

# 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 Tsantalos Concordia University

**Danny Dig** University of Colorado Boulder, JetBrains Research

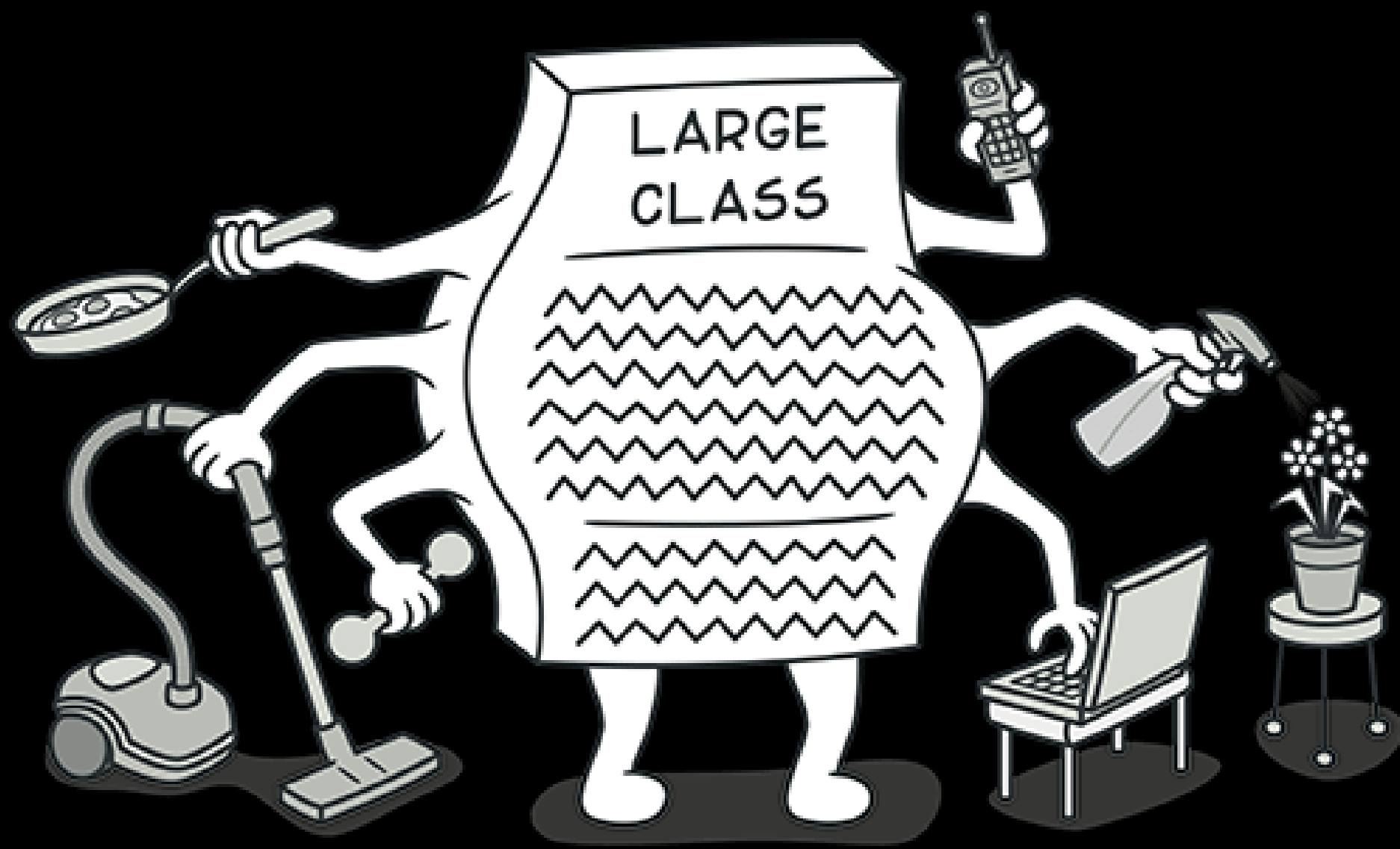


# Why MoveMethod Matters

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

```
public class Customer {  
    private Phone mobilePhone;  
  
    public String getMobilePhoneNumber() {  
        return "(" +  
            mobilePhone.getAreaCode() + ")" +  
            mobilePhone.getPrefix() + "_" +  
            mobilePhone.getNumber();  
    }  
}
```

Move to Phone class

# Current Move Method Workflow in IntelliJ



JetBrains' IntelliJ IDEA has Move Method capabilities



Semi-automated process



No automatic recommendations

```
223  * synchronization.
224  */
225  public abstract Bits getLiveDocs();
226
227  @ public FixedBitSet correctBits(DuplicateFilter duplicateFilter, Bits acceptDocs) throws IOException {
228      FixedBitSet bits = new FixedBitSet(maxDoc()); //assume all are INvalid
229      Terms terms = fields().terms(duplicateFilter.fieldName);
230
231      if (terms == null) {
232          return bits;
233      }
234
235      TermsEnum termsEnum = terms.iterator( reuse: null);
236      DocsEnum docs = null;
237      while (true) {
238          BytesRef currTerm = termsEnum.next();
239          if (currTerm == null) {
240              break;
241          } else {
242              docs = termsEnum.docs(acceptDocs, docs, DocsEnum.FLAG_NONE);
243              int doc = docs.nextDoc();
244              if (doc != DocIdSetIterator.NO_MORE_DOCS) {
245                  if (duplicateFilter.keepMode == KeepMode.KM_USE_FIRST_OCCURRENCE) {
246                      bits.set(doc);
247                  } else {
248                      int lastDoc = doc;
249                      while (true) {
250                          lastDoc = doc;
```

# 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

# Key Challenges

LLM Hallucinations - 80% invalid recommendations

Context window limits – can't reason over large projects

Workflow fit – needs to be fast and IDE-integrated



# Our Insights

Combine LLM creativity + IDE rigor

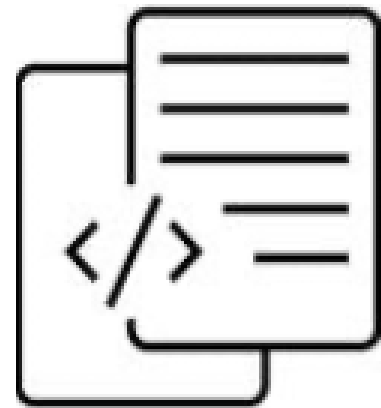
Filter hallucination via static preconditions checks in IDE

Semantic embeddings + Refactoring-aware RAG

Few high-quality recommendations ( $\leq 3$  per class)



# 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

Baselines: JMove, FeTruth, HMove, Vanilla LLM

User study: 30 participants, 1 week, own project

# Results: Synthetic Corpus

235 MM scenarios

Metric: Recall@K for top-K recommendations

MM-ASSIST Recall@1 = 67%, Recall@3 = 75%

Baselines: JMOVE ~40%, HMOVE ~26%, FETRUTH only 2–3%

**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%** (baselines) → **2.4x** improvement.

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

User study: 30 devs, 350 classes analyzed → **83% positive ratings**, avg. 7 accepted refactorings/user.

Dev quote: *“Skeptical about AI, but glad to delegate grunt work.”*

# Executive Summary

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

Key Idea: LLMs (creative)+ IDEs (validation) + Refactoring-Aware RAG (lookup)  
- addresses hallucinations + context limits via IDE + RAG

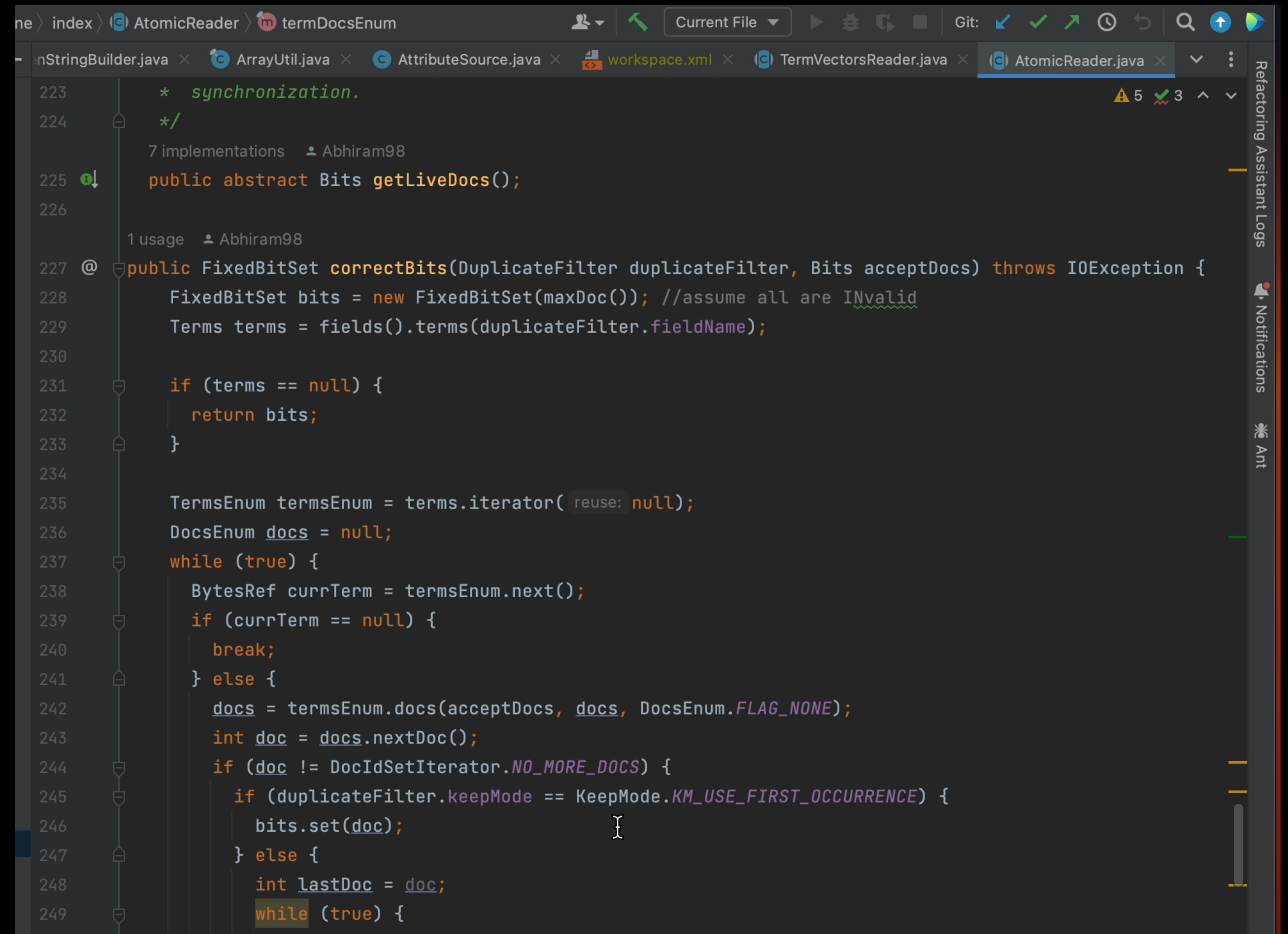
2–4× better recall, 10–100× faster

Trusted by developers (83% positive)

Techniques generalize to other refactorings

# Bonus Slides

# Move Method Refactoring in IntelliJ





# Demo MM-Assist

The screenshot displays an IDE interface with a project explorer on the left and a code editor in the center. The project explorer shows a directory structure for a project named 'kafka' located at '~Documents/TBE/evaluation\_project'. The code editor shows the source code of the 'ConfigDef' class, which is part of the 'org.apache.kafka' package. The class is a public class that implements the 'ConfigValue' interface. It contains several static fields and methods for managing configuration keys and values. The code is annotated with Javadoc comments and includes a constructor and a method for returning a set of property names.

```
74  * List<ConfigValue> configValues = defs.validate(props);
75  * // The {@link ConfigValue} contains updated configuration information given the current configuration values.
76  * </pre>
77  * <p/>
78  * This class can be used standalone or in combination with {@link AbstractConfig} which provides some additional
79  * functionality for accessing configs.
80  *
81  public class ConfigDef { 1 inheritor 1 Liquan Pei +49
82
83      private static final Pattern COMMA_WITH_WHITESPACE = Pattern.compile(regex: "\\s*,\\s*"); 1 usage
84
85      /**
86       * A unique Java object which represents the lack of a default value.
87       */
88      public static final Object NO_DEFAULT_VALUE = new Object();
89
90      private final Map<String, ConfigKey> configKeys; 19 usages
91      private final List<String> groups; 8 usages
92      private Set<String> configsWithNoParent; 5 usages
93
94      public ConfigDef() { 1 Shikhar Bhushan +1
95          configKeys = new LinkedHashMap<>();
96          groups = new LinkedList<>();
97          configsWithNoParent = null;
98      }
99
100     @
101     public ConfigDef(ConfigDef base) { 1 Ewen Cheslack-Postava +2
102         configKeys = new LinkedHashMap<>(base.configKeys);
103         groups = new LinkedList<>(base.groups);
104         // It is not safe to copy this from the parent because we may subsequently add to the set of configs and
105         // invalidate this
106         configsWithNoParent = null;
107     }
108
109     /**
110      * Returns unmodifiable set of properties names defined in this {@link ConfigDef}
111      *
112      * @return new unmodifiable {@link Set} instance containing the keys
113      */
114     public Set<String> names() { return Collections.unmodifiableSet(configKeys.keySet()); }
```

# Lessons Learned



LLM Critique –  
Can be too harsh



High hallucination rate 80%



Task Decomposition helped – instead of  
depending on LLM to do everything



Data leakage problem



# LLM Data Leakage

- Gpt-4o training data cutoff: Oct-2023

Approach	$Recall_C$		
	@1	@2	@3
MM-ASSIST	80.6	91.4	93.5
↪ <b>static method</b>	90.4	100.0	100.0
↪ <b>instance method</b>	77.8	88.9	94.4

Oracle Size	Approach	$Recall_M$			$Recall_C$			$Recall_{MC}$		
		@1	@2	@3	@1	@2	@3	@1	@2	@3
SmallClasses (38)	JMove (19)	5%	5%	5%	0%	0%	0%	0%	0%	0%
	FETRUTH	20%	20%	20%	<b>100%</b>	<b>100%</b>	<b>100%</b>	20%	20%	20%
	Vanilla-LLM	55%	68	73%	86%	86%	86%	63%	58%	63%
	MM-ASSIST	<b>76%</b>	<b>92%</b>	<b>94%</b>	86%	89%	89%	<b>71%</b>	<b>82%</b>	<b>82%</b>

Oracle Size	Approach	$Recall_M$			$Recall_C$			$Recall_{MC}$		
		@1	@2	@3	@1	@2	@3	@1	@2	@3
SmallClasses (40)	FETRUTH	7%	15%	15%	14%	14%	14%	1%	2%	2%
	Vanilla-LLM	43%	57%	65%	7%	7%	7%	3%	4%	5%
	MM-ASSIST	<b>55%</b>	<b>65%</b>	<b>70%</b>	<b>21%</b>	<b>21%</b>	<b>21%</b>	<b>12%</b>	<b>14%</b>	<b>15%</b>

# Your Questions: Core Methodology and RAG Pipeline

Regarding the core methodology, can semantic retrieval meaningfully identify better refactoring targets than traditional dependency graph analysis?

What are the specifics of the RAG pipeline—including the choice of VoyageAI models, the number of retrieved classes, and the prompt optimization strategy, and does using partial class summaries for retrieval significantly degrade recommendation quality?

How effective is cosine similarity for code, and what is the justification for weighting package proximity 2x higher than the utility metric in the RankingScore formula?

# Tool Design, Usability, and Edge Cases

From a usability standpoint, how does the tool handle edge cases like finding no suitable candidates (does it fail gracefully or hallucinate?)

What is the workflow for complex "God Classes"?

What was the rationale for limiting suggestions to three candidates, and does this hinder the full exploration of refactoring options without repeated use?

# Evaluation, Performance, and Generalizability

In terms of the evaluation, are the reported performance gains primarily due to the novel algorithm, or could they be attributed to failures or limitations in the baseline tools used for comparison?

What were the main reasons developers rejected 17.2% of suggestions?

How does the tool's performance scale to massive codebases?

How would it translate to dynamic languages like Python?

What is the expected gain from using a domain-tuned embedding model?



# Cost, Practicality, and Real-World Adoption

Considering practical adoption, what is the tool's financial viability, and how would its performance and cost change if using an open-source LLM instead of a commercial API?

What are the long-term integration complexities and developer privacy concerns?

Crucially, what are the potential implications of the LLM introducing "design hallucinations," especially when applying such a tool to safety-critical codebases?

# Future Work and Potential Improvements

For future work, beyond improving method-level suggestions by integrating static metrics or creating a personalized memory system, can this architecture be extended to support more complex tasks?

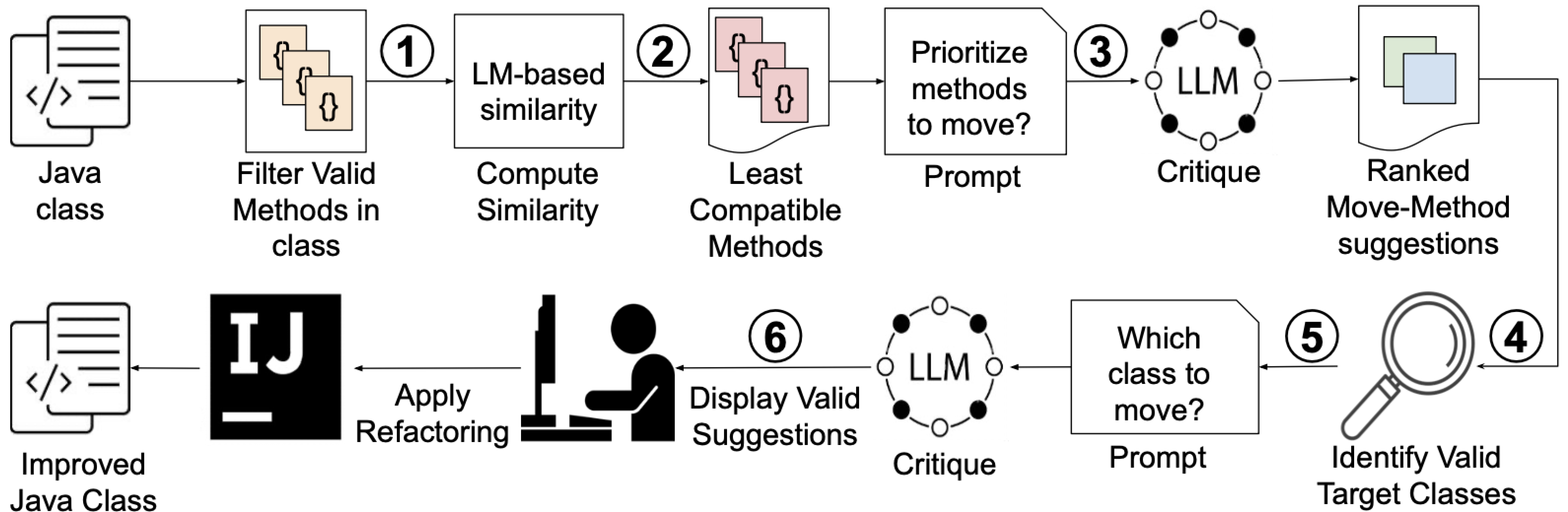
For instance, could it facilitate large-scale architectural refactorings or even serve as a tool for collaborative design sessions among multiple developers?

# Research and Development Process

Could you share insights into the development process itself? Was the plugin built from scratch or did it reuse components from EM-assist?

And how did you manage the balance between development and user testing under a tight conference deadline?

# Workflow



# RQ1: How effective are LLMs at suggesting opportunities for MM refactoring?

TABLE I: Different kinds of hallucinations from Vanilla LLM

Corpus	# R	# H1	# H2	# H3
Synthetic (235)	723	362	168	51
Real-world (210)	1293	431	275	320

R: Recommendations, H1: Hall-class, H2: Hall-Mech, H3: Invalid Method.