MulitaMiner: An LLM-Based Tool for Structuring Vulnerability Scanner Report

Douglas Lautert[1], **Beatriz Machado**[1], **Cristhian Kapelinski**[1]
Diego Kreutz[1]

AI Horizon Labs | PPGES, Universidade Federal do Pampa (UNIPAMPA)

{douglaslautert,beatrizmachado,cristhianavila}.aluno@unipampa.edu.br
diegokreutz@unipampa.edu.br

Abstract. *Vulnerability scanners produce heterogeneous, vendor-specific PDF reports that hinder automated analysis and vulnerability management. We present MulitaMiner, an open-source tool that uses LLMs to extract structured records from these reports into a canonical 18-field schema via a segment-prompt-validate pipeline. Evaluated on 129 OpenVAS reports (6,343 findings) across three pipeline versions and five LLMs, MulitaMiner raises aggregate correctness from 67.5% to 78.3% and cuts omission from 14.1% to 2.8%; DeepSeek offers the best balance, tying for the lowest omission (2.7%) and posting the lowest hallucination (4.1%). MulitaMiner turns archived scanner PDFs into queryable data for CSIRT triage, prioritization, and tracking.*

1. Introduction

A university Computer Security Incident Response Team (CSIRT) [West-Brown et al. 2003][1] scans its assets every month and files the resulting PDF report. Asking which machines still expose a given CVE means reopening and comparing dozens of those files by hand. Among a CSIRT's routine activities is vulnerability management: periodically scanning the institution's assets, identifying known flaws, notifying those responsible for the affected systems, and tracking remediation. This routine, however, operates under growing pressure. In 2025 alone, more than 48,000 new Common Vulnerabilities and Exposures (CVE) records were published, 67% above the 2023 total and more than double the 2021 figure[2]. The National Vulnerability Database (NVD) continues to expand steadily[3], while CSIRTs, generally with small teams, must triage, prioritize, and communicate a volume of information that grows faster than their analytical capacity [Foreman 2019, Shimizu and Hashimoto 2026].

In practice, this cycle is sustained by vulnerability scanners. OpenVAS[4], an open-source tool, is used by academic and institutional CSIRTs; Tenable WAS[5], a commercial tool, is adopted by large network providers. The typical CSIRT workflow is to generate the scan report as a PDF, archive a copy as a historical record, and send the document to those responsible for each system, at which point the formal control of the process ends, with no systematic follow-up on the prioritization and resolution of vulnerabilities. This workflow results in two concrete problems. **Problem 1: reports stored as PDF.** The format, designed for human reading and lacking a queryable structure, prevents the accumulated data from being used for managing and triaging the state of systems, tracking vulnerability resolution, prioritizing fixes, and even training models[6]. Elementary

[1] NIST, SP 800-61 Rev. 3, 2025. [2] CVEdetails, CVEs published per year, 2026. [3] NIST, update on NVD operations amid record CVE growth, 2026. [4] Greenbone, OpenVAS, Open Vulnerability Assessment Scanner. [5] Tenable, Tenable Web App Scanning. [6] Redbelt Security, Vulnerability management: how to prioritize and remediate efficiently, 2024.

questions, such as which machines still present a given CVE, require manually reopening and comparing dozens of reports.

Problem 2: report diversity across tools. Each scanner adopts its own structure, fields, and terminology, which prevents consolidating results from OpenVAS and Tenable WAS, for example, into a single view of the infrastructure and undermines the construction of reusable security databases.

We present **MulitaMiner**[7], an open-source tool that uses LLMs to transform vulnerability scanner PDF reports into structured, queryable records. The tool directly tackles the two problems identified: for the first, it implements a modular pipeline that combines vulnerability-block segmentation, prompt engineering, and structural response validation, extracting from the PDFs records ready for automated processing; for the second, it adopts a canonical 18-field schema, independent of the source scanner, that unifies the results from different platforms into JSON. As a result, the same report history can be queried instead of reopened. MulitaMiner is the result of three evolutionary stages of the same tool (v1, v2, and v3), in which each version corrects limitations measured in the previous one; as Section 4 shows, this progression measurably reduces omission and hallucination and makes the five evaluated LLMs converge to nearly identical performance: the gain comes from pipeline engineering, not from swapping models.

Both problems have been addressed in the literature, but in contexts different from that of CSIRTs. In PDF extraction, tools such as MinerU [Wang et al. 2024a] and models such as DocLLM [Wang et al. 2024b] and DocExtractNet [Yan et al. 2025] convert visually rich documents into processable content, and Large Language Model (LLM)-based approaches have been expanding structured extraction in scientific documents [Dagdelen et al. 2024], clinical records [Vithanage et al. 2025], evidence syntheses [Mitchell et al. 2025], and other unstructured texts [Vijayan 2023], always for general purposes. A partial exception is [Machado et al. 2025], prior work that already extracts vulnerabilities from OpenVAS and Tenable WAS PDFs, but without a complete flow from the raw PDF to a multiformat dataset evaluated field by field at scale (the original evaluation covers a single report, 34 vulnerabilities). Aside from it, no work addresses scanner reports, which combine technical fields, reference lists, and free text repeated per host, nor evaluates field-by-field extraction in this type of document.

As for unification into known standards such as those of the NVD, systems such as ThreatKG [Gao et al. 2024] consolidate threat intelligence into knowledge graphs, and recent proposals apply LLMs to threat modeling [Elsharef et al. 2024], NLP in cybersecurity [Aghaei et al. 2025], prioritization [Tang et al. 2025], advisories [Kumar 2025, Abdullah et al. 2025, Ferrag et al. 2025], and scanner report generation [Dabholkar 2023]. These efforts, however, deal with textual threat intelligence sources or with summarization for human reading: what is missing is a unified schema that normalizes the heterogeneous scanner reports, the format in which CSIRTs accumulate their history, together with a systematic evaluation of extraction quality. Table 1 compares related work along the axes that decide whether a tool fits a CSIRT's workflow: input format, approach, models, output, dataset availability, and metrics.

None of the eight references in Table 1 combine, in the same tool, direct extraction from heterogeneous PDF, a scanner-independent unified schema, and simultaneous export

[7] `https://github.com/tooltmm-png/TMM/`

Table 1. Positioning of related work

Problem	Reference	Input	Methodological Approach	Models	Output	Dataset	Metrics
Free-text vulnerability extraction	[Chen et al. 2024]	Free text reports (GitHub Advisory)	Extraction as generation: LLM produces the affected package name from the description	GPT-3.5/4, LLaMa, Vicuna	Vulnerability-package pairs (tabular)	✓	Accuracy
	[Hashemi Chaleshtori and Ray 2023]	Unstructured descriptions	Entity recognition (transformer ensemble, majority voting)	BERT Ensemble	Labeled entities (format unspecified)	✗	Precision, Recall, F1
	[Machado et al. 2025]	Unstructured PDF reports	`Chunking`, prompt-based extraction, mapping to a unified schema, NULL for missing fields	GPT-4/4.1, LLaMA 3/4, DeepSeek	Structured dataset (export format not detailed)	✓	ROUGE-L (single report, 34 vulnerabilities)
Format divergence across vulnerability sources	[Dong et al. 2019]	CVE/NVD-linked reports (~70 thousand, 20 years)	Entity and relation extraction to detect inconsistencies across sources	Supervised entity recognition	Annotated corpus (sentence/word labels)	✓	Precision, Recall, F1
	[Wang et al. 2025]	Descriptions from 4 heterogeneous databases (NVD, X-Force, ExploitDB, Openwall)	Extraction of 7 aspects via LLM (zero-/few-shot) and semantic discrepancy	GPT-3.5, GPT-4	Structured aspects (field by field) + discrepancy report	✗	F1-score, Accuracy
	[Jiang et al. 2025]	Heterogeneous data (NVD, CVEdetails)	Entity/relation extraction, unified platform-naming schema, configuration graph	Entity/relation extraction + graph	Unified platform-naming schema + configuration graph	✓	Precision, Coverage
	[Alharbi et al. 2025]	NVD (vulnerabilities, weaknesses, platforms)	Autonomous knowledge graph via entity recognition and deduction rules	Natural-language processing + rules	Knowledge Graph (triples)	✗	Not reported
	[Liu et al. 2025]	Structured (NVD/CNVD) and unstructured data	Fusion via entity/relation extraction (sequence tagger + convolutional network), graph network, and LLM	Sequence tagger + graph network + GPT-3	Dynamic Knowledge Graph (triples)	✗	F1-score
Both	**MulitaMiner**	**PDF reports from multiple vulnerability scanners (OpenVAS and Tenable WAS)**	**Block-based segmentation + `chunking`, JSON validation/repair, 18-field schema**	**GPT-4/5, DeepSeek, LLaMA 3/4**	**Tabular records in the 18-field schema**	✓	**ROUGE-L, Token-F1, Soft-F1, Set-F1, exact match, F1-macro**

in multiple formats (JSON, CSV, TSV, XLSX) with reproducible field-by-field evaluation. Extraction efforts start from free-text advisories, not from raw scanner PDF; unification efforts operate on public sources that are already textual or semi-structured, never on a CSIRT's internal scan history.

Our main contributions are: (i) a canonical 18-field schema for the uniform representation of results from different scanners (Section 2), validated at scale on OpenVAS; (ii) an automated LLM-based structured extraction pipeline, with output validation and repair (Section 2); (iii) a field-by-field evaluation methodology that uses OpenVAS's native export as a reference (Section 3); and (iv) an evaluation with five LLMs, three pipeline versions, and 129 reports from vulnerable Docker containers (Section 4).

2. Implementation

MulitaMiner is a Python command-line application of about 12,500 lines, split between an extraction core, an evaluation package, and experimentation utilities. The per-package breakdown and a reproducible count script (`tools/count_loc.py`) are in the repository. The processing logic is distributed across 19 modules organized into four subpackages of `src/`: `configs/` (declarative configuration), `utils/` (ingestion, segmentation, `chunking`, and LLM access), `scanner_strategies/` (scanner-specific consolidation), and `converters/` (multiformat export). Figure 1 illustrates how these components connect; their effects on extraction quality are measured in Section 4.

Declarative configuration. All variable behavior is defined in files within the `src/configs/` package, in three groups: `llms/` (one JSON per model: endpoint, credential, temperature, and token/chunk limits); `scanners/` (a profile with prompt template, vulnerability-start regular expression, chunking policy, and consolidation field); and `templates/` (plain text prompts, with {`context`} filled in at runtime). Thus, adding an LLM or scanner amounts to creating a JSON file, with no change to the pipeline; besides OpenVAS and Tenable WAS, the distribution already includes profiles for Nessus, Qualys, Rapid7, and the corporate CAIS format.

Ingestion and segmentation. The `pdf_loader` module combines two readings

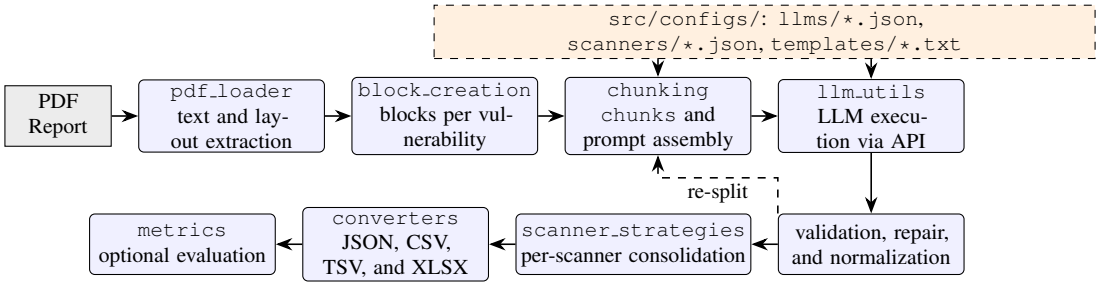

Figure 1. Organization of the MulitaMiner implementation. Solid arrows carry data between modules; dashed arrows carry declarative configuration and reprocessing on failure.

of the PDF, the running text and a layout reconstruction via `pdfplumber` to recover header context, regroups lines broken across pages, and normalizes ligatures and Unicode artifacts from these exports. On the cleaned text, `block_creation` scans the document using the profile's header regular expression and materializes one block per vulnerability, with port, protocol, and severity metadata; this metadata is propagated at the end to records whose fields come back empty or invalid from the LLM, reducing omissions in deterministic fields.

`Chunking` and prompt assembly. The `chunking` module slices each block using the model's tokenizer (`tiktoken`) within the LLM's JSON budget and refines the split by vulnerability markers, so that no record is cut in half. The order depends on the profile: for OpenVAS, token-based cutting first and then marker-based refinement; for Tenable WAS, the reverse, since base-finding and instance pairs cannot be separated. Each `chunk` is then injected into the profile's prompt template.

LLM execution, validation, and repair. The `llm_utils` module instantiates a `ChatOpenAI` client (LangChain) with a configurable endpoint, placing under the same interface the providers evaluated in this work (OpenAI, Groq, and DeepSeek) and local servers compatible with the OpenAI API. Each response goes through progressive JSON repair (four strategies, from direct parsing to removal of textual prefixes), token-budget verification, and truncation detection; failures trigger chunk re-splitting proportional to the error type and reprocessing. Each accepted record is normalized according to the 18-field schema (default values, type coercion, severity normalization, and metadata rejection) by a profile-specific validator.

Consolidation and export. The `scanner_strategies` package consolidates the records with one rule per scanner (grouping by name similarity in OpenVAS; merging base findings and instances in Tenable WAS) and writes audit logs. Export (`converters`) fixes a common load-validate-convert flow, with each writer specializing only the output format (CSV, TSV, or XLSX).

Evaluation and experimentation. The `metrics/` package, decoupled from the pipeline, is invoked with `--evaluate`: it compares the extraction against a configurable baseline, either expert-annotated or derived from the scanner's native export (used here, Section 3), using the metrics detailed in Section 3.1. The `tools/` directory provides experimentation utilities, such as `run_experiments.py` (batches with multiple LLMs and reports) and `dataset_generator.py` (consolidation into a single dataset).

2.1. Execution

MulitaMiner's interface is a single command line, centered on `main.py`, whose arguments select the declarative configuration files described earlier: `--input` indicates the PDF

report and is the only required argument, `--scanner` chooses the scanner profile, and `--llm` the model profile. Since all behavioral variation lives in the profiles, switching scanner or LLM requires no change beyond the argument value. The following command illustrates a typical extraction:

```
python main.py --input test/openvas/OpenVAS_JuiceShop.pdf --llm llama3 \
--scanner openvas --allow-duplicates --output-file openvas_test
```

The `--allow-duplicates` option controls consolidation: by default, records with similar names are merged by the scanner's strategy; with the option active, legitimate repeated occurrences of the same vulnerability (same finding on different ports, for example) are preserved, a behavior needed when the output will be compared against the scanner's baseline. `--output-file`, in turn, fixes the base name of the generated files, the JSON in the 18-field schema, and the audit logs; when omitted, the name is composed from the PDF, the LLM, and a timestamp, avoiding overwrites. The complete schema specification and the mapping of each field to the columns of the scanners' native export are documented in the tool's repository[8]. MulitaMiner does not read scanner output in formats other than PDF, and it does not judge whether a finding is a true vulnerability; of the six shipped profiles, only OpenVAS is validated at scale (Section 4).

Two further parameter groups control format conversion (`--convert`) and evaluation (`--evaluate` compares the output against a `--baseline` using ROUGE-L, an F-measure over shared subsequences, 0–1, higher better); the full option list is in the repository. For large-scale runs, `run_experiments.py` invokes `main.py` over combinations of reports and LLMs, consolidating metrics, spreadsheets, and charts; it was this batch mode that produced the results in Section 4.

3. Experimental Setup

The evaluation used a baseline of **129 OpenVAS reports**, obtained through dynamic scanning, on an isolated network, of Docker containers of common applications (`gitlab-ce`, `drupal`, `elasticsearch`) and deliberately vulnerable targets, such as OWASP Juice Shop and an application vulnerable to Log4Shell (full list in the artifact). Each scan was exported both as a PDF (MulitaMiner's input) and as OpenVAS's native CSV, with one line per vulnerability, totaling 6,343 records in the baseline (`vulnnet_scans_openvas.csv`), which preserves the original data and is referenced by every evaluation in Section 4. The evaluation focuses on OpenVAS due to practical barriers of the other scanners, not due to any limitation of the tool: most do not export PDF (as is the case with Nuclei) and commercial licenses have a prohibitive cost at scale (Tenable's cheapest plan costs R$ 21,190 per year[9]).

Each report was processed once by each of the three versions (v1, v2, v3) and five LLMs (DeepSeek, GPT-4, GPT-5, LLaMA 3, and LLaMA 4; GPT-4/GPT-5 use the `-mini` variants in Table 4): with temperature fixed at T=0, the output is practically deterministic, and scale (˜6k pairs per combination) stabilizes the rates without repetitions (exception: GPT-5, T=1 required by the model). Chunk size follows each LLM's context window. The versions differ in `chunking`: v1 can cut a vulnerability in half; v2 imposes marker-based boundaries with a fixed character cap; v3 adjusts the cap per LLM and repairs malformed JSON. Unlike the preliminary evaluation [Machado et al. 2025], whose baseline was manually annotated over a single report, the baseline here is derived from OpenVAS's

[8] Repository for version v3. `https://github.com/tooltmm-png/TMM/` [9] Tenable, Buy Nessus, 2026.

own structured export, without human curation (distinct forms of evaluation). In it, every evaluated field is filled in across the 6,343 records, which provides an objective reference and rules out small-denominator bias in the omission rates; the per-field scoring is given in Section 3.1 (Table 3, Appendix A).

(i) The `Name` field is used exclusively to align each extracted record with the corresponding baseline record (approximate matching, 85% threshold), forming the evaluated pairs; since it determines which records are compared, it is excluded from the scored set. (ii) The fields `log_method`, `plugin_details`, `instances`, and `source` have no counterpart in the OpenVAS CSV and, since they remain empty, are disregarded; the same applies to `plugin`, which is only partially filled. (iii) Extraction difficulty varies according to the origin of the information: single-column fields (`cvss`, `port`, `protocol`, `severity`) reach high accuracy from the earliest versions, while `detection_method`, `insight`, and `references` require consolidating multiple columns and free text.

3.1. Metrics used with the native baseline

The metrics in this paper are computed by `TMM_metrics_run.py` (`pandas`, `numpy`, `rouge-score`, `scikit-learn`), without dense embeddings; the original BERTScore (`metrics/bert/`, via `torch`) remains available for standalone evaluations and is not used here. ROUGE-L (longest common subsequence overlap) is the primary free-text metric; Table 3 (Appendix A) details, per field, the metric applied, and the Per-field overview (bold) is plotted in Figure 2.

Two metrics use a non-standard formulation. Soft-F1[10] reproduces BERTScore's greedy matching (each word paired with its most similar counterpart) using term-frequency vectors (TF-IDF) instead of contextual embeddings: precision is the average of the maximum per-word similarities of the extraction; recall, the same over the baseline's words; F1, the harmonic mean. The Set-F1 of `references` handles two edge cases: empty CVE sets on both sides count as a perfect match; only one side empty counts as a total error.

Each field of an aligned record pair falls into one of four mutually exclusive classes, used throughout Section 4: correct (Per-field overview $\geq$0.70), divergent (present on both sides, below the threshold), omitted (in the baseline only), and hallucinated (in the extraction only). In the aggregate rates in Section 4 we report correct, omitted, and hallucinated; the remaining fraction up to 100% corresponds to divergent. For a given LLM, Correctness is the mean of the per-field correct rates over all evaluated records; the Aggregate column of Table 2 averages that value across the five LLMs of each pipeline version.

4. Results

We analyze the batch run of Section 3 (Figure 2 and Table 2), tracking the progression from v1 to v3 and isolating each pipeline stage's contribution to the tool's overall extraction quality.

Per-field overview. Figure 2(a) shows deterministic fields (`cvss`, `port`, `protocol`, `severity`) already reach "Highly Similar" ($\geq$0.90) in early versions and approach perfect scores in v3, since these values are copied verbatim from the report and require no interpretation. Semantic fields (`description`, `detection_result`,

[10] Still labeled BERTScore in the tool's output files, a naming inconsistency we are correcting.

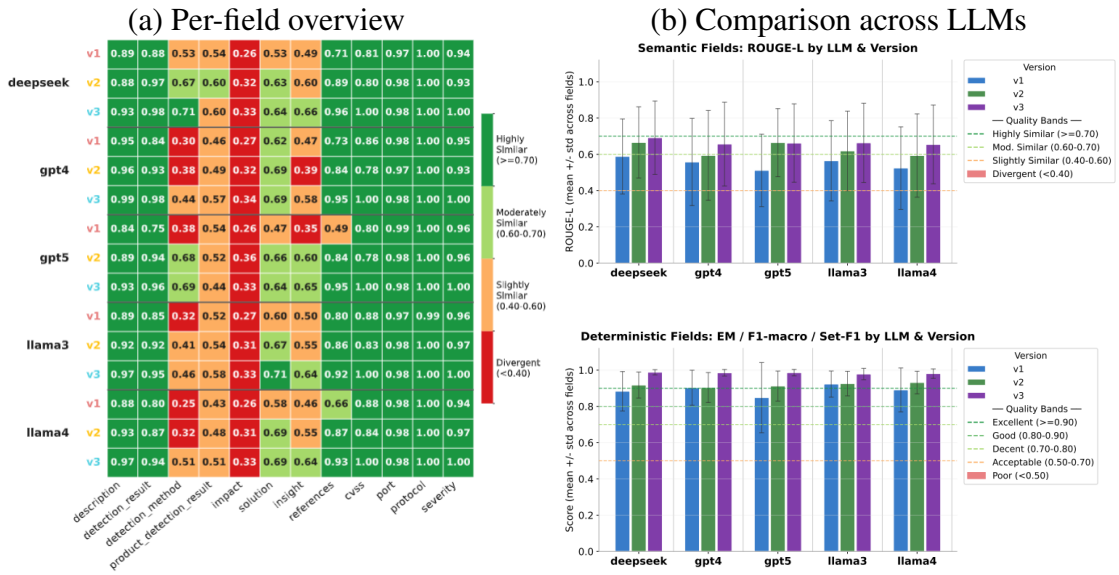

Figure 2. Per-field overview and comparison across the five LLMs, by pipeline version.

`references`) improve steadily and attain high agreement by v3 as the pipeline stabilizes chunk boundaries around each vulnerability. Free-text interpretive fields (`impact`, `detection_method`, `insight`) remain only moderately similar throughout: these fields ask the model to summarize or infer content that the PDF states more tersely than the fuller native baseline. These same fields later concentrate the residual error in Table 2.

Comparison across LLMs. Figure 2(b) and Table 2 reveals strong convergence by v3: the five models differ by at most one percentage point in correctness (77.9–78.9%), down from a six-point range under v1. Leadership shifts across versions (LLaMA 3 in v1, GPT-5 in v2, GPT-4 in v3): GPT-5, the weakest model in v1 (63.8% correct, 19.7% omission, largely from truncated fields), becomes the strongest in v2 once rigid block boundaries stop vulnerabilities from being cut mid-record, the model most sensitive to truncation being also the one that benefits most from its removal [Fan 2026]. This pattern repeats as each subsequent pipeline stage (marker-aware chunking, adaptive sizing, JSON repair) closes a structural gap common to all five LLMs, which is why the models converge rather than one simply outperforming the others: the bottleneck was never model capability, but how consistently each model received complete, well-formed input, a finding with a direct practical consequence for deployment, since teams adopting MulitaMiner are not locked into a single provider to obtain competitive results.

Omission and hallucination. Table 2 shows aggregate omission falling from 14.1% (v1) to 4.3% (v2) to 2.8% (v3), driven by stricter block boundaries and deduplication that preserve context the LLM previously lacked. The same boundary change raises hallucination, from 2.4% to 6.8% and back to 4.5%, since the model is now forced to populate fields instead of leaving them empty; v3's adaptive chunk sizing and JSON repair recover part of this increase while preserving the low omission. By v3, omission converges across all models, a property of the pipeline rather than of any single provider, while hallucination narrows but remains somewhat more model-dependent.

Aggregate summary. Table 2 confirms the pattern established above: omission concentrates in interpretive fields (especially `detection_method`), reaching near-zero for deterministic fields in v3, while hallucination appears mainly in default-valued fields (`port`, `protocol`) and, in v2, in free-text fields; v1's lack of rigid boundaries pro-

Table 2. Correctness, omission, and hallucination by LLM and pipeline version.

	DeepSeek			GPT-4			GPT-5			LLaMA 3			LLaMA 4			Aggregate		
Version	C	O	H	C	O	H	C	O	H	C	O	H	C	O	H	C	O	H
v1	67.4↓	13.7↑	2.4↑	69.2↓	11.9↑	2.8↑	63.8↓	19.7↑	**1.4**	69.8↓	11.3↑	2.7↑	67.3↓	14.1↑	2.5↑	67.5	14.1	2.4
v2	77.0↓	4.1↑	7.2↑	75.6↓	5.5↑	7.0↑	78.7↓	3.8↑	5.7↑	77.5↓	3.6↑	7.2↑	76.7↓	4.4↑	6.7↑	77.1	4.3	6.8
v3	78.3↓	**2.7**	4.1↑	**78.9**	2.8↑	5.5↑	77.9↓	2.9↑	4.2↑	77.9↓	2.9↑	4.4↑	78.1↓	**2.7**	4.7↑	78.3	2.8	4.5

C = Correct (%), O = Omission (%), H = Hallucination (%). Bold = best value overall (across all LLMs and versions) per metric. ↓/↑ mark values below/above that best.

duces GPT-5's peak omission (19.7%). Weighing correctness, omission, and hallucination jointly, DeepSeek delivers v3's best overall balance (78.3% correct, 2.7% omission, 4.1% hallucination); GPT-4 attains the highest correctness alone (78.9%) at the cost of the version's highest hallucination (5.5%), a deliberate trade-off rather than a tie. This convergence, driven by marker-aware segmentation and v3's adaptive chunking and JSON repair, demonstrates that the gains reported throughout this section are pipeline-driven rather than model-driven.

Processing cost. The 129-report v3 batch consumed approximately 119 million tokens and US$ 73.82. GPT-5 accounts for roughly 60% of the cost despite similar token volume to DeepSeek; LLaMA 4 is the most economical model (US$ 0.032 per PDF), about ten times cheaper than GPT-5 (US$ 0.342, Table 4). DeepSeek offers the best balance of correctness, omission, hallucination, and cost, making it the natural default for MulitaMiner deployments.

5. Final Considerations

MulitaMiner converts heterogeneous PDF reports into structured, queryable records with multi-LLM support and per-field evaluation. Version 3 achieves 78.3% aggregate correctness, less than 3% omission, and stable results across five LLMs, with DeepSeek offering the best overall balance (78.3% correctness, 2.7% omission, 4.1% hallucination). Overall, the results show that the gains from v1 to v3 stem primarily from pipeline engineering rather than stronger language models, with consistent improvements in correctness, omission, hallucination, and cost.

Discussion: `impact`, `detection_method`, and `insight` still need human review, since these summarize what the PDF only implies; deterministic fields (`cvss`, `severity`) are production-ready; external LLMs raise data-privacy concerns; local models are architecturally supported but not yet validated from native. **Limitations**: OpenVAS-only, native baseline; depends on external LLM APIs; no human validation loop for production deployments yet on . **Demonstration** (10 minutes). We process a fresh OpenVAS PDF through v3, showing a good case (`cvss` correct) and a bad case (`insight` omitted), then run `--evaluate` against a human-curated baseline for BERTScore; equipment: laptop, internet, power, API credentials. **Availability**: video[11] code, scanner profiles, and the 129-report baseline are in the repository under the MIT license; `tools/batch_pdf_extractor.py` and `tools/TMM_metrics_run.py` reproduce Table 2, and metrics regenerate from the persisted outputs with no further API calls. **Future work**: Future work includes mapping additional scanners and local providers, exploring field-driven re-extraction in v4, evaluating practical usefulness, and building a curated baseline.

[11] Installation and feature overview: `https://youtu.be/HHt8c_PofC0`

Acknowledgments. This work was partially supported by CAPES (Código de Financiamento 001), CNPq (Grant No. 409743/2025-9), and FAPERGS (Grant Nos. 24/2551-0001368-7 and 24/2551-0000726-1). The authors also acknowledge the use of Generative AI as a supporting tool for reviewing and improving the code, manuscript, and figures.

References

Abdullah, B. F. et al. (2025). Using llms for security advisory investigations: How far are we? *arXiv:2506.13161*.

Aghaei, E. et al. (2025). Securebert 2.0: Advanced language model for cybersecurity intelligence. *arXiv:2510.00240*.

Alharbi, H., Hur, A., Alkahtani, H., et al. (2025). Enhancing cybersecurity through autonomous knowledge graph construction by integrating heterogeneous data sources. *PeerJ Computer Science*, 11.

Chen, T., Li, L., Zhu, L., et al. (2024). Vullibgen: Generating names of vulnerability-affected packages via a large language model. In *ACL*, pages 9767–9780.

Dabholkar, C. S. (2023). Integrating open-source vulnerability scanning tools reports with Open AI API for automated report generation. Master's thesis, National College of Ireland.

Dagdelen, J., Dunn, A., Lee, S., et al. (2024). Structured information extraction from scientific text with large language models. *Nature Communications*, 15.

Dong, Y., Guo, W., Chen, Y., et al. (2019). Towards the detection of inconsistencies in public security vulnerability reports. In *USENIX Security*.

Elsharef, I., Zeng, Z., and Gu, Z. (2024). Facilitating threat modeling by leveraging large language models. In *AISCC*.

Fan, H. (2026). Capacity, not format: Rethinking structured reasoning failures. *arXiv:2606.09410*.

Ferrag, M. A., Alwahedi, F., Battah, A., et al. (2025). Generative AI in cybersecurity: A comprehensive review of LLM applications and vulnerabilities. *Internet of Things and Cyber-Physical Systems*, 5.

Foreman, P. (2019). *Vulnerability Management*. Auerbach Publications.

Gao, P., Liu, X., Choi, E., et al. (2024). Threatkg: An ai-powered system for automated open-source cyber threat intelligence gathering and management. In *1st ACM Workshop on Large AI Systems and Models with Privacy and Safety Analysis (LAMPS)*.

Hashemi Chaleshtori, F. and Ray, I. (2023). Automation of vulnerability information extraction using transformer-based language models. In *ESORICS Workshops*.

Jiang, Y., Shang, F., You, F. T. W., et al. (2025). Vulcpe: Context-aware cybersecurity vulnerability retrieval and management. *arXiv:2505.13895*.

Kumar, V. (2025). Automating security advisory evaluation through large language models. Master's thesis, Università degli Studi di Padova.

Liu, R., Xie, Y., Dang, Z., et al. (2025). Dynamic vulnerability knowledge graph construction via multi-source data fusion and large language model reasoning. *Electronics*, 14(12).

Machado, B., Lautert, D., Kapelinski, C., and Kreutz, D. (2025). Structured extraction of vulnerabilities in openvas and tenable was reports using llms. In *Anais da XXII Escola Regional de Redes de Computadores (ERRC)*, pages 144–150. Sociedade Brasileira de Computação (SBC).

Mitchell, E., Are, E. B., Colijn, C., et al. (2025). Using artificial intelligence tools to automate data extraction for living evidence syntheses. *PLoS One*, 20(4).

Shimizu, N. and Hashimoto, M. (2026). Vulnerability management chaining: An integrated framework for efficient cybersecurity risk prioritization. *IEEE Access*, 14:31407–31424.

Tang, L. et al. (2025). Polar: Automating cyber threat prioritization through llm-powered assessment. *arXiv:2510.01552*.

Vijayan, A. (2023). A prompt engineering approach for structured data extraction from unstructured text using conversational llms. In *ACAI*.

Vithanage, D., Deng, C., Wang, L., et al. (2025). Adapting generative large language models for information extraction from unstructured electronic health records in residential aged care: A comparative analysis of training approaches. *Journal of Healthcare Informatics Research*, 9(2).

Wang, B., Xu, C., et al. (2024a). Mineru: An open-source solution for precise document content extraction. *arXiv:2409.18839*.

Wang, D., Raman, N., Sibue, M., et al. (2024b). DocLLM: A layout-aware generative language model for multimodal document understanding. In *ACL*.

Wang, L., Sun, J., Jiang, J., et al. (2025). Vulnerability aspects extraction and discrepancies detection across heterogeneous threat intelligence. In *LM-SHIELD @ ACM AsiaCCS*.

West-Brown, M. J., Stikvoort, D., Kossakowski, K.-P., et al. (2003). Handbook for computer security incident response teams (CSIRTs). Technical Report CMU/SEI-2003-HB-002, Software Engineering Institute, Carnegie Mellon University.

Yan, Z., Ye, Z., Ge, J., et al. (2025). DocExtractNet: A novel framework for enhanced information extraction from business documents. *Information Processing & Management*, 62(3).

A. Metrics and Cost Details

Table 3 details the metric for each field (the Per-field overview from Section 4); Table 4 shows the token consumption and cost per model in the v3 batch (129 PDFs per model), using public API prices as of June 2026.

Table 3. Per-field scoring scheme.

Category and fields	Metrics
Free text: `description`, `detection_result`, `detection_method`, `product_detection_result`, `impact`, `solution`, `insight`	**ROUGE-L**, Token-F1, TF-IDF cosine, Soft-F1
Special: `references`	**Set-F1** over CVEs (regex)
Deterministic: `cvss`, `port`, `protocol`	**Exact match**
Severity: `severity`	**F1 macro**, exact match, confusion matrix
Not scored: `log_method`, `plugin_details`, `instances`, `source`, `plugin`	no column in the baseline (`plugin`: partial)

Table 4. Tokens and cost per model in the v3 batch (129 PDFs per model).

Model	Version	Input	Output	Cost	Cost/PDF
DeepSeek	`deepseek-coder`	22.87M	4.86M	US$ 4.56	US$ 0.035
GPT-4	`gpt-4o-mini-2024-07-18`	13.36M	5.23M	US$ 5.14	US$ 0.040
GPT-5	`gpt-5-mini-2025-08-07`	22.09M	4.51M	US$ 44.13	US$ 0.342
LLaMA 3	`llama-3.3-70b-versatile`	16.53M	7.67M	US$ 15.81	US$ 0.123
LLaMA 4	`llama-4-scout-17b-16e-instruct`	14.24M	7.66M	US$ 4.17	US$ 0.032
Total	–	**89.09M**	**29.93M**	**US$ 73.82**	–