Project Context: snort_rag_rule_generator_new

Project root: /mnt/e/projects/snort_rag_rule_generator_new

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_new/
■■■ snort_rag_rule_generator/
    ■■■ data/
    ■■■ docs/
    ■   ■■■ reference_new_version/
    ■   ■   ■■■ snort_rag_rule_generator_new.pdf
    ■   ■■■ dataset_methodology.md
    ■   ■■■ dataset_validation_report.md
    ■   ■■■ pcap_testing_plan.md
    ■   ■■■ rapport_snort_rag_devoir3.docx
    ■   ■■■ rapport_snort_rag_devoir3.pdf
    ■   ■■■ report_section_dataset_kb.md
    ■   ■■■ report_section_retrieval_benchmark.md
    ■   ■■■ rule_validation_notes.md
    ■■■ notebooks/
    ■   ■■■ devoir3_snort_rag_architectures.ipynb
    ■■■ results/
    ■   ■■■ comparison_metrics.csv
    ■   ■■■ detailed_devoir3_results.csv
    ■   ■■■ embedding_benchmark.csv
    ■   ■■■ embedding_tsne.png
    ■   ■■■ retrieval_backend_proof.json
    ■   ■■■ retrieval_benchmark_k3.png
    ■   ■■■ retrieval_benchmark_k3_summary.csv
    ■   ■■■ retrieval_topk_examples.csv
    ■■■ scripts/
    ■   ■■■ benchmark_retrieval.py
    ■   ■■■ fetch_real_sources.py
    ■   ■■■ plot_retrieval_benchmark.py
    ■   ■■■ validate_dataset.py
    ■   ■■■ validate_rules_with_snort.sh
    ■■■ src/
    ■   ■■■ snort_rag/
    ■       ■■■ __init__.py
    ■       ■■■ app_gradio.py
    ■       ■■■ architectures.py
    ■       ■■■ evaluation.py
    ■       ■■■ generate_dataset.py
    ■       ■■■ generator.py
    ■       ■■■ knowledge_base.py
    ■       ■■■ retrieval.py
    ■       ■■■ rule_parser.py
    ■       ■■■ run_devoir3.py
    ■       ■■■ source_manifest.py
    ■       ■■■ templates.py
    ■■■ tests/
    ■   ■■■ pcaps/
    ■   ■   ■■■ README.md
    ■   ■■■ test_generate_dataset.py
    ■   ■■■ test_retrieval.py
    ■   ■■■ test_rule_parser.py
    ■■■ .gitignore
    ■■■ most_important_file.pdf
    ■■■ pyproject.toml
    ■■■ README.md
    ■■■ requirements.txt
```

2. Source And Text Files

snort_rag_rule_generator/.gitignore

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

```
FILE: snort_rag_rule_generator/.gitignore
```text
__pycache__/
*.pyc
.pytest_cache/
.env
venv/
.venv/
build/
dist/
*.egg-info/
*.pcap
*.cap
*.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

```
```

snort_rag_rule_generator/pyproject.toml

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

```
FILE: snort_rag_rule_generator/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"]

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

```
```

snort_rag_rule_generator/README.md

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

```
FILE: snort_rag_rule_generator/README.md
```markdown
# Snort RAG Rule Generator - NLP Mini Project + Devoir 3

Defensive NLP/RAG project for generating valid Snort IDS rules from natural-language network-attack descriptions.

## What this project contains

- An **official Person 1 retrieval dataset** stored in `data/processed/final_snort_dataset.csv`.
- A **trusted-source knowledge base** of real Snort rules stored in `data/knowledge_base/`.
- A legacy script to **expand trusted rules into experiment rows**: `python -m snort_rag.generate_dataset --multiplier 20`.
- Seven Devoir 3 architectures:
  - baseline without RAG
  - classic RAG
  - RAG with re-ranking
  - hybrid RAG (TF-IDF dense fallback + BM25 fusion)
  - multi-hop RAG
  - graph RAG
  - agentic RAG
- Metrics and comparison table in `results/comparison_metrics.csv`.
- t-SNE embedding visualization in `results/embedding_tsne.png`.
- Gradio dashboard with PDF upload to extend the knowledge base.
- Technical report in `docs/`.
```
```

Installation

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

Build the real-rule knowledge base

```bash
python scripts/fetch_real_sources.py
```

Outputs:

- `data/knowledge_base/trusted_rule_kb.csv`
- `data/knowledge_base/trusted_rule_kb.jsonl`
- `data/knowledge_base/fetch_summary.json`

Official Person 1 dataset

The official dataset used by the application, Devoir 3 runner, and retrieval layer is:

- `data/processed/final_snort_dataset.csv`
- `data/processed/final_snort_dataset.jsonl`
- `data/processed/dataset_summary.json`
- `data/processed/person1_rules.rules`

This dataset is personal, controlled, manually reviewable, and is the main dataset for Person 1 and the default retrieval c

The exported Person 1 rules are locally pre-validated with a strengthened validator. This does not replace Snort runtime va

Submission status:

- The official Person 1 dataset is `data/processed/final_snort_dataset.csv`.
- The final dataset is personal, controlled, balanced across 10 attack types, and contains 200 rows.
- The legacy trusted-rule expansion generator remains in the repository only as experimental code.
- The old 500k-row generated files are not included in the final submitted project.

Legacy dataset generator

The legacy generator requires the trusted real-rule knowledge base and creates
multiple natural-language rows per real rule for older experiments only.

```bash
python -m snort_rag.generate_dataset --multiplier 10
# bigger dataset
python -m snort_rag.generate_dataset --multiplier 30
```

Legacy outputs if you run the experiment manually:

- `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/experiments/legacy_generated/snort_generated_dataset_summary.json`

These files are legacy experimental artifacts only. The legacy generator no longer writes into `data/processed/` and must n

Run Devoir 3 comparison

```bash
python -m snort_rag.run_devoir3
```

Outputs:

- `results/comparison_metrics.csv`
- `results/detailed_devoir3_results.csv`
- `results/embedding_tsne.png`

Launch dashboard

```bash
python -m snort_rag.app_gradio
```

The dashboard lets you choose a RAG architecture, generate a Snort rule, inspect retrieved documents and add a PDF as a new

Project structure

```text
src/snort_rag/                 package source code
data/knowledge_base/           persisted trusted-source real Snort rules
data/raw/                      source manifest
```

```
data/processed/                official Person 1 dataset
data/experiments/legacy_generated/ legacy trusted-rule expansion outputs only
notebooks/                     Devoir 3 notebook
results/                       metrics and t-SNE plot
docs/                          report files
scripts/fetch_real_sources.py  trusted-source rule ingestion
```

Example

```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"])
```

Disclaimer

This is a defensive IDS rule generation project for education. Every generated rule must still be validated in a real Snort

```

## snort_rag_rule_generator/requirements.txt

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

FILE: snort_rag_rule_generator/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
```

snort_rag_rule_generator/docs/dataset_methodology.md

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

FILE: snort_rag_rule_generator/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-like
Les règles de `snort_rule_reference` sont des règles Snort-like éducatives, écrites pour rester simples et lisibles. Elles

## 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

## Limites
Le dataset reste petit et pédagogique. Les logs sont simulés, pas collectés depuis une infrastructure réelle. Les règles Sn

```
```

snort_rag_rule_generator/docs/dataset_validation_report.md

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

```
FILE: snort_rag_rule_generator/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 Snort-like doivent encore être testées dans un environnem

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 toutefois pas une vraie validation d'exécution Snort: la syntaxe finale, les avertissements moteur et le c

```
```

snort_rag_rule_generator/docs/pcap_testing_plan.md

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

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

Les tests PCAP constituent une validation comportementale future des règles exportées pour Person 1. Ils sont distincts de

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

Aucun résultat PCAP ne doit être affirmé dans la documentation du projet tant que le replay PCAP n'a pas été réellement exé

## Principe général

Workflow recommandé:
1. Valider d'abord la syntaxe et les avertissements moteur avec `scripts/validate_rules_with_snort.sh`.
2. Préparer un ensemble de PCAP synthétiques en laboratoire, un par scénario d'attaque principal et plusieurs PCAP bénins d
3. Exécuter Snort avec `data/processed/person1_rules.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:-/usr/local/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
snort -c "$SNORT_CONFIG" -R data/processed/person1_rules.rules -r tests/pcaps/port_scan.pcap -A alert_fast
```

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
snort -c "$SNORT_CONFIG" -R data/processed/person1_rules.rules -r tests/pcaps/ssh_bruteforce.pcap -A alert_fast
```

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
snort -c "$SNORT_CONFIG" -R data/processed/person1_rules.rules -r tests/pcaps/sql_injection.pcap -A alert_fast
```

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
snort -c "$SNORT_CONFIG" -R data/processed/person1_rules.rules -r tests/pcaps/xss.pcap -A alert_fast
```

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
snort -c "$SNORT_CONFIG" -R data/processed/person1_rules.rules -r tests/pcaps/command_injection.pcap -A alert_fast
```

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
snort -c "$SNORT_CONFIG" -R data/processed/person1_rules.rules -r tests/pcaps/directory_traversal.pcap -A alert_fast
```

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
snort -c "$SNORT_CONFIG" -R data/processed/person1_rules.rules -r tests/pcaps/dns_tunneling.pcap -A alert_fast
```

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
snort -c "$SNORT_CONFIG" -R data/processed/person1_rules.rules -r tests/pcaps/icmp_sweep.pcap -A alert_fast
```

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
snort -c "$SNORT_CONFIG" -R data/processed/person1_rules.rules -r tests/pcaps/malware_c2.pcap -A alert_fast
```

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
```

```
snort -c "$SNORT_CONFIG" -R data/processed/person1_rules.rules -r tests/pcaps/benign_traffic.pcap -A alert_fast
```

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

Sans cette exécution réelle, il ne faut pas présenter les règles comme validées opérationnellement. La pré-validation local

```

## snort_rag_rule_generator/docs/report_section_dataset_kb.md

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

```
FILE: snort_rag_rule_generator/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 ont été simulés dans plusieurs formats proches d'un SOC: Apache, Nginx, firewall, WAF, proxy, Zeek-like, SSH auth

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

```

snort_rag_rule_generator/docs/report_section_retrieval_benchmark.md

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

```
FILE: snort_rag_rule_generator/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 benchm

## 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

```
````

snort_rag_rule_generator/docs/rule_validation_notes.md

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

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

Les règles de `data/processed/person1_rules.rules` sont des références Snort-like éducatives. Elles ont été écrites pour so

Le script `scripts/validate_dataset.py` vérifie leur structure minimale:
- présence de `alert`
- présence de `msg`
- présence d'au moins une condition de détection
- présence de `classtype`
- présence de `sid`
- présence de `rev`
- unicité des SID

Le contrôle local a maintenant été renforcé avec des garde-fous Snort-like supplémentaires: équilibre parenthèses/guillemet

Différence importante:
- la pré-validation locale sert à filtrer les règles manifestement incohérentes avant export
- la validation réelle Snort est le test d'autorité final sur la syntaxe et les avertissements moteur
- des tests PCAP sont nécessaires pour vérifier le comportement de détection, la qualité des alertes et le niveau de faux p

Snort reste donc l'autorité finale, pas le script local.

Commande Snort à utiliser si Snort est installé:

```bash
snort -c /usr/local/etc/snort/snort.lua -R data/processed/person1_rules.rules --warn-all --pedantic
```

Limitation actuelle: `snort` n'est pas installé dans cet environnement de travail, donc seule une validation structurelle l

```
````

snort_rag_rule_generator/notebooks/devoir3_snort_rag_architectures.ipynb

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


````
FILE: snort_rag_rule_generator/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. Le
  ]
 },
 {
  "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
}
```

snort_rag_rule_generator/results/comparison_metrics.csv

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

FILE: snort_rag_rule_generator/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
graph_rag,1.0,1.0,1.0,1.0,1.0,1.0,0.711111111111111,0.03
rag_classic,1.0,1.0,1.0,1.0,1.0,1.0,0.711111111111111,0.03
rag_hybrid,1.0,1.0,1.0,1.0,1.0,1.0,0.711111111111111,0.03
rag_rerank,1.0,1.0,1.0,1.0,1.0,1.0,0.711111111111111,0.03
baseline,1.0,1.0,1.0,1.0,0.9,0.0,0.733333333333333,0.335
agentic_rag,0.9,0.85,0.9,0.866666666666666,1.0,1.0,0.7222222222222221,0.03
multi_hop,0.9,0.85,0.9,0.866666666666666,1.0,1.0,0.7222222222222221,0.0
```

snort_rag_rule_generator/results/detailed_devoir3_results.csv

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

```
FILE: snort_rag_rule_generator/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]
```
```

snort_rag_rule_generator/results/embedding_benchmark.csv

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

```
FILE: snort_rag_rule_generator/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,,0.9,0.9,0.9,0.9,6.6609100003915955
bm25,bm25,3,ready,True,True,sentence-transformers/all-MiniLM-L6-v2,False,,1.0,0.9666666666666668,1.0,0.95,6.750520099922141
bm25,bm25,5,ready,True,True,sentence-transformers/all-MiniLM-L6-v2,False,,1.0,0.9800000000000001,1.0,0.95,6.633099999817205
dense_tfidf,tfidf,1,ready,True,True,sentence-transformers/all-MiniLM-L6-v2,False,,0.9,0.9,0.9,0.9,1.6524399999980233
dense_tfidf,tfidf,3,ready,True,True,sentence-transformers/all-MiniLM-L6-v2,False,,1.0,0.9,1.0,0.95,1.8331499999476364
dense_tfidf,tfidf,5,ready,True,True,sentence-transformers/all-MiniLM-L6-v2,False,,1.0,0.9400000000000001,1.0,0.95,2.0661100
dense_tfidf_rerank,tfidf_rerank,1,ready,True,True,sentence-transformers/all-MiniLM-L6-v2,False,,1.0,1.0,1.0,1.0,2.121279999
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]
```
```

snort_rag_rule_generator/results/retrieval_backend_proof.json

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

```
FILE: snort_rag_rule_generator/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": ""
    }
  ]
}
```

snort_rag_rule_generator/results/retrieval_benchmark_k3_summary.csv

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

```
FILE: snort_rag_rule_generator/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,,1.0,0.9666666666666668,1.0,0.95,6.750520099922141
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,,1.0,0.9,1.0,0.95,1.8331499999476364
... [truncated to first 10 rows]
```
```

snort_rag_rule_generator/results/retrieval_topk_examples.csv

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

```
FILE: snort_rag_rule_generator/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-P1-0126,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]
```
```

snort_rag_rule_generator/scripts/benchmark_retrieval.py

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

```
FILE: snort_rag_rule_generator/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()
```

snort_rag_rule_generator/scripts/fetch_real_sources.py

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

FILE: snort_rag_rule_generator/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()
```

snort_rag_rule_generator/scripts/plot_retrieval_benchmark.py

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

FILE: snort_rag_rule_generator/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()
```

snort_rag_rule_generator/scripts/validate_dataset.py

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

FILE: snort_rag_rule_generator/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


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())

```
```
```

snort_rag_rule_generator/scripts/validate_rules_with_snort.sh

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

```
FILE: snort_rag_rule_generator/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

```
```

snort_rag_rule_generator/src/snort_rag/__init__.py

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

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

```
```

snort_rag_rule_generator/src/snort_rag/app_gradio.py

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

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

from pathlib import Path
import tempfile

import gradio as gr

from snort_rag.architectures import SnortRAGArchitectures

PROJECT_ROOT = Path(__file__).resolve().parents[2]
DATASET = PROJECT_ROOT / "data" / "processed" / "final_snort_dataset.csv"
rag = SnortRAGArchitectures(DATASET)


def generate(query: str, architecture: str, k: int):
    if not query.strip():
        return "", "", "", ""
    mapping = {
        "Baseline sans RAG": rag.llm_no_rag,
        "RAG classique": rag.rag_classic,
        "RAG + re-ranking": rag.rag_rerank,
        "RAG hybride": rag.rag_hybrid,
        "Multi-hop RAG": rag.multi_hop_rag,
        "Graph RAG": rag.graph_rag,
        "Agentic RAG": rag.agentic_rag,
    }
    fn = mapping[architecture]
    result = fn(query)
    retrieved = "\n".join(
        f"{i+1}. {doc_id} | {atype} | score={score}"
        for i, (doc_id, atype, score) in enumerate(zip(result.get("retrieved_ids", []), result.get("retrieved_attack_types"
    )
    return result["generated_rule"], result["attack_type"], result["explanation"], retrieved


def add_pdf(pdf_file):
    if pdf_file is None:
        return "No PDF uploaded."
    try:
        count = rag.kb.add_pdf_to_kb(pdf_file.name, source_name=Path(pdf_file.name).name)
        return f"Added {count} PDF chunks to the in-memory knowledge base."
    except Exception as exc:
        return f"PDF import failed: {exc}"


def dataset_stats():
    df = rag.kb.df
    counts = df["attack_type"].value_counts().to_string()
    return f"Rows: {len(df)}\n\nAttack type counts:\n{counts}"


with gr.Blocks(title="Snort RAG Rule Generator") as demo:
    gr.Markdown("# Snort RAG Rule Generator\nDefensive NLP/RAG system for generating Snort rules from natural-language atta
```

```
    with gr.Row():
        with gr.Column(scale=2):
            query = gr.Textbox(label="Attack description", lines=4, placeholder="Detect SQL injection with UNION SELECT in
            architecture = gr.Dropdown(
                ["Baseline sans RAG", "RAG classique", "RAG + re-ranking", "RAG hybride", "Multi-hop RAG", "Graph RAG", "Ag
                value="Agentic RAG",
                label="Architecture"
            )
            k = gr.Slider(2, 10, value=5, step=1, label="Top-k retrieval")
            btn = gr.Button("Generate rule")
        with gr.Column(scale=1):
            pdf = gr.File(label="Ajouter un PDF à la base de connaissance", file_types=[".pdf"])
            add_btn = gr.Button("Index uploaded PDF")
            pdf_status = gr.Textbox(label="PDF status")
            stats_btn = gr.Button("Dataset stats")
            stats = gr.Textbox(label="Stats", lines=10)
    rule = gr.Textbox(label="Generated Snort rule", lines=5)
    attack_type = gr.Textbox(label="Detected attack type")
    explanation = gr.Textbox(label="Explanation", lines=4)
    retrieved = gr.Textbox(label="Retrieved documents", lines=8)
    btn.click(generate, inputs=[query, architecture, k], outputs=[rule, attack_type, explanation, retrieved])
    add_btn.click(add_pdf, inputs=[pdf], outputs=[pdf_status])
    stats_btn.click(dataset_stats, outputs=[stats])

if __name__ == "__main__":
    demo.launch()

```
```
```

snort_rag_rule_generator/src/snort_rag/architectures.py

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

```
FILE: snort_rag_rule_generator/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 generate_from_context
from snort_rag.retrieval import RetrievedDoc, SnortKnowledgeBase
from snort_rag.templates import ATTACK_KEYWORDS, detect_attack_type, generate_snort_rule
from snort_rag.rule_parser import validate_rule, option_coverage


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
        result["retrieved_ids"] = [d.id for d in docs]
        result["retrieved_attack_types"] = [d.attack_type for d in docs]
        result["retrieval_scores"] = [round(d.score, 4) for d in docs]
        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"
            valid, errors = False, ["Benign request"]
        else:
            rule = generate_snort_rule(attack_type, query)
            valid, errors = validate_rule(rule)
        result = {
            "query": query,
            "attack_type": attack_type,
            "generated_rule": rule,
            "valid_rule": valid,
            "validation_errors": errors,
            "option_coverage": option_coverage(rule) if rule != "NO_RULE_RECOMMENDED" else 0.0,
            "explanation": "Baseline without retrieval: generated only from query keywords, so context control is weak.",
            "prompt": f"User request only: {query}",
            "hallucination_risk": 0.35 if attack_type != "benign" else 0.20,
        }
```
```

```
        return self._pack("baseline_no_rag", result, [])

    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)
        inferred = first[0].attack_type if first else detect_attack_type(query)
        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 = first[0].attack_type if first else attack_type
            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 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),
        }
```

snort_rag_rule_generator/src/snort_rag/evaluation.py

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

```
FILE: snort_rag_rule_generator/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()

```

snort_rag_rule_generator/src/snort_rag/generate_dataset.py

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

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

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()

```
```

snort_rag_rule_generator/src/snort_rag/generator.py

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

```
FILE: snort_rag_rule_generator/src/snort_rag/generator.py
```python
"""Local generation module for Snort rules.

This is a controlled local generator that behaves like the generation stage of RAG,
but without prohibited external LLM APIs. It builds a transparent prompt, uses
retrieved context, classifies the attack type, fills a valid Snort template and
returns an explanation.
```

```python
"""
from __future__ import annotations

from typing import Dict, List, Sequence

from snort_rag.rule_parser import validate_rule, option_coverage
from snort_rag.templates import CLASSTYPE, detect_attack_type, generate_snort_rule
from snort_rag.retrieval import RetrievedDoc


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 != "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(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)
    return (
        f"Attack type: {attack_type}. The rule uses a conservative classtype "
        f"({CLASSTYPE.get(attack_type, 'unknown')}) 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":
            valid, _ = validate_rule(doc.rule)
            if valid:
                return doc.rule
    return generate_snort_rule(attack_type, query)



def generate_from_context(
    query: str,
    retrieved_docs: Sequence[RetrievedDoc],
    force_attack_type: str | None = None
) -> Dict[str, object]:
    attack_type = force_attack_type or choose_attack_type(query, retrieved_docs)

    if attack_type in {"benign", "benign_traffic"}:
        rule = "NO_RULE_RECOMMENDED"
        valid = True
        errors = []
    else:
        rule = choose_rule(query, attack_type, retrieved_docs)
        valid, errors = validate_rule(rule)

    prompt = build_prompt(query, retrieved_docs)

    hallucination_risk = 0.0

    if not retrieved_docs:
        hallucination_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:
        hallucination_risk += 0.3
```

```
    if not valid and rule != "NO_RULE_RECOMMENDED":
        hallucination_risk += 0.4

    if attack_type == "suspicious_user_agent" and "user" not in query.lower() and "agent" not in query.lower():
        hallucination_risk += 0.2

    return {
        "query": query,
        "attack_type": attack_type,
        "generated_rule": rule,
        "valid_rule": valid,
        "validation_errors": errors,
        "option_coverage": option_coverage(rule) if rule != "NO_RULE_RECOMMENDED" else 0.0,
        "explanation": explain_rule(rule, attack_type, retrieved_docs),
        "prompt": prompt,
        "hallucination_risk": min(1.0, hallucination_risk),
        "retrieved_ids": [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],
    }
```

snort_rag_rule_generator/src/snort_rag/knowledge_base.py

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

```
FILE: snort_rag_rule_generator/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))

```

snort_rag_rule_generator/src/snort_rag/retrieval.py

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

```
FILE: snort_rag_rule_generator/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
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

try:  # pragma: no cover
    import faiss
except Exception:  # pragma: no cover
    faiss = None

try:  # pragma: no cover
    from sentence_transformers import SentenceTransformer
except Exception:  # pragma: no cover
    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 SentenceTransformer is not None

    @staticmethod
    def faiss_runtime_available() -> bool:
        return 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:
        if SentenceTransformer is None:
            raise RuntimeError("sentence-transformers is not installed")
        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:
        if faiss is None:
            raise RuntimeError("faiss is not installed")
        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)

```
```

snort_rag_rule_generator/src/snort_rag/rule_parser.py

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

```
FILE: snort_rag_rule_generator/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"}


@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]


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 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)
    return parsed, raw


def parse_rule(rule: str) -> ParsedRule:
    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 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

```

snort_rag_rule_generator/src/snort_rag/run_devoir3.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator_new/snort_rag_rule_generator/src/snort_rag/run_devoir3.py

```
FILE: snort_rag_rule_generator/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()

```
```

snort_rag_rule_generator/src/snort_rag/source_manifest.py

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

```
FILE: snort_rag_rule_generator/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}

```
```
```

snort_rag_rule_generator/src/snort_rag/templates.py

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

```
FILE: snort_rag_rule_generator/src/snort_rag/templates.py
```python
"""Deterministic Snort rule templates for defensive generation.

These templates are intentionally conservative and are used as the local generation
module in the Devoir 3 implementation because the assignment explicitly forbids
OpenAI/Mistral/Ollama API usage for the TP. The templates are filled only after
retrieval, so generation is not a black-box response.
"""
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", "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()
    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};)')

```
```

snort_rag_rule_generator/tests/test_generate_dataset.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator_new/snort_rag_rule_generator/tests/test_generate_dataset.py

```
FILE: snort_rag_rule_generator/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}

```
```
```

snort_rag_rule_generator/tests/test_retrieval.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator_new/snort_rag_rule_generator/tests/test_retrieval.py

```
FILE: snort_rag_rule_generator/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"]

```

snort_rag_rule_generator/tests/test_rule_parser.py

Type: text | Source: /mnt/e/projects/snort_rag_rule_generator_new/snort_rag_rule_generator/tests/test_rule_parser.py

```
FILE: snort_rag_rule_generator/tests/test_rule_parser.py
```python
from snort_rag.rule_parser import 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

```
```

snort_rag_rule_generator/tests/pcaps/README.md

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

```
FILE: snort_rag_rule_generator/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.

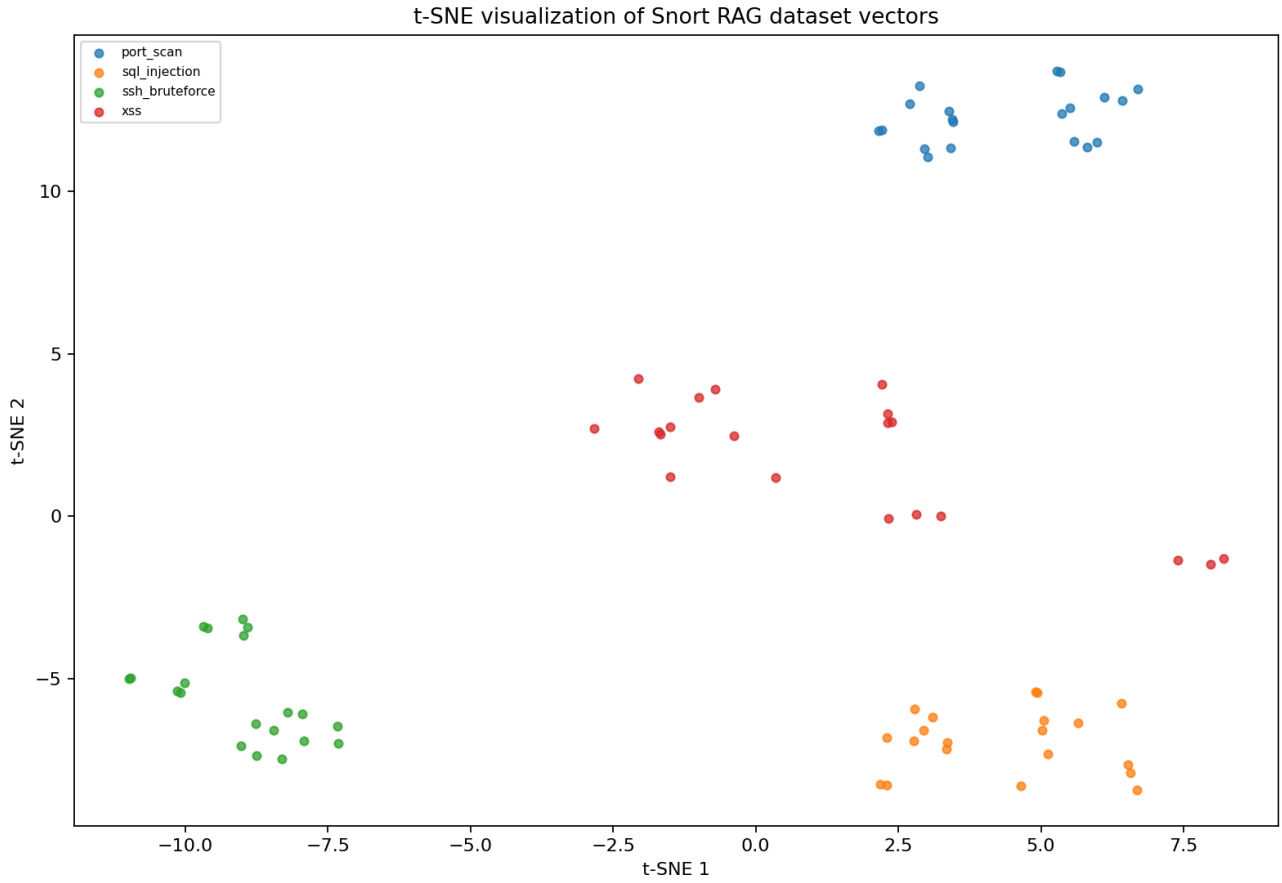
```
```

3. Images

snort_rag_rule_generator/results/embedding_tsne.png

Type: image | Source: /mnt/e/projects/snort_rag_rule_generator_new/snort_rag_rule_generator/results/embedding_tsne.png

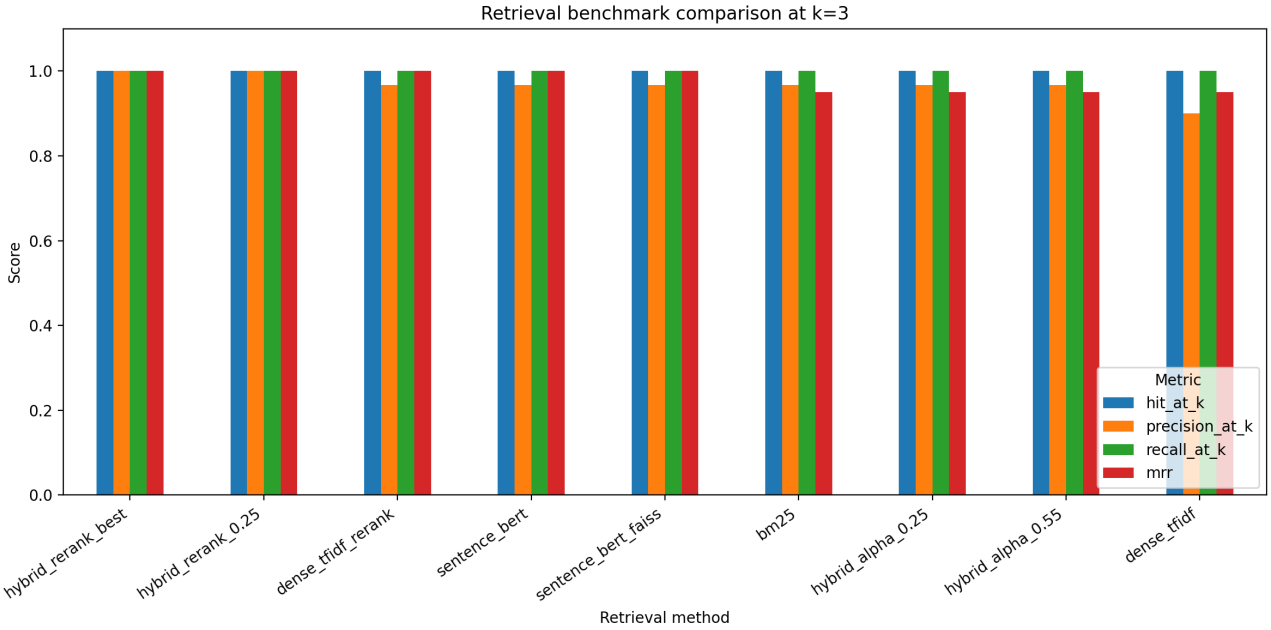
FILE: snort_rag_rule_generator/results/embedding_tsne.png (embedded preview)



snort_rag_rule_generator/results/retrieval_benchmark_k3.png

Type: image | Source: /mnt/e/projects/snort_rag_rule_generator_new/snort_rag_rule_generator/results/retrieval_benchmark_k3.png

FILE: snort_rag_rule_generator/results/retrieval_benchmark_k3.png (embedded preview)



4. Skipped Binary Files

```
- snort_rag_rule_generator/most_important_file.pdf [application/pdf]
- snort_rag_rule_generator/docs/rapport_snort_rag_devoir3.docx [application/vnd.openxmlformats-officedocument.wordprocessin
- snort_rag_rule_generator/docs/rapport_snort_rag_devoir3.pdf [application/pdf]
- snort_rag_rule_generator/docs/reference_new_version/snort_rag_rule_generator_new.pdf [application/pdf]
```

5. Export Summary

```
- Included files count: 40
- Skipped files count: 214
- Skipped binary files: 4
- Skipped large files: 0
- Warning: no image embed failures
```