Information retrieval and Pattern Matching leveraging LLMs

Aless HOSRY, Nicolas HLAD, Benoit VERHAEGHE

Who are we?

Aless Hosry

- Research Engineer at Berger-Levrault
- Lyon, France, 2025 – Present
- R&D in software analysis

Nicolas HLAD

- Research Engineer at Berger-Levrault
- Toulouse, France, 2022– Present
- R&D in software analysis

Benoit Verhaeghe

- Scientific project manager at Berger-Levrault
- Lyon, France, 2021 – Present
- R&D in software analysis, team supervision

Introduction

- Software engineering, especially software maintenance, **require investigating and mastering the codebase**.
  - To retrieve domain specific information, implementation conception, architectural decision and dependencies.
  - Developers relies on Tests, debug feature (breakpoints and logs) and searching the codebase with regex
  - Recently **Large Language Model (LLM) combined with code indexing**, offer **Information Retriveal** tool for developers

- We use **Model Driven Engineering (MDE) to build codebase models**
  - Structural and architectural analyses
  - Within Moose, **Models can be queried**, **visualized** and **explored** to better understand the system.

Our contributions

This paper explore **two ways LLM and MDE can be combined to enhance developers' experiences**.

- To provide more accurate and context-aware responses

LLM
x
Moose/FAMIX

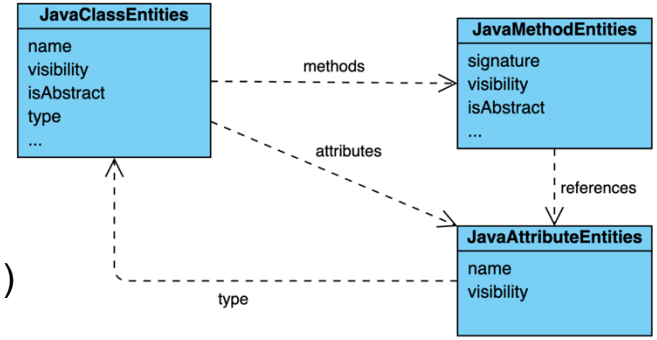
- To enhance the querying and retrieval capabilities of MDE tools

LLM
x
MoTion

GraphRa[illegible]in Moose

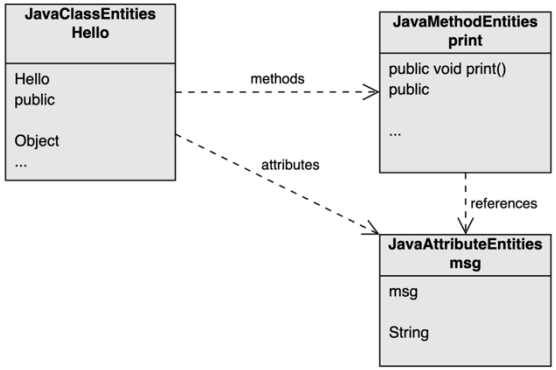
Early Work & Results

Model Driven Engineering



Metamodel (M2)

FAMIX Java



Model (M1)

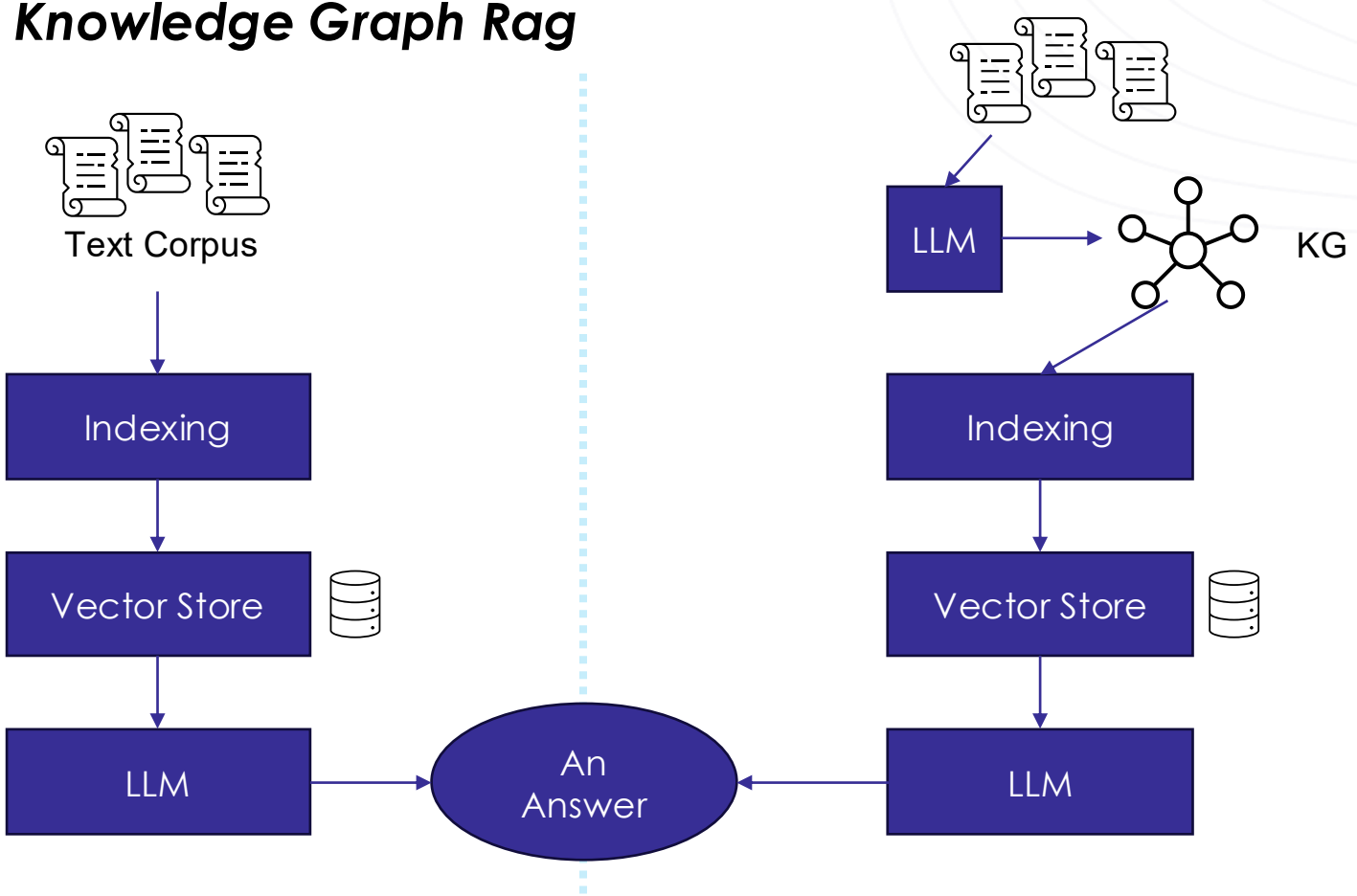
FAMIX Java instance

```
1  public class Hello {
2      String msg = "hello";
3      public void print(){
4          system.out.print(msg);
5      }
6  }
```

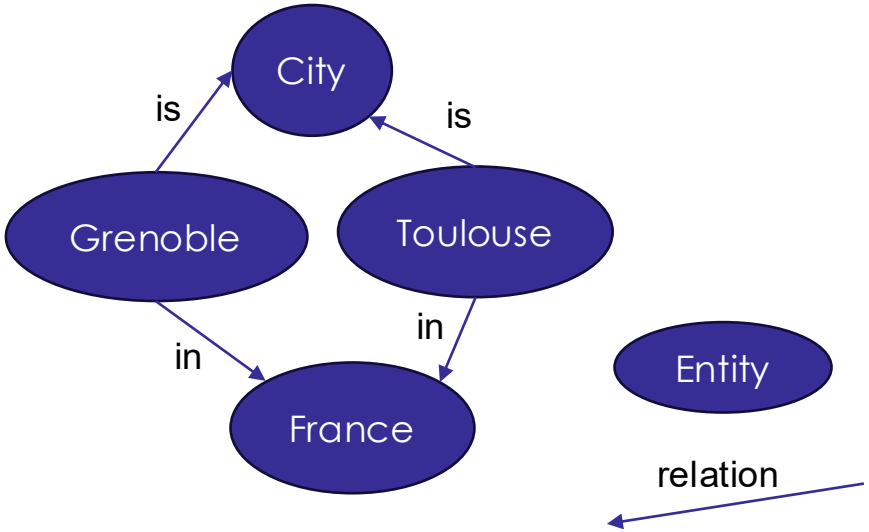
Object (M0)

Java code

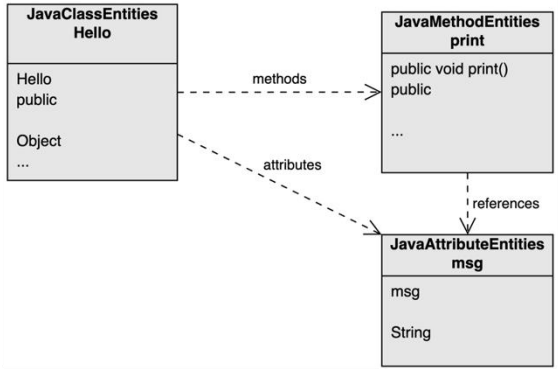
RAG & Knowledge Graph Rag



Knowledge Graphs (KG)



Example of a Knowledge Graph



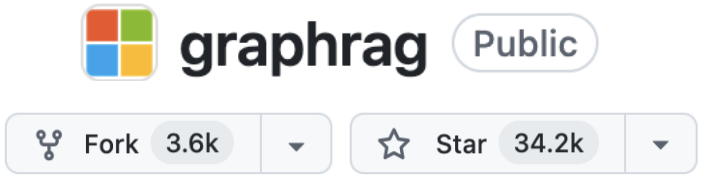
An example of a Famix Model instance

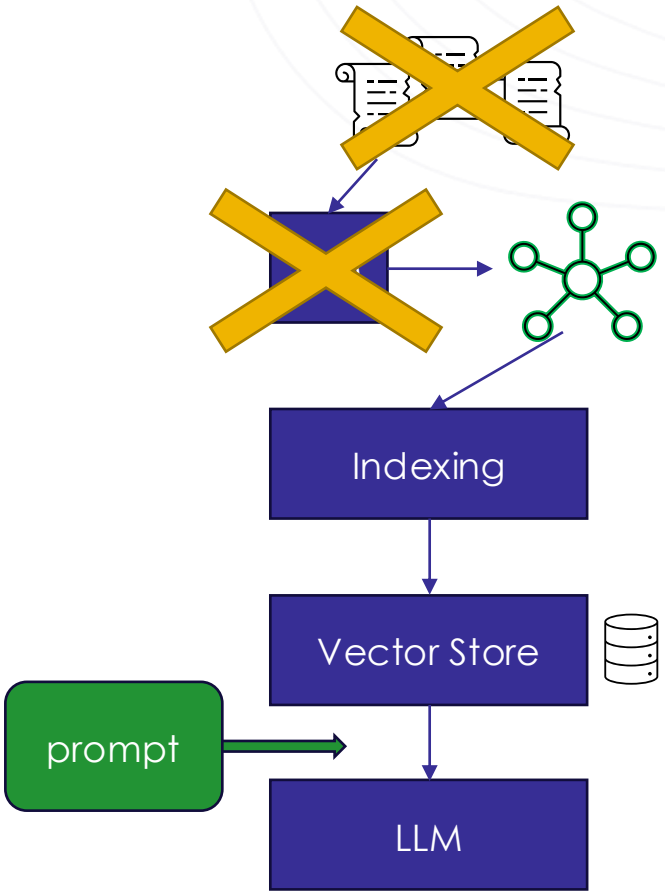
RQ: How can Famix be leverage to build KG and improve GraphRag results ?

Microsoft's GraphRag (3.0.9)

- We use Microsoft's GraphRag
  - Opensource with industrial support
  - Allows import from **an existing Knowledge Graph**
  - Allows Prompt Tuning for indexing and **querying**

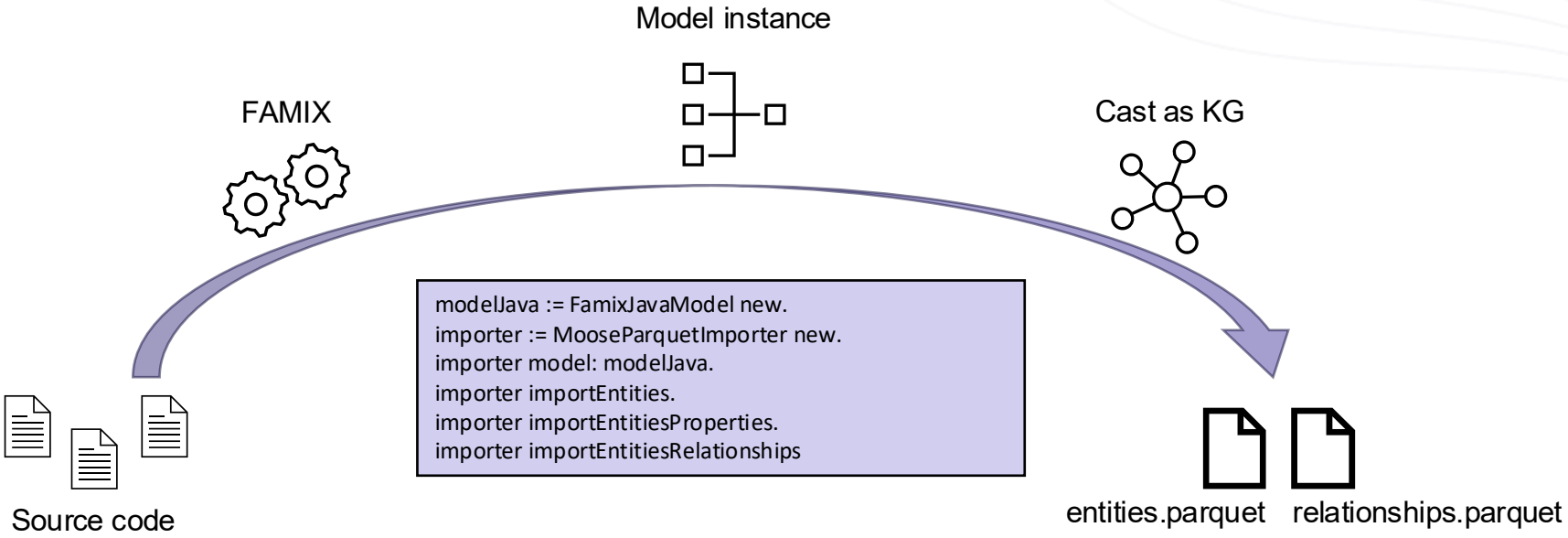
https://github.com/Microsoft/**graphrag**





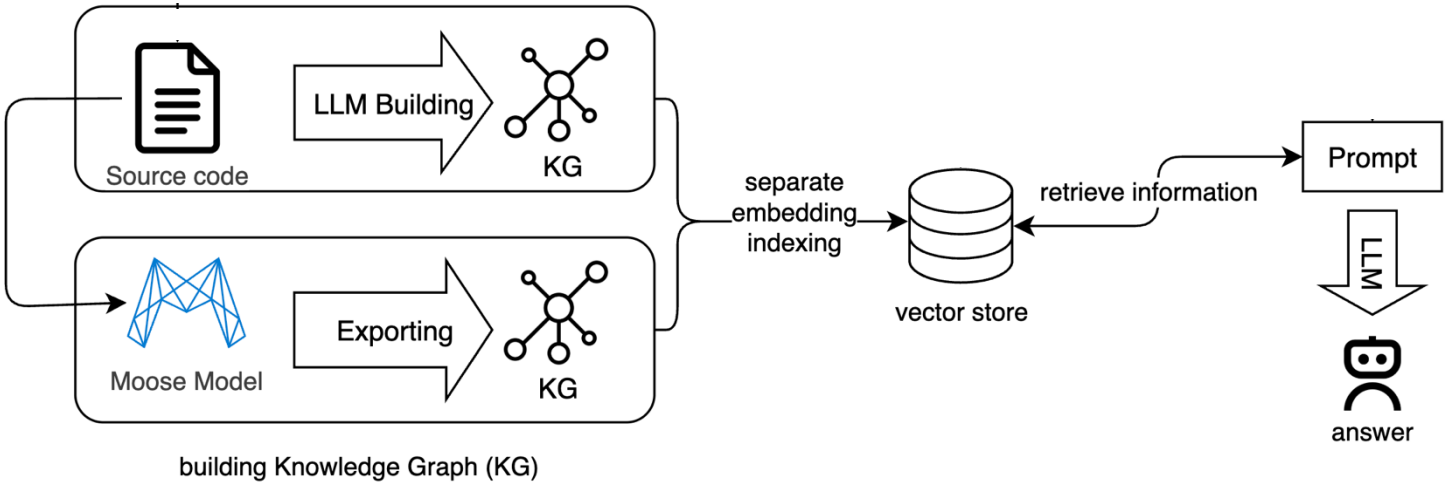
Moose and Famix as Knowledge Graph

https://github.com/Evref-BL/**MooseGraphRAG**



Experimentation setup – 1

Goal 1 : To evaluate the retrieval performance of the source GraphRag powered by Moose/Famix

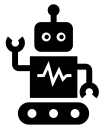


Experimentation setup – 2

Goal 2 : Creating multiple evaluators

Baseline

Random

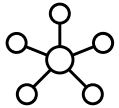


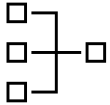
LLM only

Github Copilot

GraphRag system

GraphRag

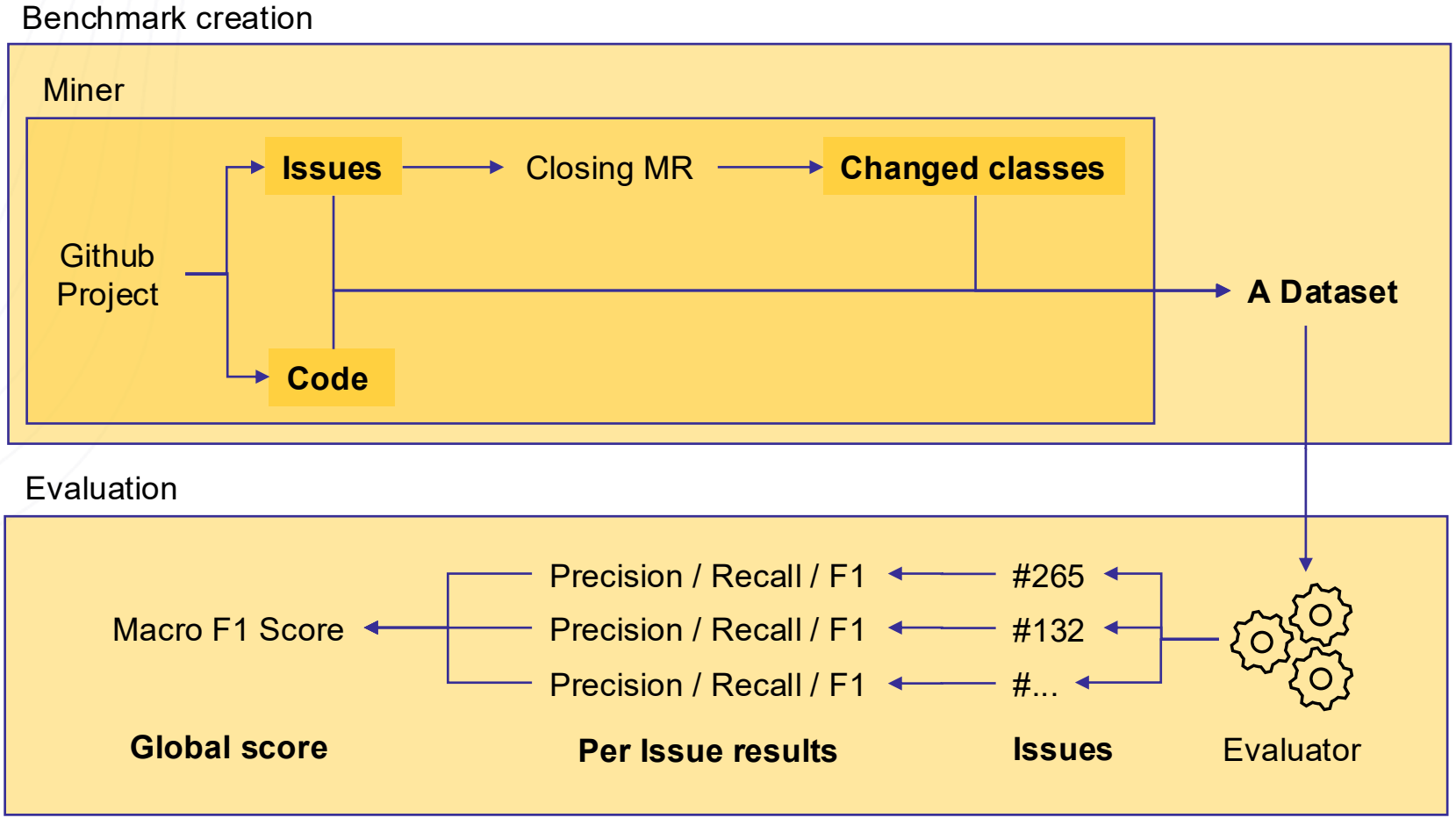




GraphRag x Moose

Goal 3 : Running our system over Small LLM locally

#1 Replication purpose (replaying our experiements)
#2 To reduce the Energy consumption impact of our solution (no tokenmaxxing)
#3 Good results under low resources would probably improve with higher resources


Benchmark creation
Miner
Issues
Closing MR
Changed classes
Github Project
Code
A Dataset
Evaluation
Precision / Recall / F1
Precision / Recall / F1
Precision / Recall / F1
Macro F1 Score
#265
#132
#...
Global score
Per Issue results
Issues
Evaluator

Initial results and discussions - 1

We select a small java project to match the capability of the small LLM

85 files
26898 LoCs
93 classes

- 71 issues extracted
- 65 issues selected
- ⚠ 53 correct issues

Computer : Macbook Pro M4 Max with 64GB of RAM.
LLM provider: local Ollama
Indexing model : nomic-embed-text-v2-moe:latest
Inference model : gemma4:e4b

Initial results and discussions - 2

Table 2
Entity type distribution in the two knowledge graphs

Entity Type	GraphRAG with Moose	GraphRAG without Moose
method	1167	51
attribute	200	0
class	93	40
dependency	0	11
package	0	11
test	0	8
entity	0	4
module	0	3
function	0	3
interface	0	2
other	0	2
Total	**1460**	**135**

Initial results and discussions - 3

Table 1
Results on `stleary/json-java`, on 64 issues. GraphRAG rows compare the base `GraphRAG` project with `GraphRAG-Moose`. The random baseline results are means over 100 iterations.

Mode / baseline	GraphRAG			GraphRAG-Moose			
	P	R	F1	P	R	F1	Δ F1
Average over modes	0.4149	0.2285	0.2676	0.4135	0.2344	0.2732	+0.0056
			baseline				
Random (100 iterations)	0.0467	0.0450	0.0457				
Direct LLM `gemma4:e4b`	0.6094	0.2879	0.3623				
Github Copilot `Agent GPT-5.4`	**0.6805**	**0.6072**	**0.6099**				

Initial results and discussions - 3

- Improvements
  - Fixing dataset collection (e.g., unrelated PR)
  - Fixing .parquet files creation (i.e., missing moose entity description)
  - Customizing KG creation for our benchmark goal (i.e., excluding method entities)
    → 1167 methods vs 93 classes

- And future work
  - Explicability of our results (instrumentation with logs over GraphRag entity extraction)
  - Improvement of our results (better KG from Famix)

MoTion Skills

Definitions: so what is MoTion ?

- MoTion
  - Library for **pattern matching** in Pharo
    - Searching for **tokens** using **specific patterns**
    - Pattern matching is applied over:
      - Objects (for example ASTs)
      - Graphs

Definitions: so what is MoTion ?

- MoTion
  - Library for pattern matching in Pharo
  - It uses a **specific syntax**
    - A **set of operators** is used, each one dedicated for a **specific functionality**

Example 1

```
pattern := OCMethodNode % {
        #'children*' <=> OCMessageNode % {
                #'selector>value' <=> #ifTrue:ifFalse:
        }
}.
```

Definitions: so what is MoTion ?

- MoTion
  - Library for pattern matching in Pharo
  - It uses a **specific syntax**
    - A **set of operators** is used, each one dedicated for a **specific functionality**

Example 2

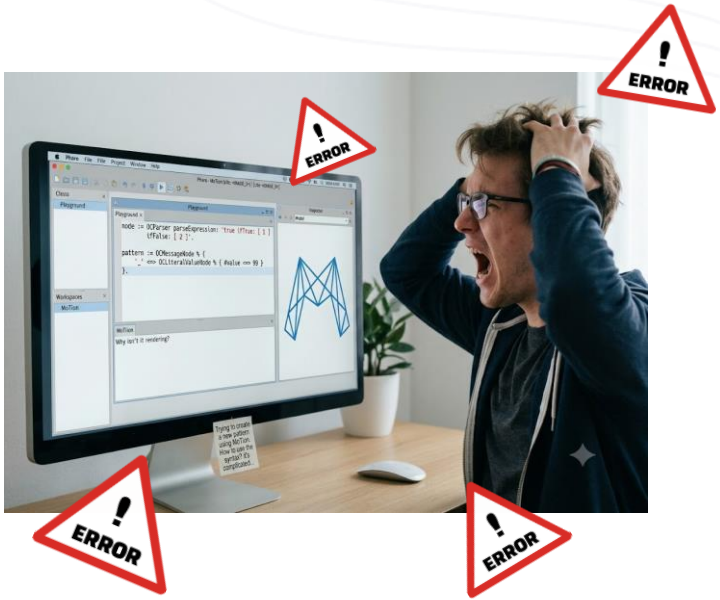
```
pattern := FASTTypeScriptClass % {
    #name <=> #'@aClassName'
    #children' <=> FASTTypeScriptMethod % {
        #name <=> #'@aMethodName'
    }
}.
```

Problem: Syntax complex to learn/use

%	Denotes the **start** of many **class** properties defined by paths and subpatterns
%%	Same as % including research in **sub classes**
<=>	Denotes the **association** of pattern definition in pattern identity
<~=>	Denotes the **negative association** of pattern definition in pattern identity
>	**Path traversal** for nested structures while matching patterns
*property **	**Recursive traversal** for nested structure with ignored depth
_	**Wildcard** for anonymous properties names
@foo	Denotes a **non-linear pattern** that always matches successfully
**L1*	Variable that contains a **List** that can be empty or contains multiple results

Problem: Syntax complex to learn/use

%	Denotes the **start** of many **class** properties defined by paths and subpatterns
%%	Same as % including research in **sub classes**
<=>	Denotes the **association** of pattern definition in pattern identity
<~=>	Denotes the **negative association** of pattern definition in pattern identity
>	**Path traversal** for nested structures while matching patterns
*property **	**Recursive traversal** for nested structure with ignored depth
_	**Wildcard** for anonymous properties names
@foo	Denotes a **non-linear pattern** that always matches successfully
**L1*	Variable that contains a **List** that can be empty or contains multiple results



Proposed solution: Leverage AI to help developers

What if developers can ask any AI model to help them create MoTion patterns following their own description?

<u>For example:</u>

« Create for me a MoTion pattern that searches for ***all Java methods in a Java model having a switch case*** *»*

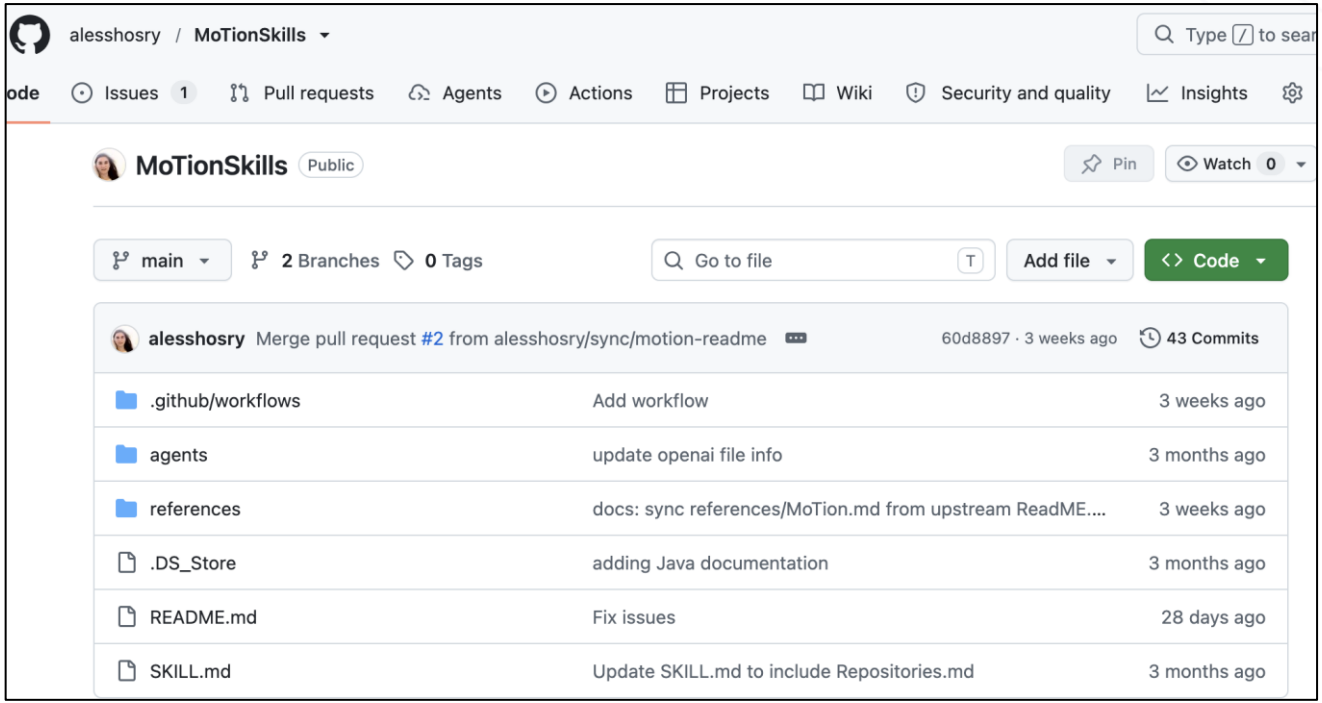
Proposed solution: Leverage AI to help developers

☹ But LLMs don't know MoTion

☺ No problem we can create **skills**:

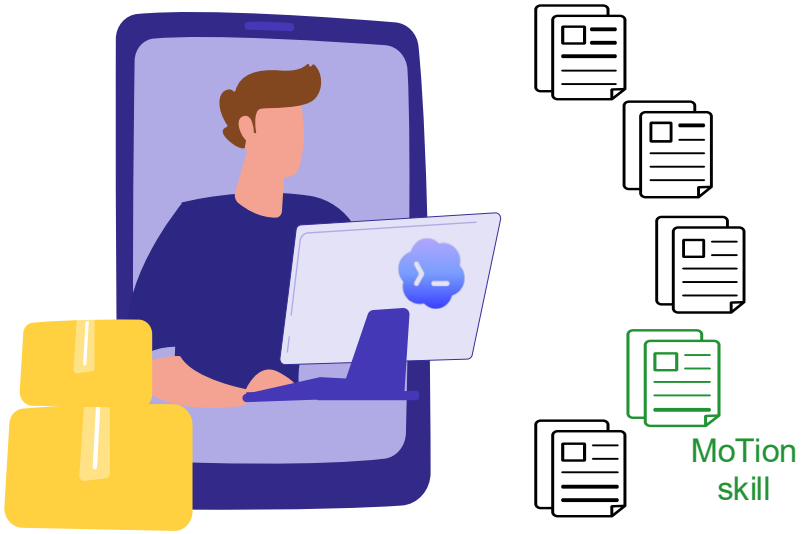
- Concept introduced recently by Anthropic
- A skill allows AI agents to consume new capabilities
- Each skill is organized within a dedicated directory containing:
  - → A SKILL.md file that lists the main documentation
  - → Executable scripts
  - → References materials
  - → Dependency guides

Proposed solution: Leverage AI to help developers



Scan me

So how MoTion Skills work ?



1

Make sure skills are installed on the AI agent (here it's Codex)

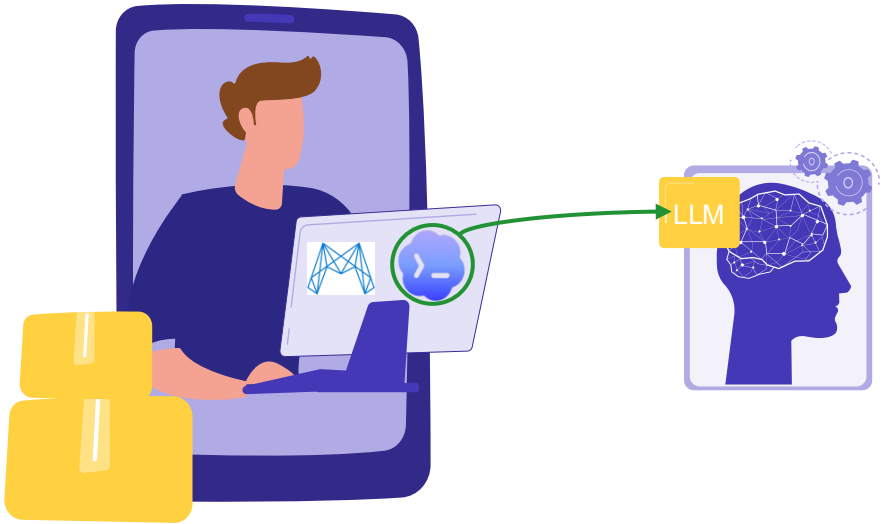
So how MoTion Skills work ?

2

Prepare the prompt for codex, for example:

/MoTion I want to create a MoTion pattern that matches empty Java classes and match it with a FASTJavaModel in Moose.

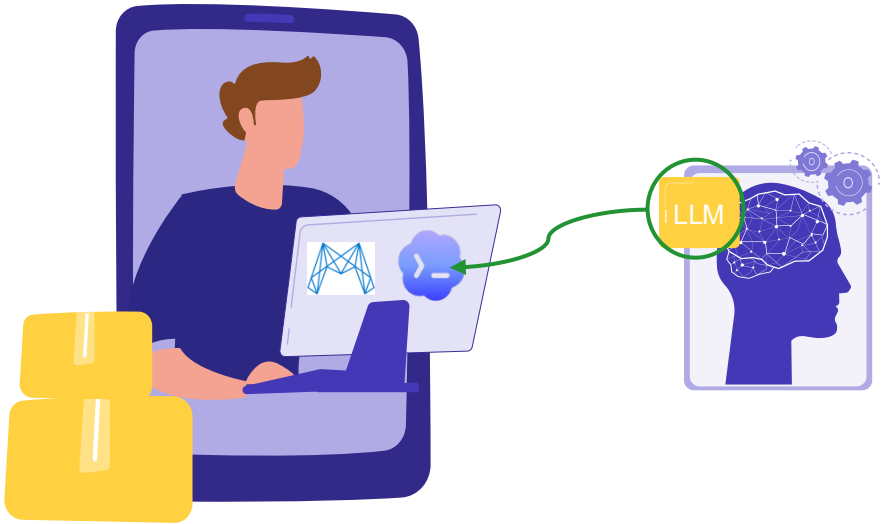
So how MoTion Skills work ?



3

- Analyse request
- Read MoTion skill
- Read appropriate references: MoTion guide + FASTJava Metamodel guide
- Prepare pattern

So how MoTion Skills work ?



3

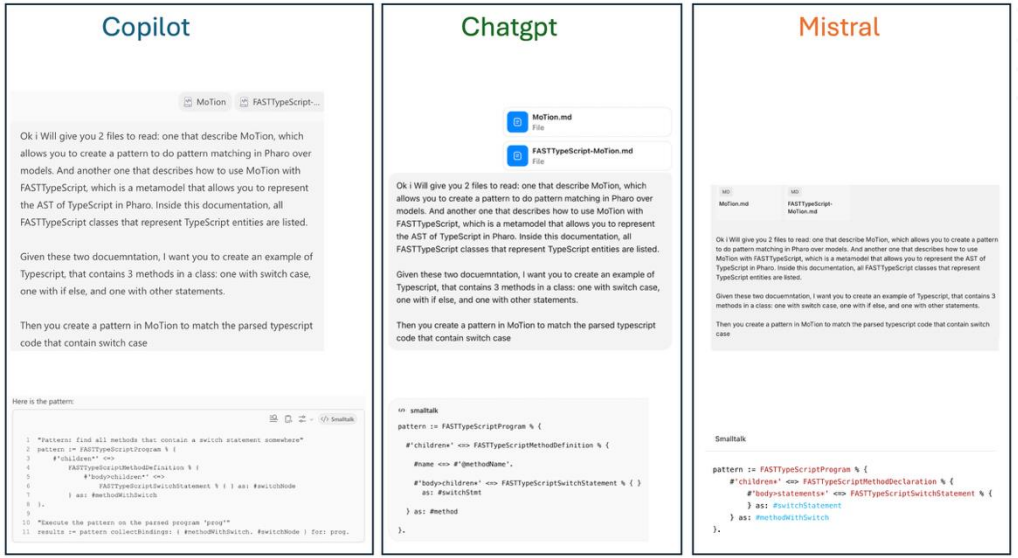
- Analyse request
- Read MoTion skill
- Read appropriate references: MoTion guide + FASTJava Metamodel guide
- Prepare pattern
- Send response back to Codex

2 experiments implemented

	First experiment	Second experiment
Job	Comparison between different LLMs	Evaluating MoTion skill integration with Codex
Objective	Check the skill performance among different LLMs	Check if skill can understand MoTion existing patterns

First experiment: Comparison between different LLMs

- 3 LLMs chosen
- Repository files were provided
- Same prompt applied for all of them
- Several observations conducted
- Observations 👇



First experiment: Observations

Responses		
Copilot	**Chatgpt**	**Mistral**

pattern := FASTTypeScriptProgram % {
#'children*' <=>
FASTTypeScriptMethodDeclaration % {
#'body>children*' <=>
FASTTypeScriptSwitchStatement % {
} as: #switchStatement
} as: #methodWithSwitch
}.
	pattern := FASTTypeScriptProgram % {
	#'children*' <=>
	FASTTypeScriptMethodDeclaration % {
	#'body>children*' <=>
	FASTTypeScriptSwitchStatement % {
	} as: #switchStatement
	} as: #methodWithSwitch
	}.
		pattern := FASTTypeScriptProgram % {
		#'children*' <=>
		FASTTypeScriptMethodDeclaration % {
		#'body>statements*' <=>
		FASTTypeScriptSwitchStatement % {
		} as: #switchStatement
		} as: #methodWithSwitch
		}.

✓ Similar patterns provided

First experiment: Observations

Responses		
Copilot	**Chatgpt**	**Mistral**

pattern := FASTTypeScriptProgram % {
#'children*' <=>
FASTTypeScriptMethodDeclaration % {
#'body>**children***' <=>
FASTTypeScriptSwitchStatement % {
} as: #switchStatement
} as: #methodWithSwitch
}.
	pattern := FASTTypeScriptProgram % {
	#'children*' <=>
	FASTTypeScriptMethodDeclaration % {
	#'body>**children***' <=>
	FASTTypeScriptSwitchStatement % {
	} as: #switchStatement
	} as: #methodWithSwitch
	}.
		pattern := FASTTypeScriptProgram % {
		#'children*' <=>
		FASTTypeScriptMethodDeclaration % {
		#'body>**statements***' <=>
		FASTTypeScriptSwitchStatement % {
		} as: #switchStatement
		} as: #methodWithSwitch
		}.

- ✓ Similar patterns provided
- ✓ Almost correct patterns (except for Mistral with slight issue)

Second experiment: Evaluating MoTion skill integration with Codex

- 25 predefined patterns extracted from a private project at Berger-Levrault (9 matching Java AST and 16 matching TypeScript AST)
- Patterns are constructed by a MoTion expert
- Same prompt applied on Codex for each pattern :
  - « Do you know what does this pattern ? »
- Several observations conducted 👇

Second experiment: Observations

- Each pattern was described in three distinct forms:
  - A brief overview
  - A detailed explanation in plain English
  - A semantic interpretation closely aligned with natural language.
- Some responses included comparative elements
  - Example: ``unlike the previous pattern.''
- TypeScript patterns were written using an outdated naming convention:
  - Reformulation proposed
  - Warnings appeared
  - No suggestion for any change

Second experiment: Observations

- Most responses included **examples**:
  - For TypeScript, the examples often omitted semicolon;
  - A match rate of 95%
- Many responses also included additional notes:
  - Clarifications on what the pattern does not match
- In most cases, the agent suggested simplified versions of the patterns by removing unnecessary parentheses

Conclusion

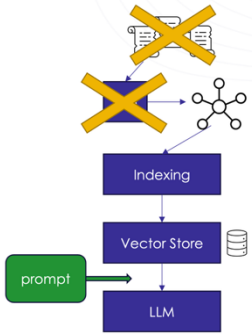
Microsoft's GraphRag (3.0.9)

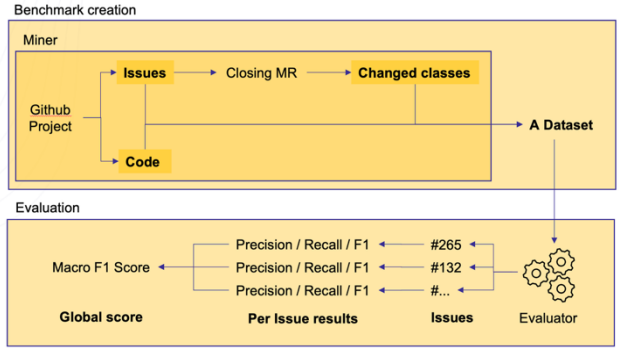
- We use Microsoft's GraphRag
  - Opensource with industrial support
  - Allows import from **an existing Knowledge Graph**
  - Allows Prompt Tuning for indexing and **querying**

https://github.com/Microsoft/**graphrag**

graphrag Public

Fork 3.6k Star 34.2k





Proposed solution: Leverage AI to help developers

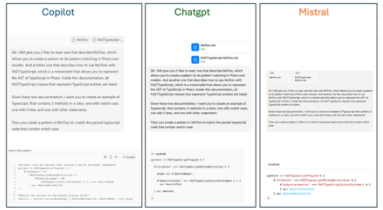
What if developers can ask any AI model to help them create MoTion patterns following their own description?

For example:

*« Create for me a MoTion pattern that searches for **all Java methods in a Java model having a switch case** »*

First experiment: Comparison between different LLMs

- 3 LLMs chosen
- Repository files were provided
- Same prompt applied for all of them
- Several observations conducted
- Observations 👇



Invalide merge requests in dataset

- **641** → google/oss-fuzz#6733
- **678** → synchrony/smsn#68
- **701** → jettison-json/jettison#53
- **706** → twilio/twilio-java#709
- **743** → jettison-json/jettison#53
- **756** → OneSignal/OneSignal-Android-SDK#2004
- **758** → elimuinformatics/a2d2#390, yext/JSON-java#1
- **771** → yext/JSON-java#1
- **789** → github/advisory-database#2856
- **893** → DevLeoko/license-gate-java-wrapper#1
- **902** → clearlydefined/curated-data#26813
- **926** → hiero-ledger/hiero-consensus-node#17264

Integrate fuzzing into JSON-java by way of OSS-fuzz #641

Closed

DavidKorczynski opened on Nov 1, 2021 — Last edited by DavidKorczynski

Hi!

I was wondering if you would be interested in getting JSON-java fuzzed by OSS-Fuzz? Fuzzing is a technique for stress-testing applications and OSS-Fuzz is a free service run by Google for fuzzing important open source projects, and it would be great to get JSON-java integrated with OSS-Fuzz. OSS-Fuzz will run the fuzzers continuously and report back with bug reports when bugs are found. These reports will have full stack traces, input triggers and more. If you are interested, then the only thing needed is an email(s) connected to a Google account that can be used for receiving the bugs reports, which can be put in the configuration file (`project.yaml`) over at the OSS-Fuzz repository.

I have set up an initial integration with OSS-Fuzz here: google/oss-fuzz#6733 and this PR on the OSS-Fuzz repository includes a simple initial fuzzer. We can always extend with more fuzzers if you are happy to integrate. I would just need an email and then get this PR merged, following that we should be good to go and down the line we can also migrate fuzzers from OSS-Fuzz to JSON-java owns repository.

stleary on Nov 4, 2021 — Owner

@DavidKorczynski Sounds interesting. You can use jsonjava060@gmail.com to get started.

DavidKorczynski on Nov 6, 2021 — Author

Thanks **@stleary** - the integration has now completed (google/oss-fuzz#6733). You should receive notifications on your email if any issues arise. Thanks!

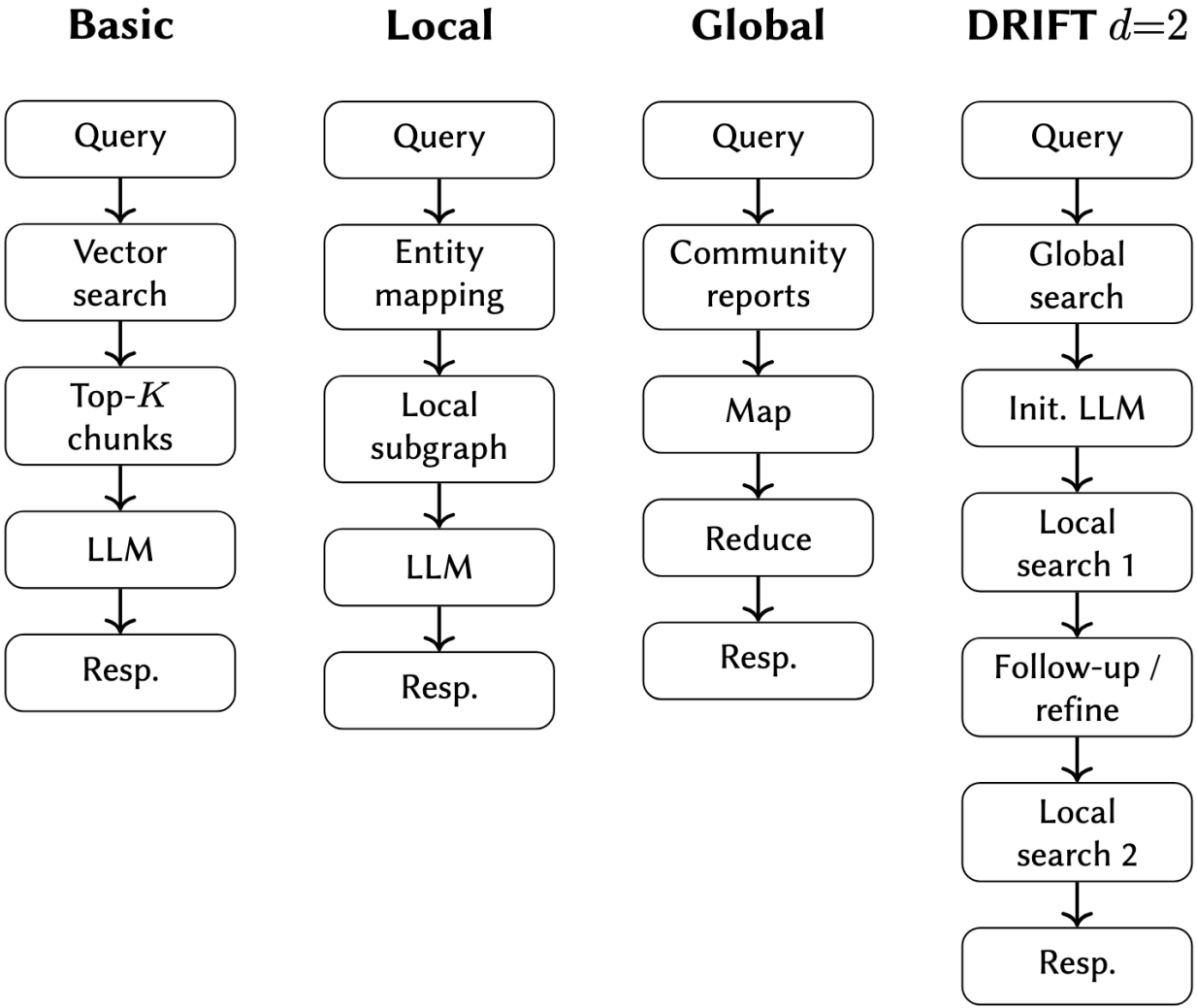
```
{
  "number": 641,
  "title": "Integrate fuzzing into JSON-java by way of OSS-fuzz",
  "url": "https://github.com/stleary/JSON-java/issues/641",
  "description_message": "Hi!\r\n\r\nI was wondering if you would be interested in getting JSON-java fuzzed by OSS-Fuzz? Fuzzing is a technique for stress-testing applications and OSS-Fuzz is a free service run by Google for fuzzing important open source projects, and it would be great to get JSON-java integrated with OSS-Fuzz. OSS-Fuzz will run the fuzzers continuously and report back with bug reports when bugs are found. These reports will have full stack traces, input triggers and more. If you are interested, then the only thing needed is an email(s) connected to a Google account that can be used for receiving the bugs reports, which can be put in the configuration file (`project.yaml`) over at the OSS-Fuzz repository.\r\n\r\nI have set up an initial integration with OSS-Fuzz here: https://github.com/google/oss-fuzz/pull/6733 and this PR on the OSS-Fuzz repository includes a simple initial fuzzer. We can always extend with more fuzzers if you are happy to integrate. I would just need an email and then get this PR merged, following that we should be good to go and down the line we can also migrate fuzzers from OSS-Fuzz to JSON-java owns repository.\r\n",
  "created_at": "2021-11-01T21:07:42Z",
  "closed_at": "2021-11-06T13:27:28Z",
  "linked_merged_pull_request_urls": [
    "https://github.com/google/oss-fuzz/pull/6733",
    "https://github.com/stleary/JSON-java/pull/608"
  ],
  "linked_merged_pull_requests": [
    {
      "number": 6733,
      "title": "json-java: initial integration",
      "url": "https://github.com/google/oss-fuzz/pull/6733",
      "repository": "google/oss-fuzz",
      "merged_at": "2021-11-06T09:30:32Z",
      "impacted_files": [
        "projects/json-java/Dockerfile",
        "projects/json-java/JsonJavaFuzzer.java",
        "projects/json-java/build.sh",
        "projects/json-java/project.yaml"
      ],
      "impacted_files_count": 4
    },
```

- **#641** (`google/oss-fuzz#6733`) — mentioned in **issue body** and in comment: https://github.com/stleary/JSON-java/issues/641#issuecomment-962451851
- **#678** (`synchrony/smsn#68`) — appears as **timeline cross-reference** on issue page
- **#701** (`jettison-json/jettison#53`) — **not currently visible** in body/comments/timeline
- **#706** (`twilio/twilio-java#709`) — **not currently visible** in body/comments/timeline
- **#743** (`jettison-json/jettison#53`) — mentioned in **issue body**
- **#756** (`OneSignal/OneSignal-Android-SDK#2004`) — appears as **timeline cross-reference**
- **#758** (`elimuinformatics/a2d2#390`, `yext/JSON-java#1`) — both appear as **timeline cross-references**
- **#771** (`yext/JSON-java#1`) — appears as **timeline cross-reference**
- **#789** (`github/advisory-database#2856`) — appears as **timeline cross-reference**
- **#893** (`DevLeoko/license-gate-java-wrapper#1`) — appears as **timeline cross-reference**
- **#902** (`clearlydefined/curated-data#26813`) — mentioned in comment: https://github.com/stleary/JSON-java/issues/902#issuecomment-2740623145
- **#926** (`hiero-ledger/hiero-consensus-node#17264`) — appears as **timeline cross-reference**

The “timeline cross-reference” entries are shown in the issue conversation timeline (not always as plain URL text in body/comments).

Table 3
Relationship type distribution in the two knowledge graphs

Relationship Type	GraphRAG with Moose	GraphRAG without Moose
invokes	4372	0
calls	0	4
depends_on	0	10
caught_exceptions	208	0
declared_exceptions	196	0
thrown_exceptions	105	0
defines	0	2
extends	0	2
reads_from	0	1
imports	0	1
unknown	73	8
Total	**4954**	**28**


Basic
Query
Vector search
Top-K chunks
LLM
Resp.
Local
Query
Entity mapping
Local subgraph
LLM
Resp.
Global
Query
Community reports
Map
Reduce
Resp.
DRIFT d=2
Query
Global search
Init. LLM
Local search 1
Follow-up / refine
Local search 2
Resp.

Initial results and discussions - 3

- Improvements
  - Fixing dataset collection (e.g., unrelated PR)
  - Fixing .parquet file creation (i.e., missing moose entity description)
  - Customizing KG creation for our benchmark goal (i.e., excluding method entities)
    → 1167 methods vs 93 classes

- And future work
  - Explicability of our results (instrumentation with logs over GraphRag entity extraction)
  - Improvement of our results (better KG from Famix)

GraphRag Community with Hierarchical Leiden algorithm

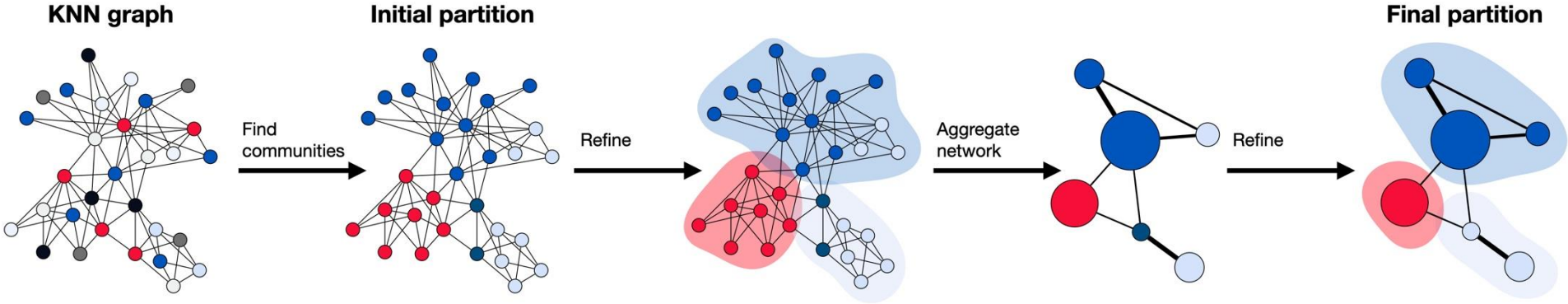


Figure from https://www.sc-best-practices.org/cellular_structure/clustering.html

response type: "{\"classes\":[{\"path\":\"<java_file_path>.java\"}]}. No markdown, no explanation, no extra keys."

Required pre prompt: MANDATORY OUTPUT FORMAT: Follow STRICTLY the *response type* specification below: {\"classes\":[{\"path\":\"<java_file_path>.java\"}]}. No markdown, no explanation, no extra keys. No markdown and no extra text outside this format."

Extra prompt: "Identify the Java class file paths impacted by the issue resolution. Use only paths ending with .java in the required JSON schema."

query:
Issue title: JSONPointer: Encoding quotes not required?
Issue description:
JSONPointer encodes quotes as \\\\\"
https://github.com/stleary/JSON-java/blob/7abd89b4bcc321152c178af5504da7ea4ac66693/src/main/java/org/json/JSONPointer.java#L271
However I can't find a requirement for that in https://tools.ietf.org/html/rfc6901#section-3 or elsewhere.
It causes a conflict in an application that expects the canonical form of the URL not to be escaped in this way. For use in fragment identifiers RFC 3986 allows %22 to encode \".