST FOUR ENTRIES (PERSONA $\times$ STRUCTURE).

Prompt name	Variant cell	Design intent
`neutral_minimal`	no persona, no structure	Pure task description. Tests fact-retrieval baseline.
`neutral_structured`	no persona, structure	Adds CPE format rules and an example. Isolates format guidance from persona framing.
`persona_minimal`	persona, no structure	Adds an analyst persona. Isolates persona framing from format guidance.
`persona_structured`	persona, structure	Both axes positive. Matches the original `config.toml` default prompt.
`evidence_first`	chain-of-thought	Three-step reasoning (extract identifiers $\rightarrow$ identify vendor/product/OS $\rightarrow$ map to CPE) with an explicit abstention clause.
`few_shot_grounded`	few-shot	Three labeled examples covering strong, weak, and mixed evidence. Tests whether examples improve abstention behavior.
`conversational-kind`	casual register	Informal phrasing, no rules, no examples. Tests sensitivity to register.

field creates a new row, so any existing `model_runs` that referenced the prior version continue to resolve to the exact text against which they were generated. This preserves reproducibility across the iterative prompt authoring that the study went through. All results reported in this thesis use prompts at version 1.0.

2) Normal vs Doubled Prompt Variants

For each prompt, every model run is executed twice: once under a standard `system`/`user` message split (the *normal* variant) and once under a doubled construction in which the system role is dropped and the user message is constructed as `[prompt + scan + "--- REPEAT --- " + prompt + scan]`. The doubled arm tests whether structurally repeating the instruction after the scan payload affects model adherence to the requested output format.

The construction operationalizes a hypothesis from the long-context literature. Models have been observed to attend more reliably to information at the start and end of a long context window than to information in the middle, a phenomenon termed "lost in the middle" [19]. Nmap scan output for the more aggressive suites is sufficiently long (thousands of tokens for a top-1000 scan with full service-version detection and script output) that any instruction prefixed to it risks being attended to less than the scan body. Sandwiching the prompt around the scan places a copy at both the

beginning and end of the user message, ensuring that the instruction appears in the high-recency region the model is most likely to attend to regardless of how the model's attention prior is distributed.

The doubled variant is implemented uniformly for local vLLM and Ollama Cloud models. For frontier model APIs, the doubled variant was executed only for the Google Gemini family; Anthropic and OpenAI models were run under the normal variant only due to per-invocation cost. The quantitative analysis of doubled-versus-normal effects in Chapter VI is therefore restricted to the local vLLM, Ollama Cloud, and Gemini families. The Nmap-native baseline carries `doubled=false` on all rows because it is not an LLM and has no prompt.

G. Inference Configuration

1) API Models

frontier model APIs are accessed through their providers' official client libraries by the `api_run.py` script. Sampling parameters are not overridden: each request is sent with whatever defaults the provider's API applies to a chat-completion call with no sampling-control fields set. The motivation to report what a typical caller of these models would receive in production without specialized configuration. The implication is that frontier-model determinism is not under the experiment's control. Where two runs against the same prompt produce different outputs, the variance is attributable to the provider's sampling defaults, not to any control the experiment is varying.

2) Ollama Cloud Models

Ollama Cloud models are accessed through Ollama's hosted endpoint. Sampling parameters are forwarded where the OllamaAPI exposes them: temperature is set to `0.0`, top-p to `1.0`, max-tokens to `2048`, and the random seed to `0`. Not every Ollama Cloud model honors every sampling parameter; the request specifies them, but the server's behavior is at the server's discretion.

3) vLLM Local Models

Local vLLM models are accessed through vLLM's OpenAI-compatible chat endpoint. The same sampling settings as the Ollama Cloud arm are applied: `temperature=0.0`, `top_p=1.0`, `max_tokens=2048`, `seed=0`. Each scan is passed through every prompt in both normal and doubled forms, with up to four trials per configuration. Trial 1 is the default unguided run. Trials 2–4 were intended to test constrained decoding by passing a regex matching valid `{"cpes": [...]}` structure to vLLM's `guided_regex`

parameter, with trial 2 holding sampling constant at trial 1's settings, trial 3 raising temperature to `0.7`, and trial 4 additionally fixing a different seed. The constrained-decoding arm did not behave as intended (the constraint failed to enforce structure on output; see Chapter VII), and the primary RQ1 and RQ2 quantitative analyses use only the unguided trial 1 runs. The guided trials remain in the database and feed the methodological analysis in Chapter VII.

H. Scoring Rubric

For LLM-generated runs, scoring begins from the `parsed_output` field stored on each completed `model_runs` document. The inference runners extract JSON from the raw model response and keep only CPE-like strings that pass the runner-side structural filter: the string must begin with `cpe:2.3`, use part `h`, `o`, or `a`, and supply non-wildcard vendor and product fields. If no string survives that step, the run is still recorded as complete, but its `scores` array is empty. The Nmap baseline is the exception: `baseline.py` bypasses model-output parsing and stores Nmap-derived CPEs directly in the same `parsed_output` schema.

For each surviving CPE, `score.py` compares the prediction against the `accepted_cpes` list of the device the scan belongs to. Both the prediction and each accepted CPE are lowercased and padded to 13 colon-separated components: the `cpe:2.3` prefix components plus the 11 CPE 2.3 attribute fields defined in the specification [20]. An accepted CPE matches when every non-wildcard field in the accepted entry equals the corresponding field in the prediction; wildcard fields in the accepted entry accept any predicted value. When several accepted entries match, the highest-weighted tier wins. When none match, the prediction is assigned tier `none`.

TABLE VI
TIER WEIGHTS APPLIED TO SCORED PREDICTIONS

Tier	Weight	Operational meaning
`exact`	1.00	Accepted exact identifier
`partial`	0.50	Accepted lower-specificity or family-level identifier
`none`	0.00	No accepted CPE matched, or no parseable CPE reached analysis

The scorer also records field-level correctness flags (`part_correct`, `vendor_correct`, `product_correct`, `version_correct`), computed by exact field equality against the accepted CPE chosen as the scoring reference, excluding wildcard and not-applicable fields. For a `none`-tier prediction, `score.py` still selects the accepted CPE with the greatest overlap across part, vendor, product, and version so the flags have a reference

point; the tier and match score remain `none` and 0. These flags are field-overlap diagnostics, not a success rate over operationally useful predictions: a prediction can score `none` on tier while still registering a correct vendor field, so the vendor- and product-correctness figures reported in Chapter VI measure field agreement, not whether the identification was usable.

The MongoDB and analysis representations differ for runs that produce no parseable CPE. In MongoDB, those runs stay complete with an empty `scores` array. When `analyze.py` builds the frame that `export_thesis_data.py` freezes, each complete run with an empty `scores` array becomes one analysis row with `predicted_cpe = None`, `best_tier = none`, and `match_score = 0`; a response with multiple parsed CPEs contributes one row per CPE. The reported mean match score is the arithmetic mean over analysis rows:

$$\text{MeanMatch} = \frac{1}{|P|} \sum_{p \in P} w(\text{tier}(p)),$$

where P is the set of analysis rows in the aggregation group and w maps each row's tier to its weight in Table VI. The per-model, per-prompt, per-suite, and per-tier results use this same formula over the matching rows. The design penalizes both abstention and overproduction: an empty response contributes a zero row, and a response carrying one useful identifier alongside several unsupported ones earns credit for the useful one while the unsupported rows pull the average down.

The accepted-CPE lists are part of the rubric, not a neutral lookup table. They were expanded during the study to credit lower-specificity but defensible identifiers. The FireTV labels show the rule: they credit the specific `amazon:fire_os` identifier and give partial credit to `google:android`, because Fire OS is Android-derived, and to `linux:linux_kernel`, because Android runs on the Linux kernel. A model that calls a Fire TV's OS Android or Linux is less specific than one that names Fire OS, but it is not equivalent to naming an unrelated product. The `rubric_version` field on each `ground_truth` document records the label-rubric version under which its accepted-CPE list was authored.

I. Reproducibility and Determinism

Three mechanisms support reproducibility of the experiment. First, every sampling parameter that the experiment controls is fixed across runs: `temperature=0.0` (forcing greedy decoding where the inference engine honors it), and a fixed random `seed=0`. These values are recorded in the `model_runs.model` sub-document for each run, so the configuration under which any specific row was generated is recoverable from

the database alone. Where a backend does not expose a given parameter, the raw output is still preserved for later inspection, but the generation process should not be described as exactly replayable.

Second, prompts are immutable once referenced. The `(prompt_name, prompt_version)` composite key ensures that any `model_runs` row resolves to the exact prompt text against which it was generated, even after the prompt source is later edited at the next version. The version of each prompt referenced by results in this thesis is 1.0.

Third, the entire database state is reconstructible from artifacts checked into source control: the scan XML files under `scans/`, the per-device `Truth.toml` ground-truth files, the prompt definitions in `seed_prompts.py`, the scan-suite definitions in `config.toml`, and the rubric implementation in `score.py`. The full database reset procedure is given in `Documents/README.md`; in summary, it drops the existing database, recreates the collections and indexes (`init.py`), reloads the prompts (`seed_prompts.py`), ingests every scan XML (`ingest.py`), loads the ground-truth files (`truth.py`), and is ready to accept fresh inference runs. The reset itself takes under a minute on a local MongoDB instance; the inference runs that populate the resulting empty `model_runs` collection take substantially longer, on the order of a week for the full vLLM sweep on the available hardware and a similar duration for the Ollama Cloud sweep.

Determinism is bounded by the inference engine. Even with greedy decoding and a fixed seed, runs against the same prompt may produce non-identical outputs when the inference engine introduces variance through batching, PRNG usage in internal operations, or hardware-level non-determinism in floating-point reductions [8].

The analysis reported in the remainder of this thesis operates on a frozen export of the database, generated by `export_thesis_data.py` (which imports the aggregation primitives from `analyze.py`). The export is the canonical analysis dataset: it pins the device, scan, prompt, model, and score counts at a specific moment in time so that subsequent edits to the live database — re-runs, additional scoring passes, ground-truth adjustments — do not silently change figures in the thesis. The counts reported in Chapter VI are the counts in that export.

VI. Results

This chapter presents the experimental results and analyzes the factors that shaped CPE-identification accuracy.

A. Model Performance

Figure 2 shows the performance pattern at the device-by-model level. Accuracy generally decreases as models get smaller: the frontier models remain relatively strong across a wider set of devices, the larger cloud-hosted models show the same pattern with lower overall success, and the smaller local models perform worst overall. The local tier is not uniform, however. Some models collapse to near-zero accuracy across many devices, while others, such as `Foundation-Sec-1.1-8B-Instruct` and `gemma-2-9b-it`, perform better than expected.

The figure also shows that the task is not uniformly difficult. Some devices receive moderate scores across many models, suggesting that their scan outputs contain enough evidence for useful product identification. Other devices remain difficult across nearly all model families, suggesting that those failures are driven by weak or ambiguous scan evidence rather than by a single poor model. Conversely, devices that are handled well by frontier models but poorly by local models mark the capability gap most relevant to this thesis: they are not impossible cases, but they appear to require reasoning capacity or prior product knowledge that smaller local models often lack.

1) Family

Mean match score varies greatly across model family, with a spread of 0.28. Figure 3 plots the per-model mean match score with standard error colored by family; Table VII aggregates the same data to the family level. The bottom three (Granite, Mistral, Llama) sit between 0.12 and 0.22. Notably, the gap from Claude (0.395) to Llama (0.116) exceeds the spread between the best and worst prompt (0.10) and between the best and worst scan suite (0.23, see Section C), making model-family choice the single largest source of variance in the experiment.

The Llama family's bottom position is driven primarily by the smallest model in the lineup. The Llama 3.x line was included to test the lower bound of the open-weight tier, and it sits there with an average exact rate of 5.1% and a hallucination rate of

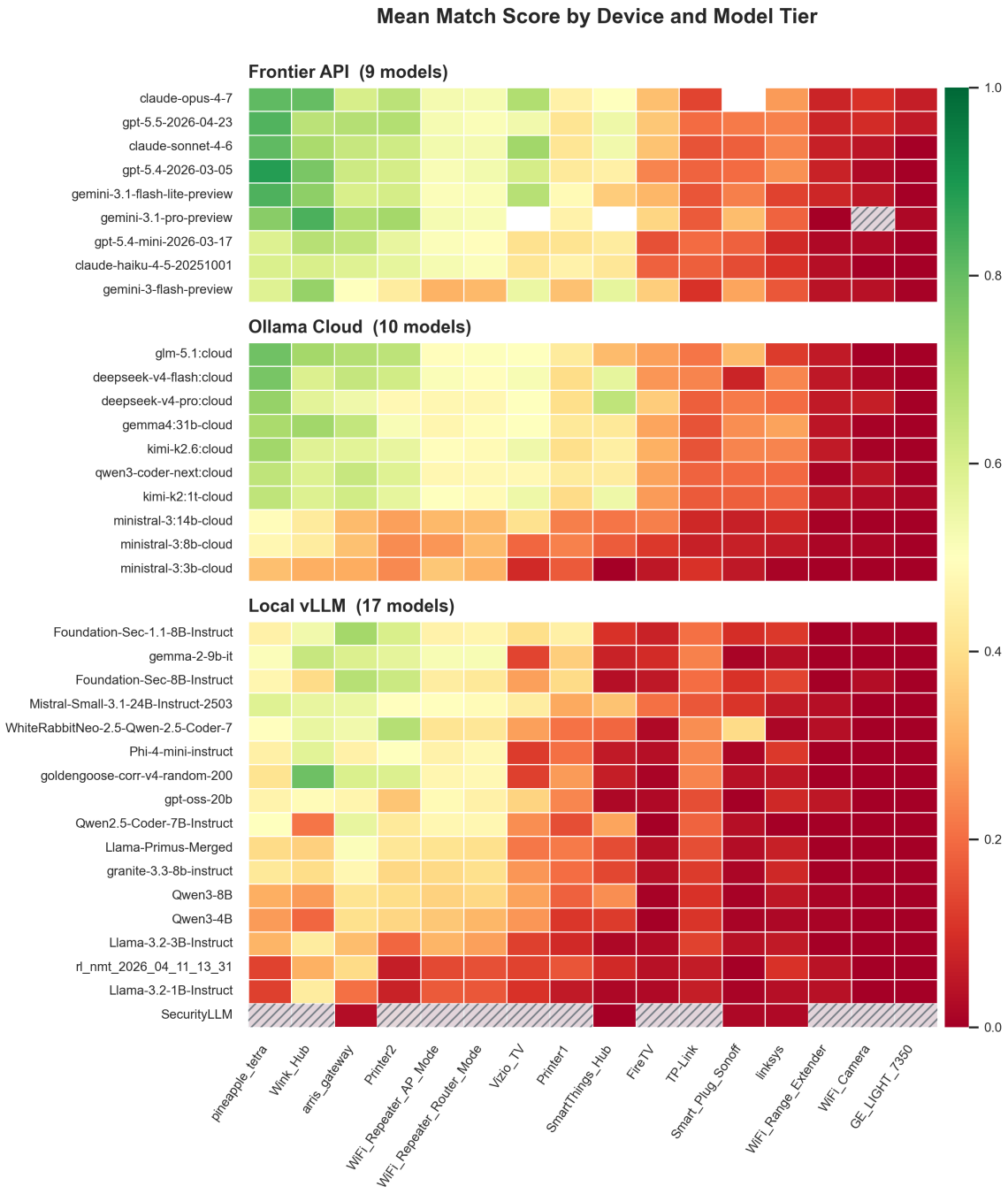

Fig. 2: Per-device, per-model mean match score.

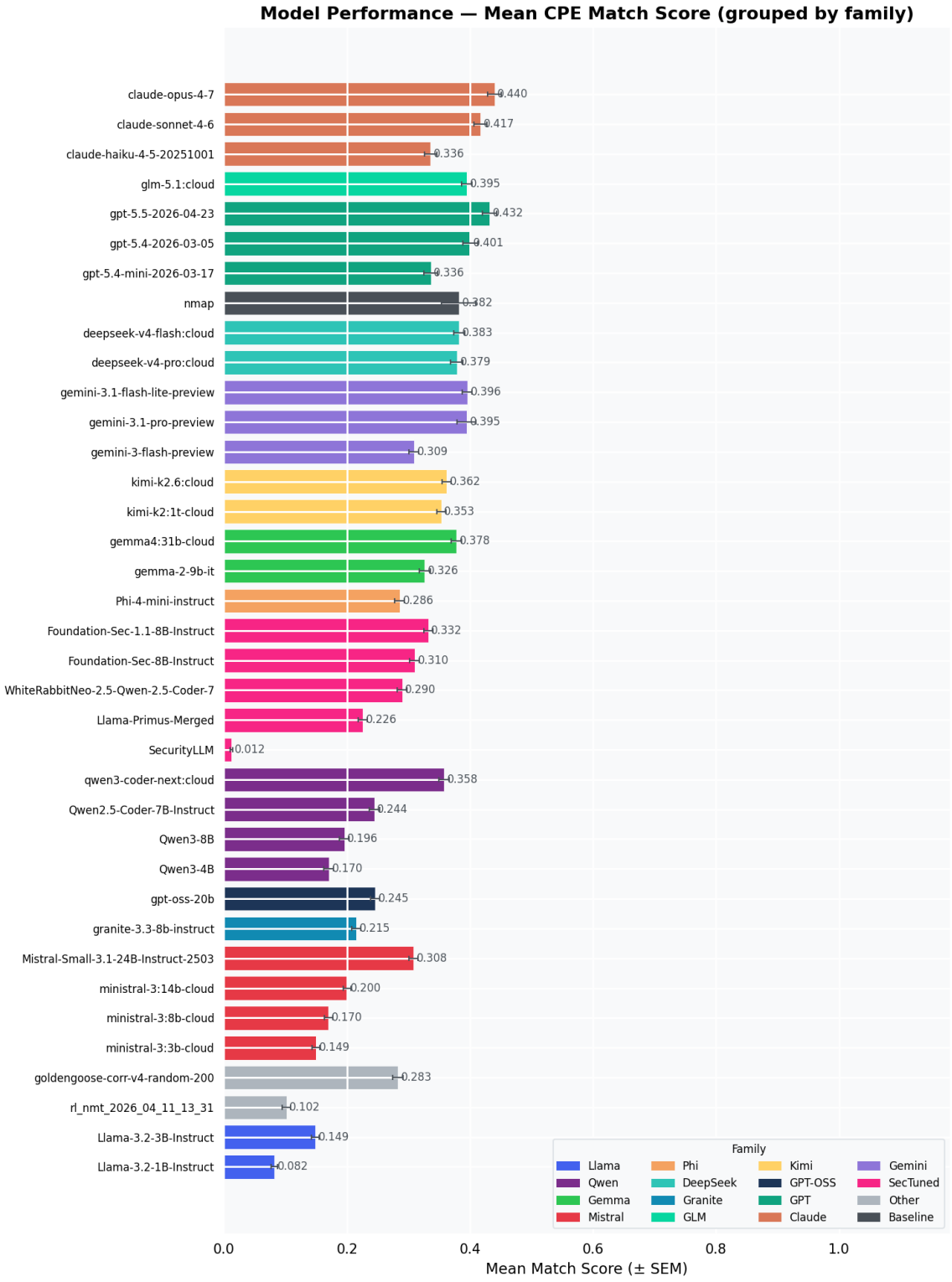

Fig. 3: Mean match score per model

TABLE VII
MATCH-SCORE AND HALLUCINATION RATE BY MODEL FAMILY

Family	Mean	Exact	Halluc.	Vendor	Product	# Models
Claude	0.395	0.166	0.166	0.740	0.555	3
GLM	0.395	0.173	0.158	0.686	0.587	1
GPT	0.390	0.167	0.143	0.722	0.554	3
DeepSeek	0.381	0.167	0.135	0.681	0.534	2
Gemini	0.358	0.146	0.132	0.697	0.504	3
Kimi	0.358	0.137	0.172	0.729	0.503	2
Gemma	0.353	0.159	0.143	0.678	0.521	2
Phi	0.286	0.139	0.223	0.581	0.439	1
SecTuned	0.258	0.108	0.217	0.558	0.378	5
Qwen	0.249	0.104	0.224	0.532	0.387	4
GPT-OSS	0.245	0.104	0.292	0.611	0.461	1
Granite	0.215	0.092	0.318	0.546	0.333	1
Mistral	0.211	0.085	0.264	0.542	0.353	4
Other	0.203	0.102	0.287	0.407	0.325	2
Llama	0.116	0.051	0.373	0.380	0.227	2

37.3%. The Mistral family shows the same pattern: it includes a 3B model whose poor performance drags the family average down.

B. Prompt Variation Effect

Prompt design affects mean match score across a 0.10 range, smaller than the family-level spread but larger than the doubled-prompt variant effect (Section 1)). Table VIII reports per-prompt accuracy and hallucination across all models in the experimental matrix.

TABLE VIII
MATCH SCORE, HALLUCINATION RATE, AND FIELD-LEVEL ACCURACY BY PROMPT

Prompt	mean	exact	halluc	vendor_acc	product_acc
`persona_structured`	0.321	0.138	0.247	0.630	0.483
`neutral_structured`	0.312	0.132	0.247	0.613	0.478
`few_shot_grounded`	0.303	0.147	0.184	0.553	0.454
`persona_minimal`	0.270	0.116	0.230	0.624	0.409
`neutral_minimal`	0.262	0.109	0.241	0.596	0.399
`conversational-like`	0.259	0.103	0.221	0.615	0.394
`evidence_first`	0.226	0.082	0.134	0.477	0.331

The 2×2 factorial across persona and structure isolates the contribution of each prompt-design axis. Table IX reports the marginal means. Structure — the inclusion of explicit CPE field rules, format guidance, and an example — adds +0.051 to mean

match score. Persona — the inclusion of a "cybersecurity analyst" framing — adds +0.009. The structure effect is roughly six times the persona effect, and the two best-performing prompts are precisely the two structured prompts.

TABLE IX
PERSONA × STRUCTURE 2 × 2 MARGINAL MEANS

Condition	Mean	N
Persona absent	0.287	18,301
Persona present	0.296	19,233
Structure absent	0.266	18,979
Structure present	0.317	18,555

The three other prompts each tell a different story. `few_shot_grounded`, which provides three labeled examples covering strong, weak, and mixed evidence, achieves the highest exact-match rate in the entire prompt inventory (0.147) and the second-lowest hallucination rate (0.184). Examples appear to influence models toward precise output when the evidence supports it and toward abstention when it does not. `evidence_first`, the chain-of-thought prompt with an explicit abstention clause, achieves the lowest hallucination rate (0.134) but also the lowest mean score (0.226), the abstention clause is doing what it was designed to do, refusing to commit to identifications under weak evidence, at the cost of also refusing identifications it could have produced. `conversational-like`, the casual informal-register prompt, performs similarly to `neutral_minimal` despite the very different design philosophy.

1) Doubled Prompt Variant

The doubled-prompt construction (Section 2)) tests whether sandwiching the instruction around the scan payload mitigates the lost-in-the-middle attention effect. Aggregated across the families where both modes were run (local vLLM, Ollama Cloud, and Gemini) the overall effect is essentially null. The mean match score is 0.270 under the normal variant and 0.267 under doubled, a difference of −0.003. Anthropic and OpenAI models were not run under the doubled variant (Due to the high cost of inference) and are excluded from this comparison.

The per-family decomposition in Table X reveals the lack of an overall effect. Small open-weight families gain modestly from doubling: Llama by +0.026, Mistral by +0.017, Phi by +0.016, Gemma by +0.016. Larger families lose: DeepSeek by −0.018, GLM by −0.016, Gemini by −0.013. The pattern is consistent with the lost-in-the-middle hypothesis [19]: smaller models attend less reliably to instructions buried in front of long scan payloads, and reinforcing the instruction in the high-recency region

helps them. Larger models attend to the original instruction without reinforcement, so the duplicate scan content in the doubled variant simply adds noise.

TABLE X
DOUBLED-VS-NORMAL MEAN-SCORE DELTAS PER MODEL FAMILY

Family	normal	doubled	Δ
Llama	0.102	0.128	+0.026
Mistral	0.202	0.219	+0.017
Phi	0.278	0.294	+0.016
Gemma	0.345	0.361	+0.016
Kimi	0.353	0.363	+0.011
Qwen	0.246	0.251	+0.005
GPT-OSS	0.246	0.245	−0.001
Granite	0.218	0.212	−0.006
Gemini	0.364	0.352	−0.013
GLM	0.402	0.387	−0.016
SecTuned	0.266	0.250	−0.016
DeepSeek	0.390	0.372	−0.018
Other	0.222	0.185	−0.038

C. Per-Scan Suite Performance

The six nmap scan suites span an information-density range described in Section D. Table XI reports per-suite performance across all LLM models and prompts.

TABLE XI
MATCH SCORE & HALLUCINATION RATE BY NMAP SCAN SUITE

Suite	mean	exact	halluc	vendor_acc	product_acc	n
`01-sv-osc-top1000`	0.343	0.133	0.212	0.688	0.557	14,607
`02-sv-intensity1`	0.341	0.149	0.184	0.655	0.497	11,324
`03-sv-intensity5`	0.340	0.151	0.187	0.649	0.498	11,262
`04-sv-intensity9`	0.255	0.152	0.287	0.521	0.331	7,324
`06-upnp-udp1900`	0.123	0.022	0.275	0.425	0.212	7,448
`05-dns-sd-udp5353`	0.112	0.024	0.229	0.413	0.155	6,946

Three patterns emerge. First, the top three suites are very close in mean score: `01-sv-osc-top1000` (0.343), `02-sv-intensity1` (0.341), and `03-sv-intensity5` (0.340) all cluster within 0.003 of each other. The most aggressive suite, which adds operating-system detection and the default nmap script set to the top-1000 TCP probe, produces essentially the same mean accuracy as the lightest TCP suite that runs only service-version detection at intensity 1. The addition of operating-system fingerprinting and script scanning does not seem to measurably assist CPE

identifications in the models.

Second, increasing service-version probe intensity from 5 to 9 (`03-sv-intensity5` versus `04-sv-intensity9`) drops mean match score by 0.085 and raises hallucination rate by 10 percentage points. The intensity-9 probe set is significantly more aggressive, sending probes that most services were not designed to respond to. The additional banner text it elicits appears to be more confusing than informative: models produce more output (more rows of attempted predictions) but at lower per-row accuracy and higher hallucination.

Third, the two User Datagram Protocol (UDP)-script probes — `05-dns-sd-udp5353` and `06-upnp-udp1900` — drop 0.22 below the worst TCP suite. These probes return only service-discovery announcements when the device participates in the relevant protocol and return nothing identifying when the device does not. The exact-match rate on these suites is 0.022–0.024, well below the TCP suites. This is the sparse-signal regime: there is not enough evidence in the scan output for the model to identify the device, and the models that attempt to identify it anyway are essentially guessing.

D. Local vs Frontier Comparison

This comparison addresses the thesis's central question: how much accuracy is the operator giving up by hosting the inference locally rather than routing scan data to a frontier API model. Three tiers are defined: Frontier for the nine proprietary models accessed through OpenAI, Anthropic, and Google endpoints; Ollama Cloud for the ten open-weight models served remotely through Ollama's hosted infrastructure; and Local (vLLM) for the seventeen open-weight models served locally. Table XII reports the tier-level summary.

TABLE XII
LOCAL, CLOUD, FRONTIER MODEL COMPARISON

Tier	mean	exact	halluc	vendor_acc	product_acc	n_models
Frontier (API)	0.378	0.158	0.145	0.717	0.533	9
Ollama Cloud	0.306	0.126	0.193	0.633	0.449	10
Local (vLLM)	0.231	0.101	0.261	0.524	0.371	17

Frontier models lead the tier comparison on every metric: +0.147 mean score over local, with hallucination roughly half (0.145 versus 0.261). Ollama Cloud sits between the two on mean score (0.306) but its tier mean understates the capability of its strongest entries; the tier is heterogeneous, mixing frontier-comparable models (`glm-5.1:cloud` at 0.395, `deepseek-v4-flash:cloud` at 0.383) with much smaller models (`ministral-3:3b-cloud` at 0.149) that drag the mean down.

The tier means describe averages, not ceilings. Table XIII reports the strongest model in each tier and the gaps between them.

TABLE XIII

BEST MODEL IN EACH TIER BY MEAN MATCH SCORE.

Tier	Model	Mean score
Frontier (API)	`anthropic/claude-opus-4-7`	0.4398
Ollama Cloud	`glm-5.1:cloud`	0.3946
Local (vLLM)	`fdtn-ai/Foundation-Sec-1.1-8B`	0.3320

The best frontier model beats the best truly-local model by 0.108 on mean match score. That is a smaller gap than the tier-mean gap of 0.147, and it is approximately equal to the gap between the best and worst prompt design (Section B). A defender restricted to local inference is not limited to frontier-equivalent capability, but they are roughly one prompt-design tier behind the best hosted model they could reach.

Two findings within the local tier modify this picture.

First, `fdtn-ai/Foundation-Sec-1.1-8B` (the best local model of reasonable size) reaches 0.332 on the strength of a security-domain fine-tune over an 8B base, which is small enough to host comfortably on a single consumer GPU. Performance close to the weakest frontier entry (`gemini-3-flash-preview` at 0.309) is achievable at a hardware cost well within the deployment constraints described in Chapter IV. Second, the local tier's hallucination rate is nearly double the frontier rate (0.261 versus 0.145), which compounds the accuracy gap from the operator's perspective: local-tier predictions are not only less accurate on average but more likely to be confidently wrong (see Section 1)).

The accuracy comparison above is meaningful only insofar as the models outperform the current non-LLM standard. Figure 4 shows per-model mean lift over the nmap-native CPE extraction baseline, computed per scan on the inputs where Nmap emitted a CPE of its own. Positive bars indicate the model produced higher-scoring identifications than Nmap's version-detection database did for the same scan; negative bars indicate it did worse.

The result cuts against the framing the tier means invited. A handful of hosted models clear the baseline cleanly: gemini-3.1-pro (+0.065) and gpt-5.5 (+0.065) lead, with deepseek-v4-pro (+0.039), Claude Opus (+0.041), and glm-5.1 (+0.038) following. But the bottom of the frontier tier does not. claude-haiku (–0.018), gpt-5.4-mini (–0.013), and gemini-3-flash (–0.003) all score below the deterministic baseline on the scans Nmap could handle. And the entire local vLLM tier is negative: Foundation-Sec-

1.1-8B, the best local model in the lineup, still lifts –0.060. No model served locally beat Nmap on its own output.

That last finding looks worse than it is, and the reason is a selection effect worth stating plainly. Nmap emits a CPE only when its version database already matches the banner, so its predictions concentrate on the easy scans and it stays silent on the rest. Its mean of 0.382 across 144 predictions is therefore higher than the frontier tier's 0.378 across more than ten thousand, not because Nmap reasons better, but because it never attempts the hard cases that pull the model means down. The lift metric compares the two only where Nmap produces output, which is in Nmap's favor, and a deterministic matcher with high precision and almost no recall is hard to beat. The operational value of an LLM here is not winning the head-to-head; it is producing a usable identifier on the scans where Nmap returns nothing at all, which is the regime this testbed was built to stress.

E. Failure Modes

1) Hallucination Rates

A hallucination, as scored in this thesis, is a prediction that is well-formed at the CPE-string level but whose part, vendor, or product fields do not match any accepted reference for the device. Across the 61,346 LLM predictions in the filtered set, 13,432 (21.9%) were flagged as hallucinations. Distribution across the four outcome categories: 11.9% exact, 32.4% partial, 0.0% related, 55.7% none. The `related` tier was not assigned to any prediction in the dataset.

Hallucination rate varies sharply across models. Table XIV reports the ten worst offenders.

TABLE XIV
MODELS WITH THE HIGHEST HALLUCINATION RATES

Model	Hallucination Rate
`Llama-3.2-1B-Instruct`	0.431
`granite-3.3-8b-instruct`	0.318
`Llama-3.2-3B-Instruct`	0.317
`goldengoose-corr-v4-random-200`	0.302
`ministral-3:3b-cloud`	0.300
`Qwen2.5-Coder-7B-Instruct`	0.300
`gpt-oss-20b`	0.292
`Llama-Primus-Merged`	0.284
`Qwen3-4B`	0.275
`ministral-3:14b-cloud`	0.272

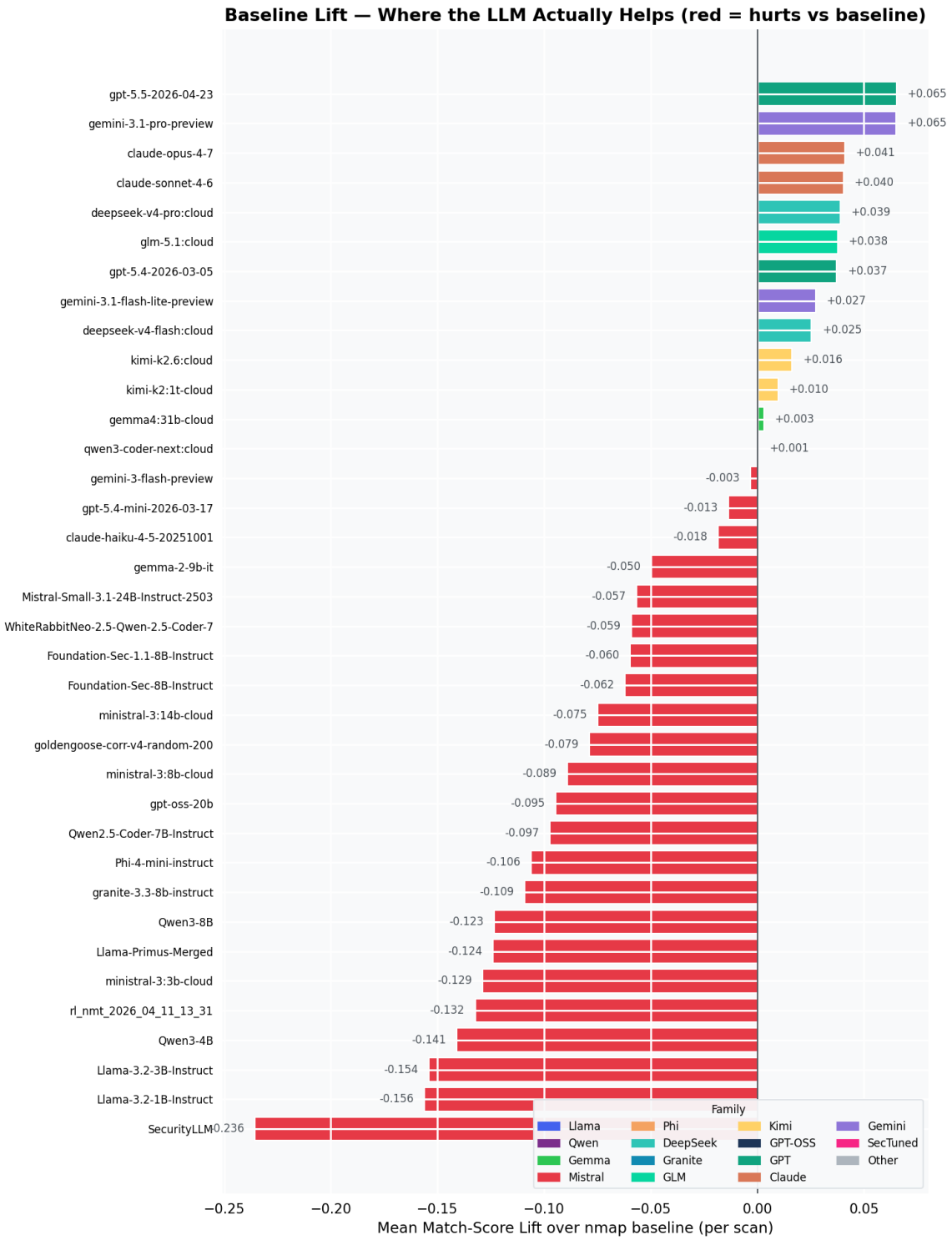

Fig. 4: Per-model mean per-scan lift over the nmap-native baseline.

The pattern at the high end of the hallucination table is clear: hallucination scales inversely with model size and frontier status. The single highest hallucinator is the smallest local model in the lineup (`Llama-3.2-1B-Instruct` at 43.1%); the next two are the next-smallest Llama and Granite entries. Seven of the ten worst hallucinators are under 10B parameters. The bottom of the table (lowest hallucination rates) is dominated by frontier models: Claude Opus at 0.141, GPT-5.5 at 0.122, Gemini 3.1 Flash Lite at 0.132. Across deployment tiers, hallucination is 0.145 in the frontier tier, 0.193 in Ollama Cloud, and 0.261 in the local-vLLM tier — a near-doubling from frontier to local.

One model in the dataset reports a hallucination rate of 0.017 (`ZySec-AI/SecurityLLM`), which would appear to be the most reliable model in the lineup. It is not: the model produces parseable CPE output on only ∼3% of its output, with a mean match score of 0.012 and a product-accuracy of 0.007. The low hallucination rate reflects the model's near-total failure on the task rather than any precision in the few predictions it does give.

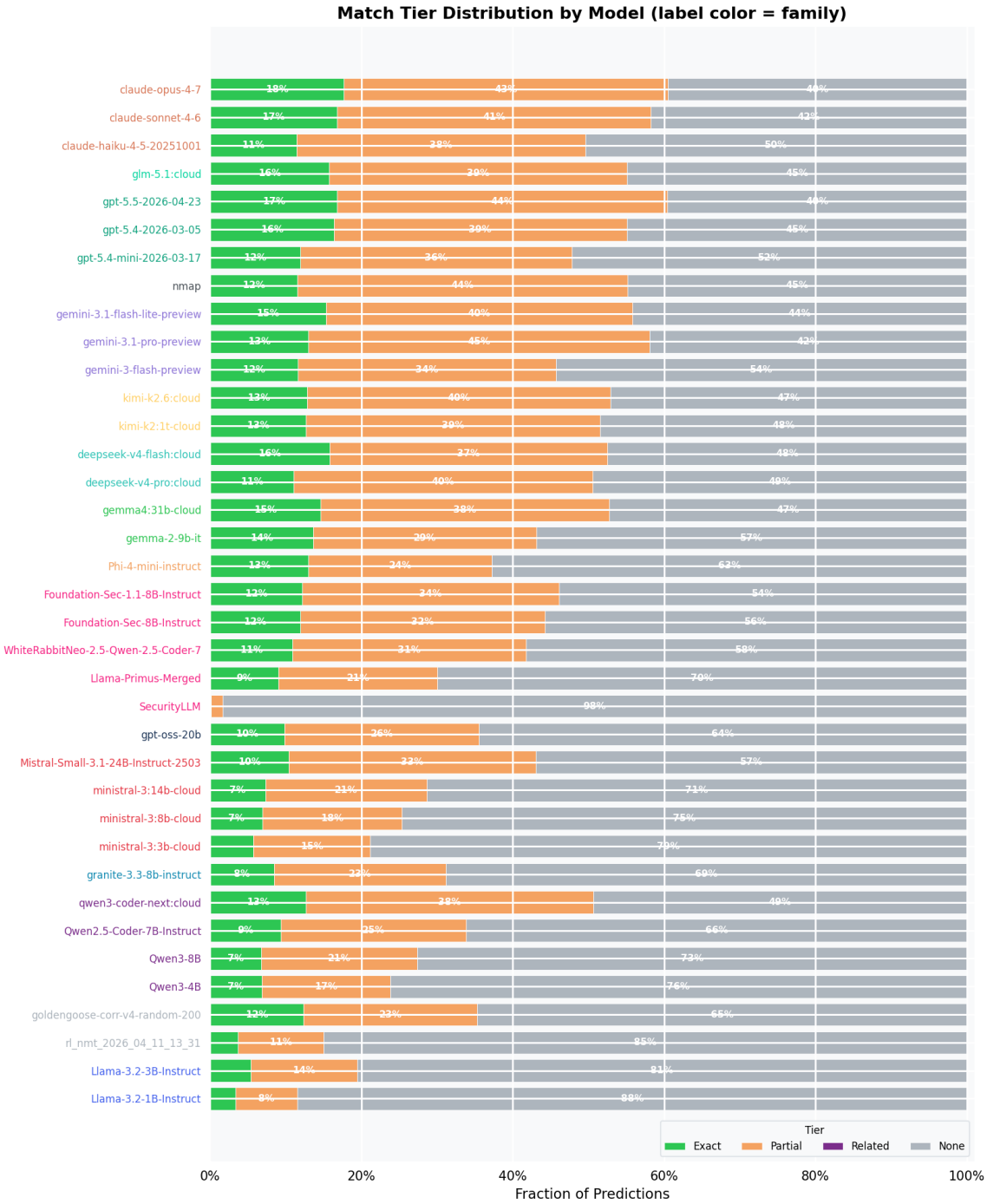

Fig. 5: Per-model distribution across scoring tiers

VII. Discussion

The Results chapter (Chapter VI) reported empirical accuracy, hallucination, and per-tier comparisons against a fixed experimental matrix. This chapter interprets them. The failures that matter most are the quietly wrong ones. The deployment-tier choice is a real trade-off rather than a clean win. The methodology measured a narrower slice of the problem than originally scoped.

A. Where Models Fail and Why

The most consequential failure modes observed in this study are not the loudest ones. A structurally malformed CPE string fails to parse, the downstream lookup never executes, and the operator immediately knows the prediction is unusable. A model that abstains entirely produces an empty response. Both failure modes degrade aggregate accuracy but neither produces operationally dangerous output. The failure modes that matter are the ones that produce well-formed, plausible-looking CPE strings whose downstream NVD lookup returns the wrong vulnerability list without flagging the error.

The constrained-decoding arm of the experiment, intended to address constrained decoding as a remedy for malformed output, did not behave as expected. The vLLM `guided_regex` parameter was specified for trials 2-4 with a regex matching the CPE V2.3 structure, but the with-CPE rates for guided versus unguided runs were close across all 17 vLLM-served models tested. The most obvious case was `ZySec-AI/SecurityLLM`, which emitted security-analyst narrative prose for 97% of its invocations: under a strict full-match guided-regex constraint, the model would have been forced to emit { as its first token and could not have begun any response with prose. The empirical behavior is consistent with one of two underlying causes: the vLLM server in this deployment treated the regex as a match-anywhere pattern, allowing arbitrary prose prefix before any pattern-matching content; or the server silently dropped the parameter for some or all model architectures.

A different pattern of silent failure appeared in two of the Ollama Cloud models in the lineup. `kimi-k2.6:cloud` and `qwen3.5:cloud` returned empty response bodies for more than 70% of their invocations across the original sweep. The other Ollama Cloud models served from the same infrastructure returned normal responses, which rules out a service-level outage. Diagnostic re-invocation of `kimi-k2.6:cloud` with a trivial prompt ("What is 2+2?") returned the expected answer along with the model's

internal reasoning trace stored in a separate `thinking` field of the response. The most likely explanation is that Kimi K2.6 and the recent Qwen3 series are reasoning models that consume most of their token budget on internal deliberation before producing user-visible content.

B. Trade-offs: Accuracy, Cost, Privacy

The choice between deployment tiers is the central operational question this thesis addresses. The accuracy gradient, established in Chapter VI §Local vs Frontier Comparison, is steep but not insurmountable: frontier models lead the local tier by 0.147 on tier-mean and by 0.108 on best-of-tier. Hallucination roughly doubles from the frontier tier (14.5%) to the local tier (26.1%). These numbers favor frontier deployment when accuracy is the only consideration. The remaining three axes complicate the choice.

The cost gap between tiers is approximately an order of magnitude per axis. Frontier model API pricing at the time of this work places Claude Opus, GPT-5, and Gemini Pro in the range of $15-75 per million input tokens, with output tokens priced higher. The Ollama Cloud open-weight tier charges substantially less per invocation, in many cases offering a free tier suitable for the scan volumes in a typical reconnaissance workflow. Locally hosted vLLM inference has zero per-invocation cost once the hardware is in place; the cost is the capital expenditure on the workstation. The Ollama Cloud tier's strongest models (`glm-5.1:cloud` at 0.395, `deepseek-v4-flash:cloud` at 0.383) reach mid-frontier accuracy at a small fraction of the per-token cost, and the per-invocation cost gap from Ollama Cloud to frontier is the most economically lopsided axis in the comparison.

Privacy is the axis on which the local tier wins decisively. A locally hosted model with no network egress to the inference provider extends no trust boundary at the time of inference. Ollama Cloud and frontier APIs both extend the trust boundary outside the operator's control. Major providers have substantially improved their enterprise data-handling guarantees, with business-tier agreements offering training-data exclusion, regional processing, and audit logging that begin to approach the requirements of the compliance frameworks discussed in Chapter IV. These guarantees mitigate but do not eliminate the trust extension.

C. Reflections on Experiment Design

The experiment as executed is narrower than the one originally scoped. The project began as a deployment exercise: the goal was to build a tool that would take a scan,

identify the device, query the NVD, and surface the resulting CVE list to an operator. As timeline constraints emerged, the scope contracted toward what could be measured cleanly and defended quantitatively. The final form is a comparative measurement study rather than a deployable product, which is reflected throughout this writeup. The pipeline isolates CPE production as its measurable endpoint, the downstream CVE lookup stage was removed because it measured NVD behavior rather than model capability. The thesis answers a specific quantitative question rather than delivering a finished operational tool.

Device selection was driven by a deliberate choice to make the task hard. Full-size general-purpose computers, laptops, and modern smartphones tend to produce nmap scan output rich enough that even the deterministic version-detection database identifies them at high specificity. Including such devices would have raised aggregate model accuracy on easy cases and obscured the differentiated performance the experiment was designed to measure. The physical testbed was therefore biased toward low-end IoT hardware: white-label Chinese cameras and extenders, end-of-life smart-home hubs, smart bulbs and plugs with minimal exposed surface, and consumer-grade routers. This selection produces a dataset on which most models perform poorly in absolute terms but on which the differentiation between models is empirically meaningful. The numbers reported in Chapter VI should be read in this context: they are not a general estimate of LLM reconnaissance accuracy but a measurement of LLM reconnaissance accuracy on the kind of devices that conventional fingerprinting tools struggle with most.

The frontier model API client tools (the conversational interfaces that ship with Claude, ChatGPT, and Gemini) were considered as an additional inference path and rejected. There were two reasons. First, those client tools route requests through routing layers that may select a different model variant than the one named in the API call, complicating reproducibility. Second, the tools maintain conversational memory across requests, which would have introduced cross-scan contamination as each device's prediction was conditioned on the model's accumulated context from earlier devices. The decision was to use the raw APIs for all hosted-model invocations to keep the experimental conditions controlled. This restricts the experiment from measuring the tool-assisted reasoning that an operator might actually benefit from.

Ground truth expansion improves fairness but introduces judgment calls. What counts as exact, partial, or wrong is ultimately my judgment as the researcher. While manual ground-truth labeling introduces inherent subjectivity, this is a recognized limitation in this evaluation framework and was mitigated by the researcher attempting

to remove personal tone and voice from the prompts. In the same light, the prompts are my choice as well. The variance of prompt choice is informed by prior literature, but the setup, structure and voice they convey are my own.

VIII. Conclusion

A. Summary of Findings

The first question asked which input variable moves CPE-identification accuracy most: deployment tier, model family, scale, prompt design, or scan suite. The answer is model family, by a clear margin. The spread across families was 0.28, from Claude at 0.395 down to Llama at 0.116; this is wider than the gap between the best and worst prompt (0.10) or the best and worst scan suite (0.23). Scale tracked family: the smallest models, `Llama-3.2-1B` and the 3B Mistral, sat at the bottom regardless of lineage. Prompt design mattered less and in a specific way: structural guidance added 0.051 to mean match score against 0.009 for the analyst persona, so format rules carried the prompt effect while role framing barely registered. The doubled-prompt variant, meant to counter lost-in-the-middle degradation, was null in aggregate (-0.003) but asymmetric by size — the smallest open-weight families gained slightly, the largest and the frontier models lost slightly, consistent with smaller models attending less reliably to an instruction buried ahead of a long scan.

On the second question, what an operator gives up by keeping inference local, the gap is real but survivable. Frontier models led the local tier by 0.147 on tier mean and 0.108 best-to-best, and their hallucination rate was roughly half (0.145 against 0.261). The cost is not disqualifying. `fdtn-ai/Foundation-Sec-1.1-8B` reached 0.332, within reach of the weakest frontier entry (`gemini-3-flash-preview` at 0.309), on an 8B base that fits a single 24 GB consumer card. The bottom of the local tier — `Llama-3.2-1B`, the `rl_nmt` fine-tune, and `SecurityLLM` — scored below the Nmap-native baseline, so model choice within the local tier matters more than the local-versus-frontier decision itself. A defender who must keep reconnaissance data in-house can do useful work with a well-chosen 8B model; one who reaches for whatever open-weight model is at hand can do worse than the deterministic tool they already have.

The third question concerned the quietly wrong answers. Across the filtered set, 21.9% of LLM analysis rows were confident and wrong: parseable CPEs with no accepted match and no correct vendor-field agreement. The rate scaled inversely with size and frontier status. `Llama-3.2-1B` hallucinated on 43.1% of its predictions, most of the ten worst offenders were under 10B, and the tier rate climbed from 0.145 at the frontier to 0.261 locally. This is the failure mode that matters, because it is silent: a well-formed but wrong CPE retrieves a plausible CVE list for the wrong product

and gives the analyst no signal that anything is off. One figure cuts against the grain and should not be misread — `ZySec-AI/SecurityLLM` posted the lowest hallucination rate in the study (0.017), but only because it emitted a parseable CPE on roughly 3% of its runs. Near-total abstention is not reliability.

B. Future Work

This thesis suggests three follow-up directions.

Quantization was deliberately excluded from the experimental matrix to reduce the number of independent variables. Adding aggressive quantization (4-bit, possibly 3-bit) would test whether the accuracy gradient observed across model sizes survives the additional precision reduction. It would also address the deployment constraint a consumer-grade single-GPU operator actually faces: fitting the largest possible model in available VRAM.

The current pipeline measures only the CPE-production endpoint of the broader reconnaissance phase. Reconnaissance in penetration testing also covers passive intelligence gathering, service-level vulnerability identification, exploit availability lookup, and target prioritization. Each of these involves the same ambiguous-input-to-structured-output transformation that LLMs are suited to. Extending the measurement methodology developed here across the wider pipeline, with appropriate tier scoring at each stage, would test whether the deployment-tier rankings established for CPE identification generalize to other reconnaissance subtasks. It would also give a defender a more complete operational picture when choosing a deployment approach across an entire engagement.

The security-domain fine-tunes evaluated in this study produced inconsistent results: high vendor recognition paired with elevated product invention, structured-output regression that collapsed the aggregate score for `SecurityLLM`, and middling overall placement despite explicit domain training. A more targeted follow-up would train a fine-tune against a CPE-identification objective directly, rather than against general cybersecurity narrative, and ask whether the result exceeds the per-model performance of a similarly-sized general-purpose model. The dataset assembled for this thesis (96,162 scored model runs across a documented experimental matrix) is suitable as both a training and evaluation corpus for such a fine-tune. Releasing it alongside the methodology would lower the threshold for follow-up work.

References

[1] K. Scarfone, M. Souppaya, A. Cody, and A. Orebaugh, "Technical guide to information security testing and assessment." NIST Special Publication 800-115, Sept. 2008.

[2] G. Deng, Y. Liu, V. Mayoral-Vilches, P. Liu, Y. Li, Y. Xu, T. Zhang, Y. Liu, M. Pinzger, and S. Rass, "PentestGPT: An LLM-empowered automatic penetration testing tool," 2024.

[3] B. Wu, G. Chen, K. Chen, X. Shang, J. Han, Y. He, W. Zhang, and N. Yu, "AutoPT: How far are we from the end2end automated web penetration testing?," 2024.

[4] D. Goyal, S. Subramanian, A. Peela, and N. P. Shetty, "Hacking, the lazy way: LLM augmented pentesting," 2024.

[5] W. Jansen and T. Grance, "Guidelines on security and privacy in public cloud computing." National Institute of Standards and Technology, December 2011.

[6] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, "Zero trust architecture," August 2020.

[7] S. Liu, D. Cheng, T. Xu, Z. Gou, *et al.*, "Quantization meets reasoning: Exploring LLM low-bit quantization degradation for mathematical reasoning," 2025.

[8] E. Kurtic, A. Marques, S. Pandit, M. Kurtz, and D. Alistarh, ""give me bf16 or give me death"? accuracy-performance trade-offs in llm quantization," 11 2024.

[9] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A.-R. Sadeghi, and S. Tarkoma, "Iot sentinel: Automated device-type identification for security enforcement in iot," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pp. 2177–2184, 2017.

[10] A. Sivanathan, H. H. Gharakheili, F. Loi, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Classifying iot devices in smart environments using network traffic characteristics," *IEEE Transactions on Mobile Computing*, vol. 18, no. 8, pp. 1745–1759, 2019.

[11] G. Lyon, "Nmap reference guide: Service and version detection." `https://nmap.org/book/man-version-detection.html`, 2024. Accessed: 2024.

[12] A. Sabetta and M. Bezzi, "Software vulnerability analysis using CPE and CVE," 2017.

[13] National Institute of Standards and Technology, "NVD developer documentation: Products (CPE)." `https://nvd.nist.gov/developers/products`, 2024. Accessed: 2024.

[14] A. Ramanujam, P. Soni, and B. Krishnamurthy, "CPE-Identifier: Automated CPE identification and CVE summaries annotation with deep learning and NLP," 2024.

[15] X. Shen, L. Wang, Z. Li, Y. Chen, W. Zhao, D. Sun, J. Wang, and W. Ruan, "PentestAgent: Incorporating LLM agents to automated penetration testing," 2024. AsiaCCS 2025.

[16] P. D. Luong, L. T. G. Bao, N. V. K. Tam, D. H. N. Khoa, N. H. Quyen, V.-H. Pham, and P. T. Duy, "xOffense: An AI-driven autonomous penetration testing framework with offensive knowledge-enhanced LLMs and multi-agent systems," 2025.

[17] L. Gioacchini, M. Mellia, I. Drago, A. Delsanto, G. Siracusano, and R. Bifulco, "AutoPenBench: Benchmarking generative agents for penetration testing," 2024.

[18] E. Wåreus and M. Hell, "Automated cpe labeling of cve summaries with machine learning," 07 2020.

[19] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, "Lost in the middle: How language models use long contexts," *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 157–173, 2024.

[20] MITRE Corporation, "Common platform enumeration: Naming specification version 2.3." MITRE Corporation, Nov. 2011.

[21] Z. R. Tam, C.-K. Wu, Y.-L. Tsai, C.-Y. Lin, H.-y. Lee, and Y.-N. Chen, "Let me speak freely? A study on the impact of format restrictions on performance of large language models," 2024.