Together We Are Better

LLM, IDE and Semantic Embedding to Assist Move Method Refactoring

Abhiram Bellur University of Colorado Boulder
Malinda Dilhara Amazon Web Services
Yaroslav Zharov JetBrains Research
Kai Ishikawa NEC Corporation
Masaharu Morimoto NEC Corporation
Takeo Hosomi NEC Corporation
Hridesh Rajan Tulane University
Fraol Batole Tulane University
Mohammed Raihan Ullah University of Colorado Boulder
Timofey Bryksin JetBrains Research
Haifeng Chen NEC Laboratories America
Shota Motoura NEC Corporation
Tien N. Nguyen University of Texas at Dallas
Nikolaos Tsantalis Concordia University

Danny Dig University of Colorado Boulder, JetBrains Research

Why MoveMethod Matters

Top-5 most common refactoring

Improves cohesion, reduces coupling

Reduces Technical Debt and removes code smells: God Class, Feature Envy, Duplicate Code

MoveMethod Refactoring to the Rescue

ElasticSearch (876e7015)

```java
public class EsqlSession {
  private PolicyResolver policyResolver;
  ...
  public void execute(EsqlQueryRequest request, ...){
    LOGGER.debug("ESQL query:\n{}", request.query()); ...}
  private LogicalPlan parse(String query, ...) {...}
  public void analyzedPlan(...) {...}
  public void optimizedPlan(...) {...}

  private void preAnalyze(...) {
      ...
    resolvePolicy(groupedListener, policyNames, resolution);
  }
  ...
}
```

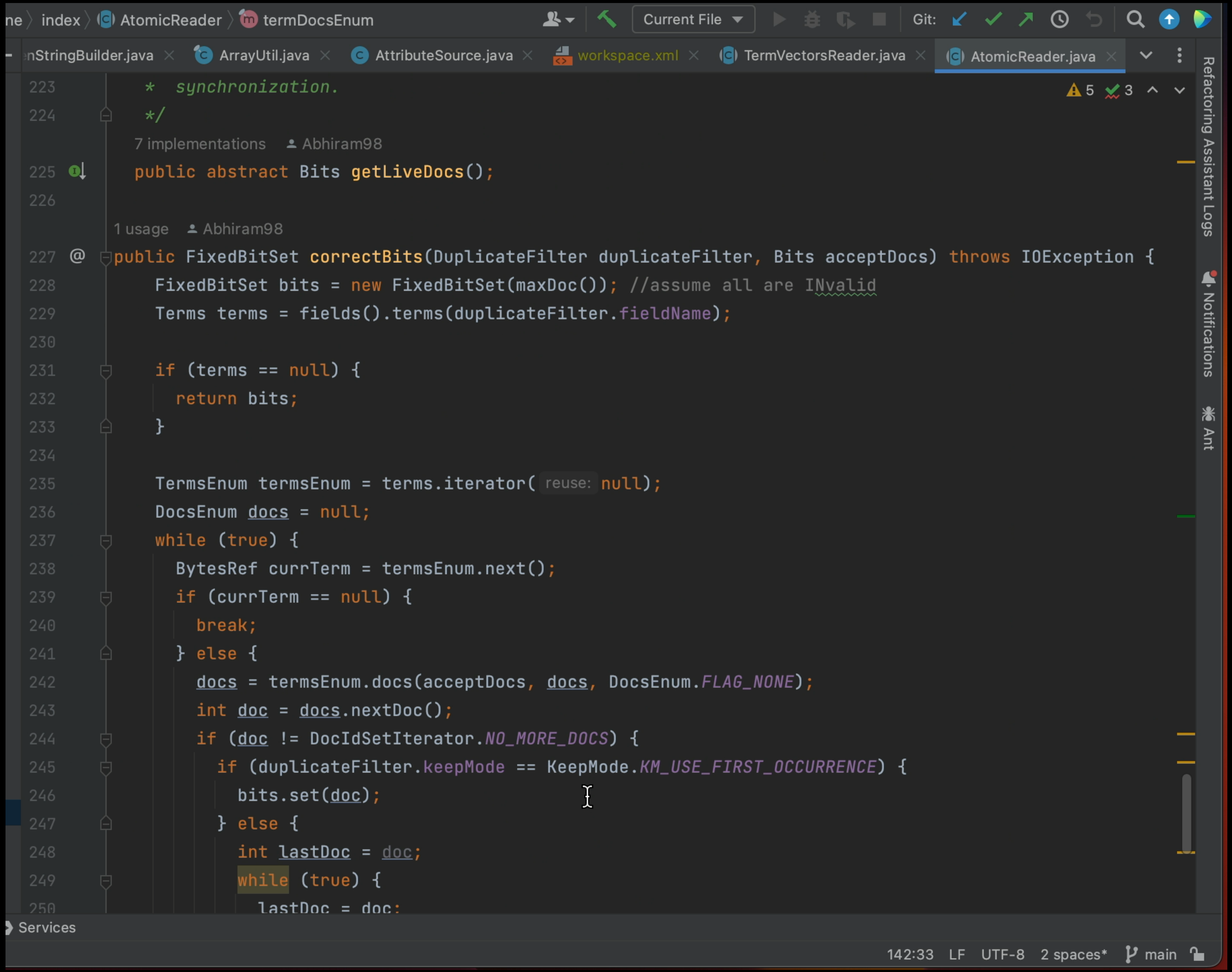
```java
policyResolver.resolvePolicy(...);
```

```java
/* Resolves a set of policies and adds them to
a given resolution.*/
private void resolvePolicy(
  ActionListener groupedListener,
  Set policyNames,
  Resolution resolution) {
    ...
    for (policyName : policyNames) {
      this.resolvePolicy(
        policyName,
        resolution.resolvedPolicies()::add)
    );
```

PolicyResolver

Current Move Method Workflow in IntelliJ

- JetBrains' IntelliJ IDEA has Move Method capabilities
- Semi-automated process
- No automatic recommendations



Approaches for MM Recommendations

Static analysis (JMove, JDeodorant)
- thresholds, slow (hours), poor scalability

ML (RMove, PathMove) / DL (FeTruth, Hmove)
- need retraining, overwhelm users

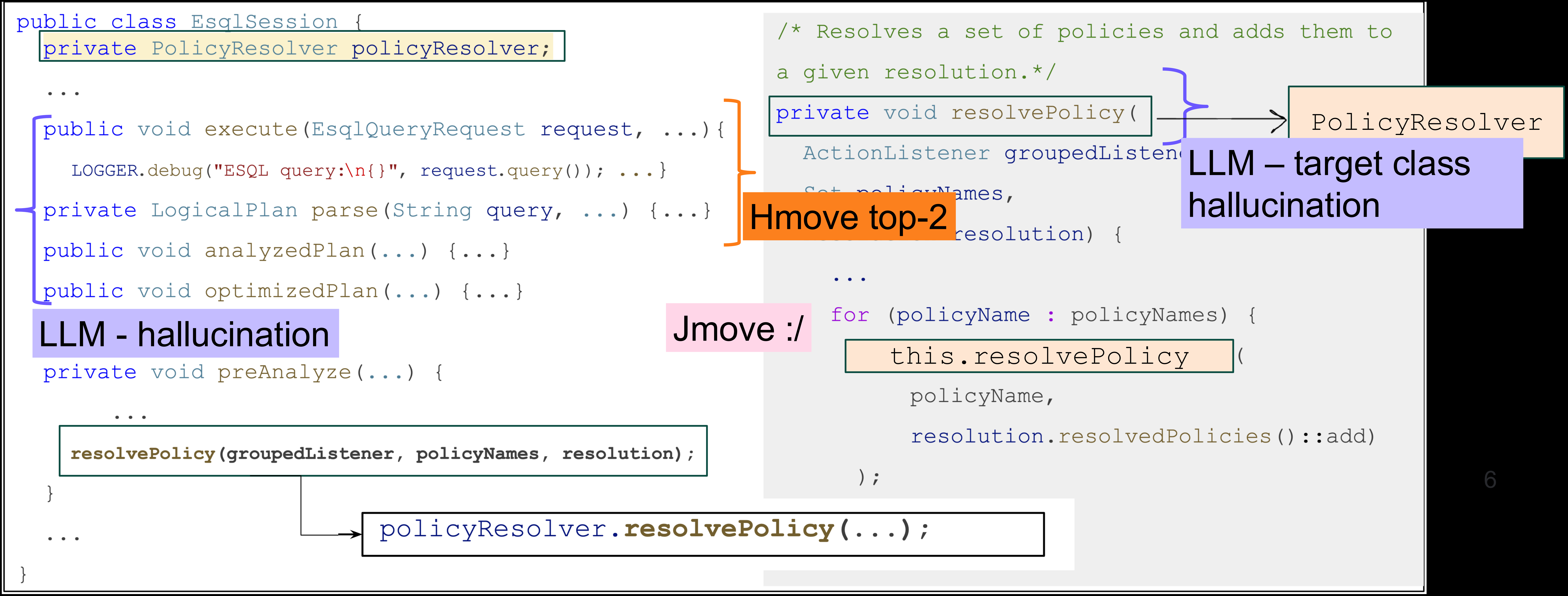
Optimize software quality metrics

Do not align with how developers refactor code

LLMs
- prolific, capture semantic intuition

MoveMethod Refactoring Case Study

ElasticSearch (876e7015)



Key Challenges in using LLMs

LLM Hallucinations - 80% invalid recommendations

Context window limits – can’t reason over large projects

Workflow fit –practical needs to be fast, IDE-integrated, don’t overwhelm developers

Our Insights

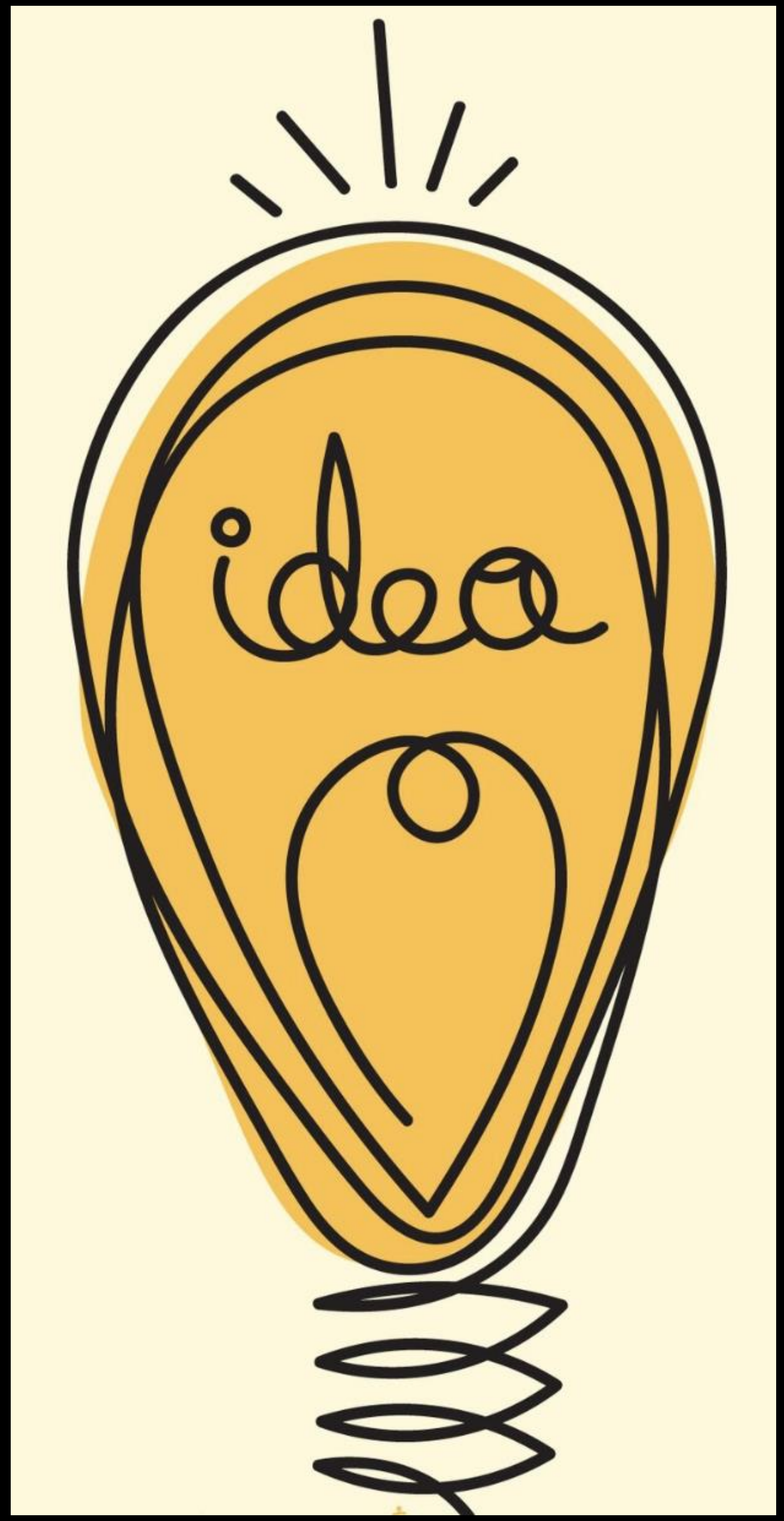
Combine LLM creativity + IDE rigor

Filter hallucination via static preconditions checks in IDE

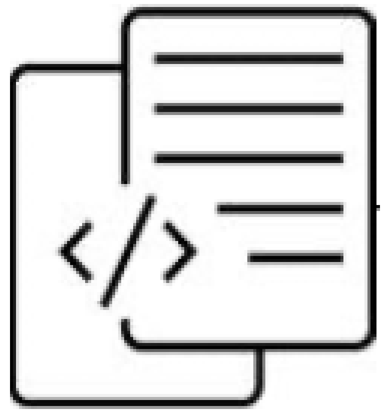
Semantic embeddings + Refactoring-aware RAG

Few high-quality recommendations (≤3 per class)

Our tool: MM-Assist, an IntelliJ Plugin



MM-Assist: Workflow



Java class

Empirical Evaluation Setup

Two Datasets:
- Synthetic corpus of 235 MM scenarios
- New real-world corpus 210 MM (2024+, OSS), avoids LLM training contamination

Formative study, OSS replication, repository mining, user study, questionnaire survey

Baselines: JMove, FeTruth, HMove, Vanilla LLM

User study: 30 participants, 1 week, own project

Results: Synthetic Corpus

Metric: Recall@K for top-K recommendations

Synthetic corpus: 235 MM scenarios

MM-ASSIST Recall@1 = 67%, Recall@3 = 75%
1.7x improvement over best baseline

LLM alone performed better than old tools but still plagued by hallucinations

Results: Real-World Corpus + User study

Replicated 210 OSS refactorings (uncontaminated by LLM training)

MM-ASSIST Recall@3 = **80% vs 33%** (best baselines) → **2.4x improvement**.

Runtime: **~30 seconds** vs hours or days for baselines.

User study: 30 devs, used on own project for a week
1150 analyzed classes -> gave recommendations in 350 classes
83% positive ratings
avg. 7 accepted refactorings/user.

MM-Assist Summary

First end-to-end LLM-powered Move Method assistant

LLM + IDE + Human >> Sum of the individual parts

LLMs (creative)+ IDE (validation) + Refactoring-Aware RAG (lookup)

2–4× better recall, 10–100× faster, 5 cents / class

Trusted by developers (83% positive)

Ongoing work: refactoring agents

MoveMethod-Assist　DEMO

ExtractMethod-Assist DEMO

Your questions

1. Robustness, Reliability, and Failure Modes (Very Popular)
Core theme: When and why does MM-Assist fail, behave inconsistently, or give brittle results?

2. Subjectivity, Human Judgment, and Architectural Intent (Very Popular)
Core theme: The tension between automated refactoring and human design intent.

3. Generalization Across Codebases, Domains, and Technical Debt (Very Popular)
Core theme: How well does the approach transfer beyond “clean” or familiar projects?

4. Dependence on Language, Tooling, and Ecosystem (Popular)
Core theme: How much of the success is Java- and IntelliJ-specific?

5. Semantics, Embeddings, and Representation Quality (Popular)
Core theme: Are embeddings the “right” abstraction for code organization?

Your Discussion Points

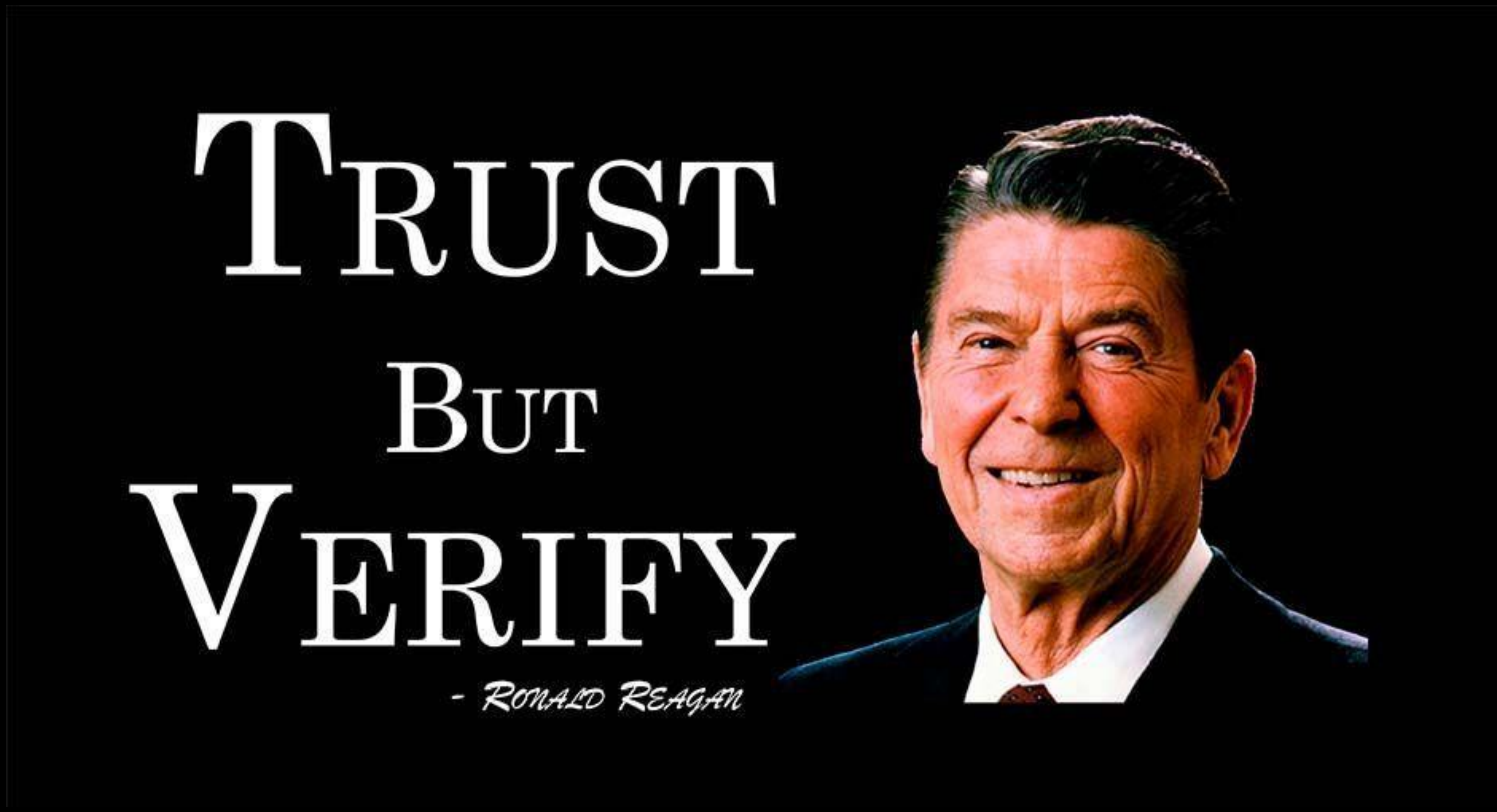
1. Role of LLM vs. System Orchestration (Very Popular)
Core idea: Understanding what actually drives MM-Assist's success—LLM reasoning or careful system design.

2. Narrow Focus vs. General Insights (Very Popular)
Core idea: Examining whether focusing on Move Method refactoring constrains or sharpens insights.

3. Safety, Correctness, and Refactoring Hygiene (Popular)
Core idea: Concerns around correctness, anti-patterns, and reversibility of refactorings.

4. Metrics, Evaluation, and Recommendation Strategy (Popular)
Core idea: Evaluating the task as a recommendation problem and how we measure success.

Lessons Learned



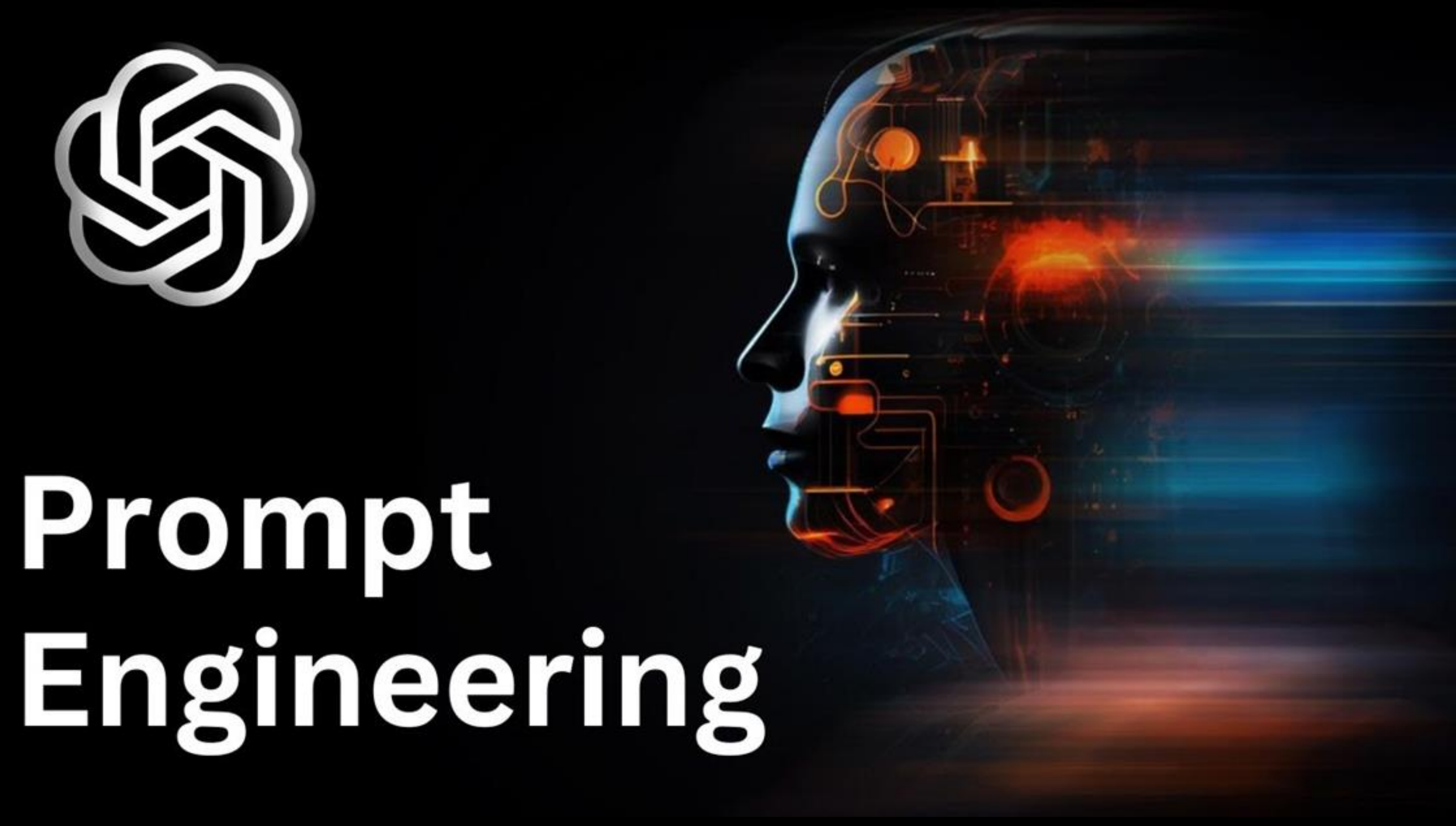
LLMs are Prolific but with High rate of hallucinations:

- ExtractMethod: 73% rate of hallucinations
- MoveMethod 80% hallucinations
- PyCraft: 65% hallucinations
- Unit tests: 35% hallucinations

Do what LLM suggests, not what they do => need for powerful validators

- remove hallucinations automatically reusing static analysis from the IDE (e.g., refactoring precondition) Where else can we reuse the IDE as validator?
- new static analysis
- dynamic analysis: generated small unit tests in PyCraft, used original code variant as validator

Lessons Learned



Precise prompt for higher quality suggestions

- append line numbers for the code input
- ask LLM to give you precise response using line numbers
- ask LLM to specify the output in structured format (JSON): useful if the output is consumed by other tools

Few-shot learning worked best for both EM-Assist and PyCraft

For MoveMethod-Assist: RAG needed to focus the LLM laser in large projects, along with Chain-of-Thought

Lessons Learned: Taming LLM nondeterminism

To get consistent high-quality suggestions, you need to reprompt (in the background), accumulate results shown to the user

Re-prompting not a waste

Newly-designed ranking to match LLM workflow (e.g., popularity of suggestions, heat map of the code affected by suggestions)

Sweet spot: tuning LLM hyperparameters (e.g., temperatures and number of iterations) is essential

- Higher randomness in Large Language Models is preferred when a solid validation framework exists

MANTRA DEMO

MANTRA: Enhancing Automated Method-Level Refactoring with Contextual RAG and Multi-Agent LLM Collaboration

Yisen Xu
SPEAR Lab, Concordia University
Montreal, Canada
yisen.xu@mail.concordia.ca

Feng Lin
SPEAR Lab, Concordia University
Montreal, Canada
feng.lin@mail.concordia.ca

Jinqiu Yang
O-RISA Lab, Concordia University
Montreal, Canada
jinqiu.yang@concordia.ca

Tse-Hsun (Peter) Chen
SPEAR Lab, Concordia University
Montreal, Canada
peterc@encs.concordia.ca

Nikolaos Tsantalis
Department of Computer Science and Software Engineering, Concordia University
Montreal, Canada
nikolaos.tsantalis@concordia.ca

Your questions

Why does the reviewer agent contribute the most to the system's performance?

How to use Mantra?

Why were these six refactorings chosen over other common refactorings?

Would MANTRA still work if refactorings were embedded in ongoing development rather than isolated "pure" commits?

Are compilation and test success sufficient proxies for refactoring correctness in all cases?

How often does the repair phase introduce changes that go beyond structural refactoring?

How dependent is MANTRA on high-quality test coverage?

Your discussion points

1> Trust, Hallucinations, and What “Correctness” Means
Whether MANTRA’s safeguards are actually sufficient, and what we should accept as “correct” in LLM-assisted refactoring.

2> Why the Reviewer Agent Matters So Much
The ablation result that the reviewer agent is central to MANTRA’s success.

3> Cognitive Load, Developer Experience, and Human Trust
Whether agentic tools truly help developers—or just shift effort elsewhere.

Project Ideas

1> What new project ideas did you get from reading these 4 refactoring papers this week?

Potential new directions:

- Extend CoRenameAgent to support new refactoring kinds (e.g., MoveMethod – close to 90% of move methods are performed in a coordinated style, other common program changes)
- Extend MoveMethod Assist to support new refactoring kinds (e.g., Split Class)
- Extend MANTRA to support new refactoring kinds
- Extend any of these tools to support other languages (e.g., Python), IDEs (VSCode), Language/IDE independence (e.g., via LSP)
- Expand RefactoringBench to include new, uncontaminated, real-world refactorings
- Implement a refactoring Agent and evaluate it against the baseline implementation from RefactorBench or MANTRA
- Design a new agent that introduces Design Patterns (e.g., from the OOAD class, Gang of Four)