MASTER'S THESIS DEFENSE · UNIVERSITY OF NEBRASKA–LINCOLN

AutoSARIF

Automated SARIF Support for Linters

Idriss Abdelmadjid

Adviser: Dr. Robert Dyer · School of Computing

November 18, 2026

MOTIVATION

Every linter reports findings in its own format

Bandit · Python

Dash-separated blocks, one per issue, with severity, confidence, CWE and location

ESLint · JavaScript

A file name line, with every finding for that file indented beneath it

PMD · Checkstyle · Java

One finding per line: path, line, rule and message

SARIF 2.1.0 is the shared JSON schema for tool metadata, rules and results. GitHub code scanning, Azure Pipelines and the VS Code SARIF Viewer all consume it, and so does recent research that feeds static-analysis alerts to LLMs (ZeroFalse, Munson et al., Qin et al.).

BACKGROUND · SARIF 2.1.0

What a SARIF file looks like

```
{ "version": "2.1.0",
  "runs": [{
    "tool": { "driver": {
      "name": "Bandit", "version": "1.8.3",
      "rules": [{ "id": "B101", … }]
    }},
    "results": [{
      "ruleId": "B101",
      "level": "note",
      "message": { "text": "Use of assert detected." },
      "locations": [{ "physicalLocation": {
        "artifactLocation": { "uri": "ChatTTS/core.py" },
        "region": { "startLine": 315, "startColumn": 9 }
      }}]
    }]
  }]
}
```

Run

One execution of one tool: schema version, tool name and version

Rules

tool.driver.rules: id, descriptions, help link

Results

One per finding: rule, level, message, file and line/column

In SARIF, lines and columns start at 1.

MOTIVATION

But SARIF support is uneven

NATIVE EXPORTER

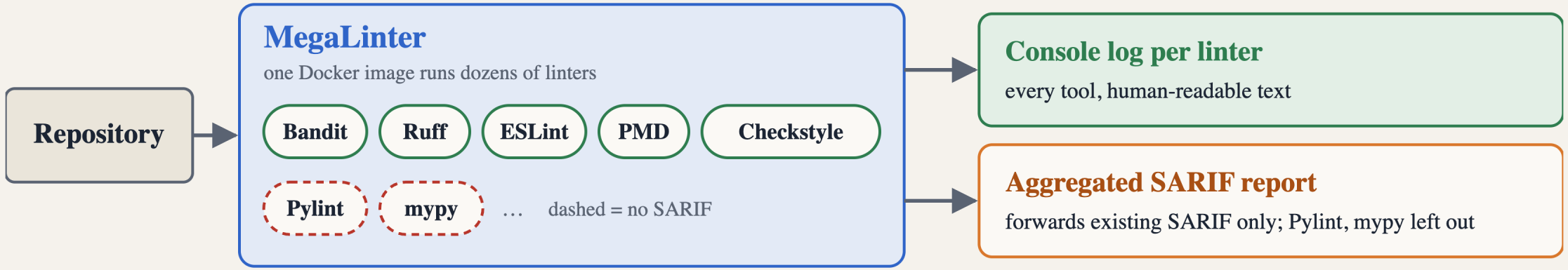
- Bandit ≥ 1.7.8, Ruff
- PMD ≥ 6.31.0
- Checkstyle ≥ 10.3.3

THIRD-PARTY ADAPTER

- Pylint, Flake8
- ESLint (separate formatter)

NO SUPPORT

- mypy, and others
- Researchers write own parsers



IDEA

The log already holds SARIF's facts

Bandit console log

```
>> Issue: [B310:blacklist] Audit url open for
permitted schemes. …
   Severity: Medium   Confidence: High
   Location: ./Scripts/get_all_links.py:28:14
27      try:
28          ret = urlopen(url, timeout=2)
```

SARIF result (simplified)

```
{ "ruleId": "B310",
  "level": "warning",
  "message": { "text": "Audit url open …" },
  "locations": [{ … "uri":
"Scripts/get_all_links.py",
    "startLine": 28, "startColumn": 15 }] }
```

Can an LLM, given only the console text and the tool's name, recover these facts reliably enough to replace an exporter? The hard parts: **long, nested JSON** is a known LLM weakness, and one run can report **thousands of findings**.

RELATED WORK

Where AutoSARIF fits

Line of work	What it does	How AutoSARIF is different
Adding SARIF by hand CogniCrypt · SARIF SDK	Engineers write a SARIF exporter or converter for one tool at a time	**No per-tool code: one prompt works on any linter's console log**
LLMs that improve analysis KNighter · IRIS · ZeroFalse	LLMs write new checkers, or decide which SARIF warnings are real bugs	**We don't change the analysis; we create the SARIF these tools take as input**
LLMs that structure text LogParser-LLM · LILAC	LLMs turn system logs into templates; benchmarks test JSON generation	**SARIF is far deeper and longer; we check structure and values separately**
Warning datasets Single-language datasets	Collections of warnings, used to study which ones developers fix	**Ours pairs each console log with the tool's own SARIF: 5 tools, 3 languages**

Gap: no prior work uses an LLM to turn a linter's plain console output into SARIF, for any tool and language.

RESEARCH QUESTIONS

Four questions guide the evaluation

RQ1 · CHUNKING

What is the best chunk size for converting console logs to SARIF?

RQ2 · FEASIBILITY

Can LLMs automatically convert linter console output into SARIF?

RQ3 · QUALITY

How does the generated SARIF compare with the SARIF the tools produce themselves?

RQ4 · PROMPT

What impact does prompt design have on the structure and accuracy of the SARIF?

CONTRIBUTIONS

What this thesis contributes

01 · Splitting and retry strategy

Token-aware, format-aware chunking with a round-based retry; chunk sizes derived from measured log-to-SARIF expansion

02 · Dataset

Paired console logs and official SARIF for 5 linters on 2,979 repositories; a verified subset of 1,137 pairs; the SARIF from 4 LLMs (Zenodo)

Dataset paper under review at MSR

03 · Empirical evaluation

5 linters, 3 languages, 4 LLMs, with separate metrics for completion, structure and values

Planned submission to EMSE

04 · Evidence on feasibility

Whether LLM conversion can replace the per-tool exporters the SARIF ecosystem depends on

APPROACH

The AutoSARIF pipeline

Console log

Input: one linter, one repo, plain text

```
>> Issue: [B101]
Location: …py:315:8
--------------------
```

1 · Splitter

Detects the format, cuts chunks of ≤ L tokens

chunk 1
chunk 2
chunk 3

2 · LLM

Zero-shot prompt per chunk → SARIF

```
{ "runs": [{
  "results": [ … ]
}] }
```

3 · Verifier

Results in SARIF = findings in log? Completeness, not correctness

log 12 = sarif 12

✓ pass ✕ fail

Chunk fails? Re-split it at half the limit and send it back to step 1, round after round: 3,200 → 1,600 → 800

ONCE EVERY CHUNK OF THE REPOSITORY PASSES

4 · Joiner

Merges chunks and rules, remaps indices

5 · Normalizer

Inlines URIs, splits multi-location results

SARIF 2.1.0

One verified file per repository

or

Incomplete

Left out; never a partial SARIF

APPROACH · SPLITTER

Format-aware, token-bounded chunking

Dash-separated

BANDIT

```
>> Issue: [B101:assert_used] …
   Severity: Low   Confidence: …
   CWE: CWE-703 (https://…)
   More Info: https://bandit…
   Location: …/core.py:315:8
315    assert vocos_ckpt_path, …
--------------------------------
>> Issue: [B101:assert_used] …
   Severity: Low   Confidence: …
```

Boundary: the dash line. Unit: one issue block.

Line-per-issue

ESLINT, PMD, CHECKSTYLE

```
examples/delayed.js
  19:3   warning  …  no-console
  23:3   warning  …  no-console
examples/events.js
  21:5   warning  …  no-console
  43:18  error    …  curly
  47:12  error    …  curly
examples/many.js
   2:5  warning … no-
unused-vars
```

Unit: a file line plus its indented findings.

Blank-line-separated

RUFF, FALLBACK

```
docs/example.py:1:1: F403 …
  |
1 | from manimlib import *
  | ^^^^^^^^^^^^^^^^^^^^^ F403
2 |
3 | class
SquareToCircle(Scene):
  |
⏎ (blank line)
docs/example.py:3:22: F405 …
```

Boundary: the blank line. Unit: one finding.

APPROACH · TOKEN LIMITS

Chunk sizes come from data

SARIF is **2–8× longer** than its log, so each chunk's SARIF must fit in **one LLM answer**.

1 How much longer is SARIF than the log?

Tool	Log ÷ SARIF	SARIF is
Ruff	0.4179	2.4×
Checkstyle	0.3396	2.9×
Bandit	0.3124	3.2×
PMD	0.1278	7.8×
ESLint	**0.1256**	**8.0×**

Average over all logs in the dataset, measured in tokens

2 How big can a chunk of log be?

32,000 max output (GPT-4.1) × **0.8** safety margin × **0.1256** ESLint ratio = **3,200** tokens per chunk

Safe for the most verbose tool, so safe for all. If a chunk still grows too much, the verifier catches it and the chunk is halved.

3 How small can a chunk be?

5 Bandit findings × **140** tokens each (most of any tool) ≈ **700** minimum tokens

APPROACH · PROMPTING

One zero-shot prompt for every tool

```
Convert the following {tool} ({version}) linter log into
valid SARIF 2.1.0 JSON. …
- all top-level keys (version, $schema, runs) …
Ensure that all errors in the log are included. …
IMPORTANT: For ruleId, always use the most concise, canonical
identifier …
… include as much information as you can in the output SARIF.
Under no circumstances may you omit, reduce, preview, or
truncate any content.
---
{input}
```

- No per-tool prompts, examples or extraction rules
- Explicit empty results array when a chunk has no findings
- Asks for rich output grounded in the log

Abridged; full text in prompts/convert-to-sarif.txt and the thesis listing.

EVALUATION · DATASET

Dataset and verified oracle

2,979

repositories: top 1,000 by stars per language (994 Python, 992 JavaScript, 993 Java)

300

verified pairs in the LLM sample: 50 per tool, 100 for ESLint, from small to very large logs

Commits recorded; tools pinned in MegaLinter v8.7.0

Oracle check: console finding count = SARIF result count

Tool	Pairs	Verified	Pass rate
Bandit	230	205	89.1%
Ruff	400	386	96.5%
ESLint	283	134	**47.3%**
PMD	196	184	93.9%
Checkstyle	245	228	93.1%
Total	**1,354**	**1,137**	**84.0%**

Four models, three measurements

Model	Kind	Start limit
GPT-4.1	Instruction	3,200
GPT-5-mini	Reasoning	3,200
GPT-5-nano	Reasoning	3,200
gpt-oss-20b	Open, local	800

gpt-oss-20b ran on Ollama on a Mac mini (M2 Pro, 32 GB); it only passed at 800 tokens in pilot runs.

Completion

Share of repositories where every chunk verified

Key metric (structure)

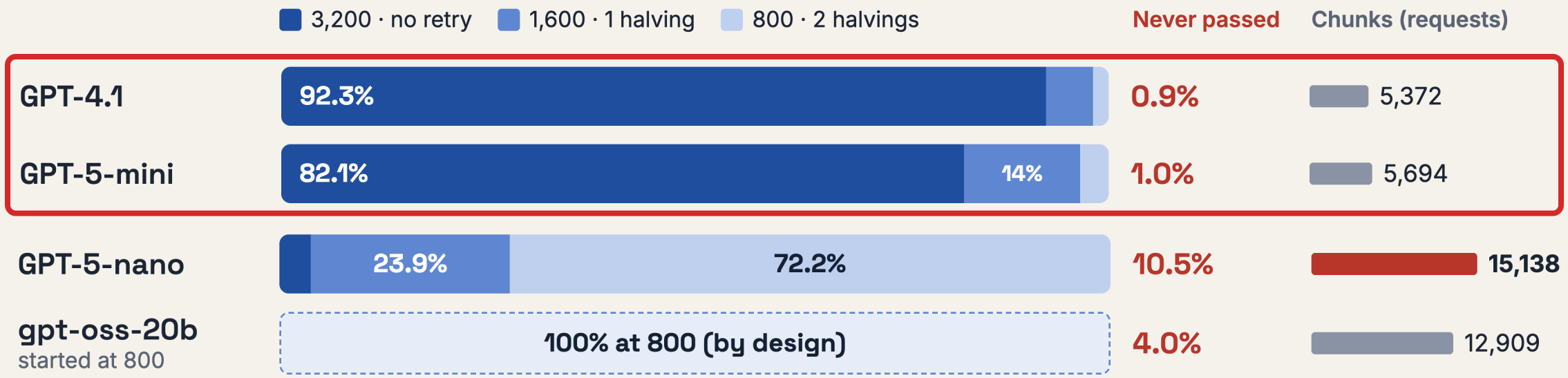
Key paths with counts: TP = min(o, l). Views: full, **fair** (only keys the log can supply), core (14 essential keys)

Value metric (content)

Results paired on file and line; each oracle value is correct, wrong or missing

RQ1 · WHAT IS THE BEST CHUNK SIZE?

3,200 tokens works for strong models



Answer: no single best size. Strong models pass >80% at 3,200; weaker models need ≤800, still fail more, and need ~3× the requests.

RQ2 · CAN LLMS CONVERT LINTER OUTPUT INTO SARIF?

Most repositories convert end to end

Repos completed	GPT-4.1	GPT-5-mini	GPT-5-nano	gpt-oss-20b
Checkstyle (50)	49	50	9	40
PMD (50)	4	2	5	2
ESLint (90)	90	90	49	48
Bandit (50)	50	50	49	45
Ruff (50)	50	50	40	41
Total (290)	**243**	**242**	**152**	**176**

Why PMD fails `LoosePackageCoupling - No packages or classes specified`

The last line of every PMD log is a config message, not a finding. Models turn it into **+1 result**, so the last chunk fails: **46 / 46** GPT-4.1 failures (GPT-5-mini 48 / 48). All but 7 other PMD chunks passed.

Answer: yes for most tools: GPT-4.1 and GPT-5-mini complete 97–100% of four tools. PMD is the exception.

RQ3 · STRUCTURE (KEY METRIC)

Structure is close to native SARIF

Tool	GPT-4.1	5-mini	5-nano	oss-20b
Checkstyle	0.988	0.985	0.888	0.951
PMD*	0.995	0.995	0.991	0.977
ESLint	0.998	0.997	0.983	0.974
Bandit	0.899	0.947	0.882	0.880
Ruff	0.849	0.876	0.928	0.833

Overall key F1, fair view, on each model's complete repositories. *PMD rests on 2–5 repositories.

Core keys: 0.96–1.00

For every tool and model, on the 14 essential keys

Rules part is weakest

F1 0.44–0.84: models add valid extra fields (name, fullDescription, default level)

A few result keys

Bandit code snippets left out (7,901×); Ruff end positions

RQ3 · VALUES (VALUE METRIC)

Values are right when the log shows them

Value F1, fair view	GPT-4.1	GPT-5-mini	GPT-5-nano	gpt-oss-20b
PMD · ESLint	1.000 · 0.999	0.999 · 1.000	0.996 · 0.990	0.989 · 0.992
Ruff	0.895	0.875	0.854	0.831
Checkstyle	**0.742**	0.822	0.809	0.815
Bandit	0.801	0.708	0.702	0.700

Checkstyle ids

Oracle uses final.parameter; the log shows FinalParameters

Bandit columns

Log is 0-based, SARIF 1-based: 12,651 off by one

Bandit levels

Low maps to note; models write warning

GPT-4.1 levels

warning for 39,045 of 86,773 [ERROR] findings

DISCUSSION · RQ4

Can LLMs replace exporters?

If the log prints file, line and rule, **one generic prompt** gives usable SARIF with no tool-specific code.

Feasible for most tools

Core-key F1 ≥ 0.96 everywhere. The cost moves from code to requests: 5,372 for GPT-4.1, almost 3× for GPT-5-nano.

Completeness depends on every chunk

One unverified chunk drops the repository; PMD shows how much a single stray line matters.

Weaker models fail differently

gpt-oss-20b misspelled keys; GPT-5-nano wrote 494 malformed messages. Both pass a count check, so add schema validation.

The log limits what can be recovered

Absent values (PMD columns, ESLint end positions) and hidden conventions (1-based columns) stay out of reach.

RQ4: one prompt, so its impact is not measured. Models mostly followed it; "never truncate" broke in 24 chunks.

THREATS TO VALIDITY

What could limit these conclusions

Threat	The risk	How I handled it
Construct Are we measuring the right thing?	The tool's own SARIF could have wrong fields; the fair view's rule was set after a first run	**Kept only pairs whose counts match the log (84%); report the full view too**
Internal Could my setup bias results?	Normalizer loses the file name of 545 gpt-oss-20b findings; temperature 1.0, one run each	**Counted and reported; it only lowers gpt-oss-20b. Claim only gaps above 0.02**
External Does it generalize?	Only tools that already export SARIF (needed as ground truth); popular repos; PMD on only 2–5 repos	**Next: tools with no exporter (mypy, Pylint) on a hand-labelled sample**
Reliability Can others repeat it?	LLM outputs change from run to run	**Commits, tool versions, prompts, outputs and scripts are on Zenodo**

FUTURE WORK

Where this goes next

Tools without SARIF

mypy, Pylint, with a small manually labeled sample

Cleaner input, stricter checks

Filter non-finding lines (PMD); add schema validation

Prompt variants

A log-fields-only prompt; a rule-id repair pass (RQ4)

Judging texts

Finish the local Qwen judge on differing messages

More models, repeated runs

Separate model differences from sampling noise

CONCLUSION

An LLM can stand in for a SARIF exporter

One zero-shot prompt plus chunk-level verification converts console logs to SARIF for most linters, without tool-specific code.

97–100%

of Bandit, Ruff, ESLint and Checkstyle repositories completed by GPT-4.1 and GPT-5-mini

0.96–1.00

key F1 on the essential keys, for every tool and model

>80%

of chunks pass at 3,200 tokens without retry for the strong models

PUBLICATIONS

Dataset paper: under review at MSR

AutoSARIF study: planned submission to EMSE

Thank you

Questions?

With thanks to Dr. Robert Dyer, Dr. Bonita Sharif, my committee, and my lab colleagues

Data, prompts, LLM outputs and scripts: AutoSARIF replication package on Zenodo