Protocol: Ablation (DSL vs Python API) and Extensibility Study

Scope: This protocol describes how participants will

- (i) implement two new AI-specific code-smell rules using the **SpecDetect4AI DSL**
- (ii) re-implement the same rules using a **pure Python API** that exposes the same predicates as regular functions. It also specifies the metrics, data collection, and packaging required for reproducibility.

Repository Layout & Pre-requisites :

Download the repo **:**

https://anonymous.4open.science/r/SpecDetect4AI-5DCD/README.md

Requirements :

- Python 3.9+
- Lark 1.2.2

```
python -m pip install -r requirements.txt
```

Structure of the repo :

```
/docs # all the documentation
/static # images of the repo
/DSL_Compare # Comparison between other DSL (Semgrep and CodeQL)
/RAG # Other project with (LLM) RAG to create new rules
/Evaluation # Evaluation folder with ground truth and mlflow
comparison
/grammar # grammar of the DSL in .lark
/parser # DSL parser in .py
/test_rules # all the rules with all the files needed to test each
     /RX # all the rules X between 1 and 22
        /generated_rules_RX.py # the matcher generated
        /test_RX.py # the test file of the rule
        /test_RX.dsl # the rule write in DSL Format
```

```
specDetect4ai.py  # the tool
```

Study Scenarios (Implement new detection rules)

We evaluate two **representative** smells from recent literature. Each participant implements both smells in **two modalities**: **DSL** and **directly in Python**.

Scénario A (Using existing predicates only for the DSL modality)

Name of the code smell : *Index Column Not Explicitly Set in DataFrame Read*

Ref : *Moshi Wei, Nima Shiri Harzevili, Yuekai Huang, Jinqiu Yang, Junjie Wang, and Song Wang. 2024. Demystifying and Detecting Misuses of Deep Learning APIs. In Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (Lisbon, Portugal) (ICSE '24). Association for Computing Machinery, New York, NY, USA, Article 201, 12 pages. https://doi.org/10.1145/3597503.3639177*

Intent : Flag calls that load a pandas DataFrame without setting an explicit index, which may cause downstream misalignment (joins/plots). For the DSL rule, please use this folder to see which predicates coils be reused :

```
/docs/predicates.md
```

Primary patterns (non-exhaustive):

- `pandas.read_csv(...)` **without** `index_col=...` (or explicitly `index_col=None`)
- `pandas.read_excel(...)` **without** `index_col`

- Potentially `pd.DataFrame(data)` where the caller later relies on implicit positional index (Scenario A keeps it simple: focus on *read* functions).

Positive examples (should be detected):

```
import pandas as pd
df = pd.read_csv("data.csv")  # no index_col argument
```

```
import pandas as pd
df = pd.read_excel("data.xlsx", sheet_name=0)  # no index_col
```

Negative examples (should NOT be detected):

```
df = pd.read_csv("data.csv", index_col="id")
```

```
df = (pd.read_excel("data.xlsx", sheet_name="Sheet1")
        .set_index("id"))  # explicit index set immediately after
```

Scenario A constraint: implement using existing predicates only (no parser/DSL extension).

We evaluate two **representative** smells from recent literature. Each participant implements both smells in **two modalities**: **DSL** and **directly in Python**.

Scénario B (Create a new predicate)

Follow this folder to have the explanation of how add a new predicate :

```
/docs/add_rule.md
```

Name of the code smell : *EarlyStopping Not Used in* `model.fit`

Ref : *Yao Yao, Lei Zhang, and Abhimanyu Chaudhuri. 2008. Early Stopping and Its Applications to Boosting. Journal of Machine Learning Research 9 (2008), 947–970.*

Intent : In Keras-like training, warn when `model.fit(...)` is called **without** `EarlyStopping` callback, potentially increasing overfitting risk.

Primary patterns (non-exhaustive):

- **`model.fit(X, y, ...)` without** `callbacks=[..., EarlyStopping(...), ...]`
- `callbacks` present but **no** `EarlyStopping` instance
- (Optional nuance) `EarlyStopping` imported indirectly or aliased.

Positive examples (should be detected):

```
model = build_model()
history = model.fit(X_train, y_train, epochs=50)  # no callbacks
```

```
from keras.callbacks import ReduceLROnPlateau
model.fit(X, y, callbacks=[ReduceLROnPlateau()])  # no EarlyStopping
```

Negative examples (should NOT be detected):

```
from keras.callbacks import EarlyStopping
es = EarlyStopping(patience=5, restore_best_weights=True)
model.fit(X, y, callbacks=[es])
```

Scenario B constraint: introduce **one new semantic predicate** (and add minimal parser support in the DSL) to conveniently express "*callback list contains EarlyStopping instance*".

Tests

Once the rule is written, run all the tests to be sure your rule is well written and pass all the tests. We provided the test file for each scénarios (`test_R23.py for scenario B and test_R24.py for scenario A`)

go to the folder :

```
/test_rules
```

run :

```
./run_all_tests.sh
```

If the tests pass, job is done, else keep going and try to understand what is the problem (ask me for any questions)

Do the same job, but without using the DSL, but only a pure Python API exposing the same predicates as regular functions.

The predicates functions are available in :

```
/test_rules/RX/generated_rules_RX.py
```

We expected you to create a function using the available predicates and walking in the AST.

To give you an example, here is an example of a Python function for the rule R21 : `Columns and DataType Not Explicitly Set`:

```
def rule_R21(ast_node):
    import ast
    add_parent_info(ast_node)
    #set_deterministic_flag(ast_node)
    # "Columns and DataType Not Explicitly Set"
    variable_ops = gather_scale_sensitive_ops(ast_node)
    scaled_vars = gather_scaled_vars(ast_node)
    problems = {}
    for sub in ast.walk(ast_node):
        if (((isPandasReadCall(sub) and (not hasKeyword(sub, "dtype")))
and (not hasKeyword(sub, "names")))):
            line = getattr(sub, 'lineno', '?')
            if line != '?':
                problems[line] = sub
    for line, node in problems.items():
        report_with_line("Call to a pd.read function without the 'dtype'
parameter; consider specifying dtype to ensure explicit column type
handling at line {lineno}", node)
```

Link to Google Form for metrics evaluation :
https://docs.google.com/forms/d/e/1FAIpQLScwzw18u37K6CZvCMFBjJGH2JGBbTSGav0W26slTVBhf2pqRA/viewform?usp=dialog

Thank you for your help 😀