A Methodology for Investigating AI Patterns Prevalence in Software Repositories

Srinath Perera, Hasinthaka Piyumal, Frank Leymann, and Rania Khalaf
WSO2 (authors 1,2,4), University of Stuttgart (author 3)
e-mail: {srinath, hasinthaka, rania}@wso2.com,
frank.leymann@iaas.uni-stuttgart.de

Introduction

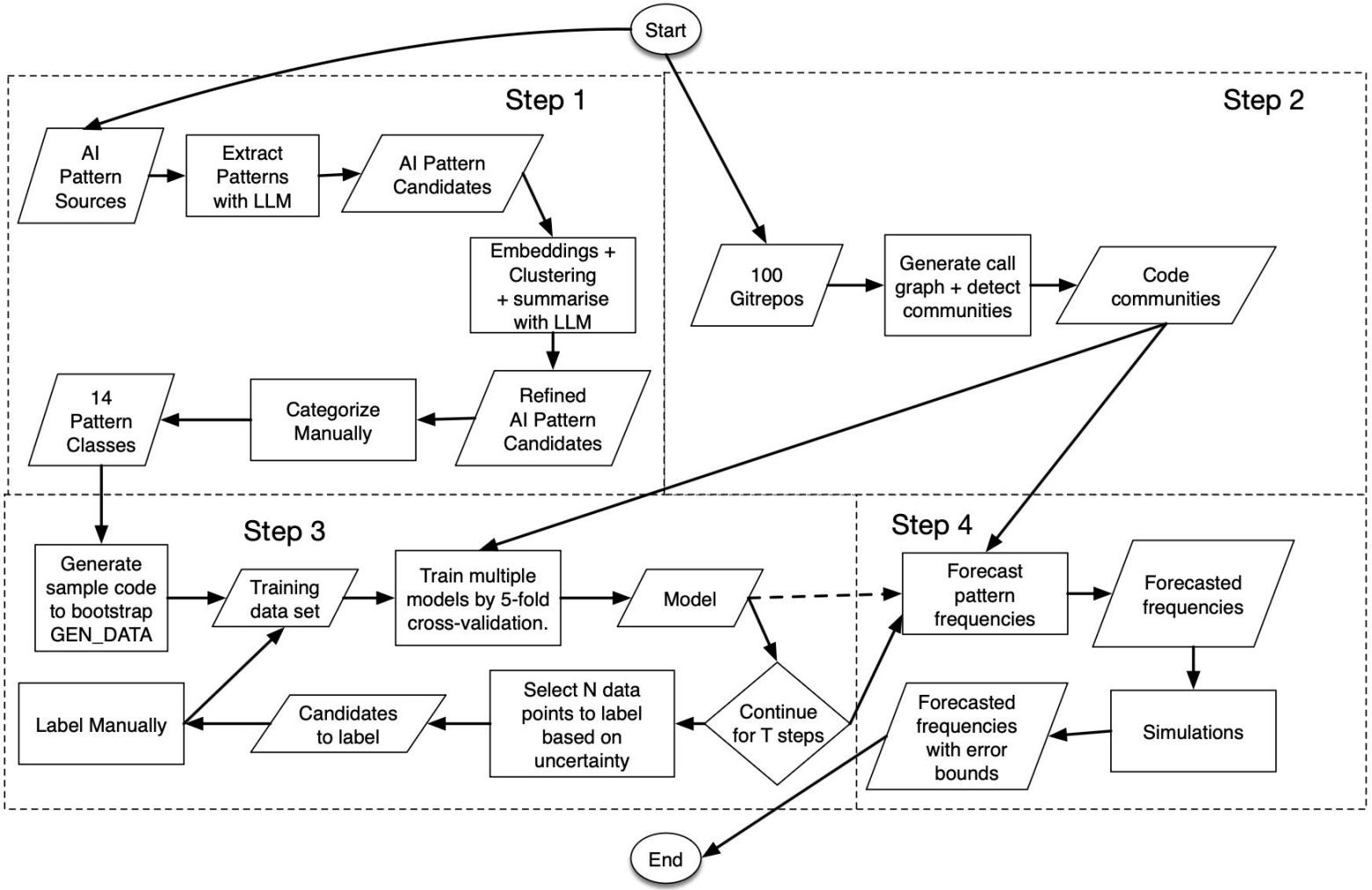
- Many AI patterns have been proposed
  - Aim to capture the authors' observations and experiences.
  - Too many
- Will help developers and students to share knowledge and good practices
- But we do not know the pattern's real-world prevalence (usage frequency)
- If we can find the real-world prevalence of patterns, it will help
  - Find the relative importance of patterns
  - Give clarity to developers
  - Can verify candidate patterns using the rule of X[6]

Related Works

- AI Patterns
  - Huang [8] is a book on LLM design patterns. Gullí [9] and Liu et al. [10] discuss agent design patterns.
  - practitioner's perspectives
    - Subramaniam [11], Jain [12] , AWS patterns web page [13], Databricks documentation [14], The Azure MLOps website [19]
  - lists a collection of Agent system design patterns.
  - RAG patterns
    - Surveys Arslan et al. [32] and Singh et al. [15]
    - present practitioners' perspectives (articles [16] and [11])
  - Among pre-LLM patterns
    - Textbook on ML patterns Lakshmanan et al. [17]
    - Taxonomy/ Surveys of ML solution patterns - Nalchigar [53], Washizaki [18], Rodaz et al. [40]
- Full list of 20 sources can be found in the paper

Related Works (Contd.)

- Pattern Mining
  - Search-based approaches use representations closer to code and search for patterns.
    - Kramer et al. [21] Dabain et al. [22], Ghulam et al. [23] , Zdun et al. [24]
  - A logical extension of this method is to represent patterns as subgraphs and to search the code graph for them.
    - May-van [25] , Tsantalis [26] , GEML [27]
  - ML-based
    - Extract features + Classification techniques
      - Uchiyama et al. [28] and Dwivedi et al. [29], Chihada et al. [30] use features, Zanoni et al. [31]
    - The downsides of initial ML techniques include the need for feature engineering, loss of information in features, and limited availability of training data.
  - Embedding based
    - Najam et al. [32] build a representation of the source code, apply the word2vec algorithm to construct an embedding space, and then use a classification algorithm.
    - Pandey et al [19] use code embeddings (RoBERTa) generated from code with a KNN algorithm to achieve an F1 score of 0.91.
    - This approach is very flexible as it can be used with minimal effort to detect new patterns if labels for the data are available.
- Most of the above approaches work with benchmarks
  - P-MART [33] and DPB [34], which primarily focus on software engineering patterns (e.g., GOF). No suitable dataset is available for detecting AI patterns.


Start
Step 1
AI Pattern Sources
Extract Patterns with LLM
AI Pattern Candidates
Embeddings + Clustering + summarise with LLM
Refined AI Pattern Candidates
Categorize Manually
14 Pattern Classes
Step 2
100 Gitrepos
Generate call graph + detect communities
Code communities
Step 3
Generate sample code to bootstrap GEN_DATA
Training data set
Train multiple models by 5-fold cross-validation.
Model
Continue for T steps
Select N data points to label based on uncertainty
Candidates to label
Label Manually
Step 4
Forecast pattern frequencies
Forecasted frequencies
Simulations
Forecasted frequencies with error bounds
End

Step 1: Extracting Pattern Candidates from the Literature

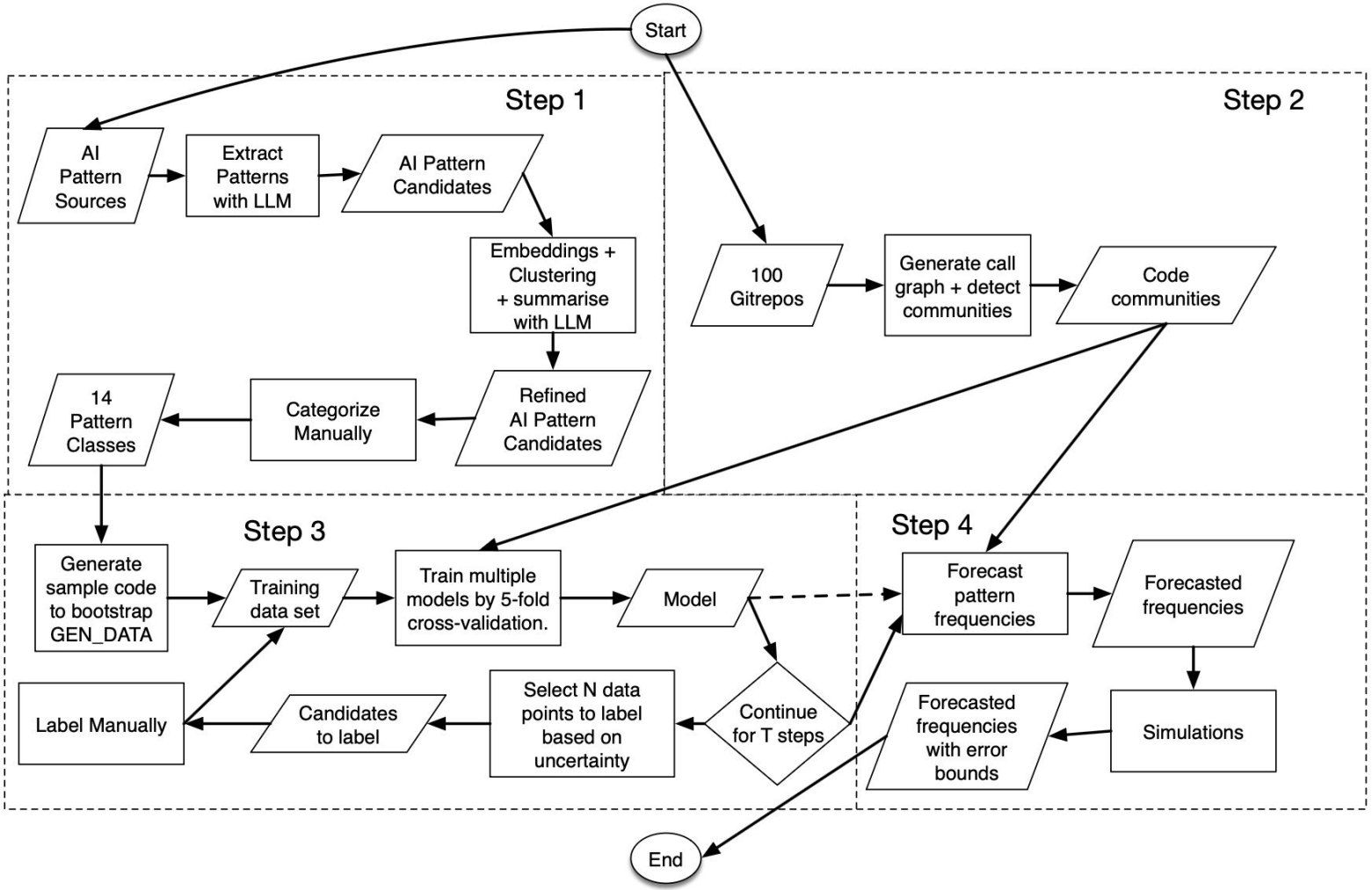
- Extracted sources listed (videos we used the transcript). We query an LLM (using Prompts [20]) to extract pattern candidates (769).
- We generated word embeddings for pattern description and clustered embeddings using the DBSCAN algorithm.
- Then, by manually inspecting 300+ pattern candidate clusters, we categorize them under 14 pattern classes.

Identified Pattern Classes

- Advanced LLM Prompting
- Forecasting with Classical Models
- Evaluating LLM Results
- LLM-based Multimodal Generative Prompting
- Model Abstraction
- Agent Architecture
- Preprocessing Text and Numerical Data
- Retrieval Augmented Generation (RAG)
- Using Tools with LLMs
- LLM-based User Intent Extraction
- LLM Fine-Tuning, Training & Alignment
- Enabling Reliability, Explainability, or Robustness
- LLM-based Planning, XoT, ReAct, or Reasoning
- MLOps

Step 2: Collecting Real-world Code Communities

- 100 open-source Python repositories hosted on GitHub with between 250 and 1,000 stars.
    - 250-1000 stars is selected because manual inspection of 50 repositories with over 1,000 stars revealed that the majority were foundational frameworks, such as TensorFlow. We believe the selected repos represent applications.
- We manually verified each candidate to ensure it was a valid AI-related project.
- For chunking the code, we built a procedure call graph and detected communities using the Louvain method, and used code communities as chunks.


Start
Step 1
AI Pattern Sources
Extract Patterns with LLM
AI Pattern Candidates
Embeddings + Clustering + summarise with LLM
Refined AI Pattern Candidates
Categorize Manually
14 Pattern Classes
Step 2
100 Gitrepos
Generate call graph + detect communities
Code communities
Step 3
Generate sample code to bootstrap GEN_DATA
Training data set
Train multiple models by 5-fold cross-validation.
Model
Continue for T steps
Select N data points to label based on uncertainty
Candidates to label
Label Manually
Step 4
Forecast pattern frequencies
Forecasted frequencies
Simulations
Forecasted frequencies with error bounds
End

Step 3: Build a model with Active learning

1. To predict pattern classes given a code community, we build a classifier that takes embeddings for the code community as input.
2. We initialize the model using sample code generated for each pattern.
3. Then, using an active learning approach, at each iteration, we selected 100 data points to label based on margin sampling (based on model probabilities), and labeled them.
4. We built a new model, including the labeled data, with a 5-fold cross-validation.
5. Then go back to step 3
6. We repeat for 4 iterations.

Results after 4 Iterations (Random chance -11%)

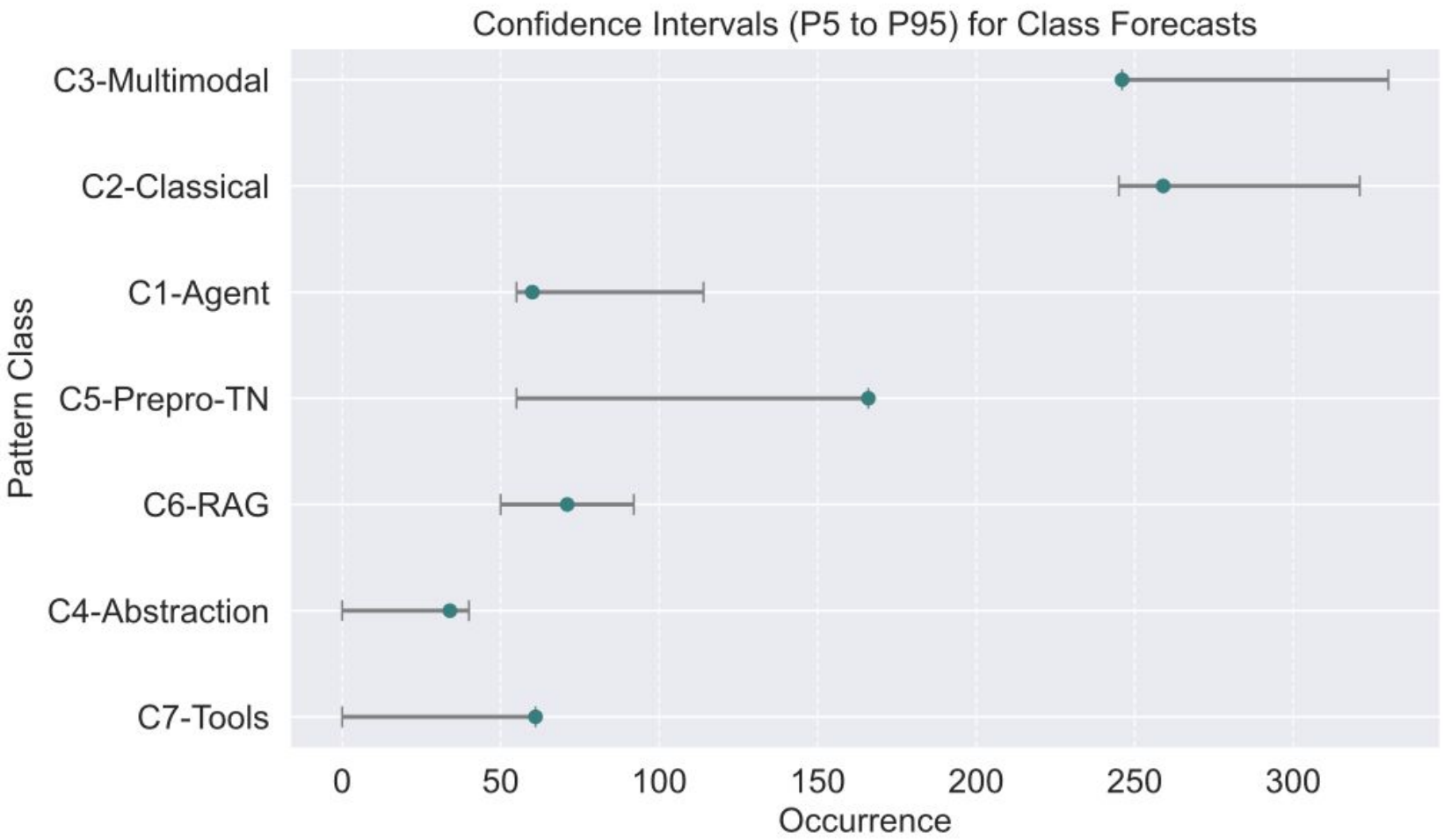
TABLE I. CLASSIFICATION PERFORMANCE METRICS ACROSS CATEGORIES.

	Category	Precision	Recall	F1-Score	Verified Data
C1	Agent Architecture	0.50	0.30	0.38	20
C2	Forecasting with Classical Models	0.69	0.79	0.74	63
C3	Multimodal Generative Prompting	0.62	0.73	0.67	52
C4	Model Abstraction	0.45	0.50	0.48	20
C5	Preprocessing Text and Numerical Data	0.53	0.50	0.52	50
C6	RAG	0.67	0.64	0.65	28
C7	Using Tools with LLMs	0.42	0.35	0.39	31
C8	None	0.56	0.54	0.55	85
	Overall Values	**0.56**	**0.55**	**0.55**	

Input: C (Confusion Matrix), $\mathbf{O}$ (Observed counts), α (significance level), N (Iterations)
Output: $[L, U]$ (Confidence interval bounds)

$\mathcal{R} \leftarrow \emptyset$ // Initialize results collection
$C_{norm} \leftarrow \text{normalize}(C)$ // Normalize columns to sum to 1
$n \leftarrow \sum \mathbf{O}$ // Total sample count
$\mathbf{p} \leftarrow \mathbf{O}/n$ // Observed frequencies
for $i \leftarrow 1$ to N do
    // Sample from multinomial to simulate observation noise
    $\mathbf{O}_{samp} \leftarrow \text{sample\_multinomial}(n, p)$
    // Solve linear system using Equation 1
    $\mathbf{E} \leftarrow \text{solve}(C_{norm}, \mathbf{O}_{samp})$
    $\mathcal{R} \leftarrow \mathcal{R} \cup \{\text{clip}(E)\}$ // Store clipped estimates
$L \leftarrow \text{percentile}(\mathcal{R}, \alpha/2)$
$U \leftarrow \text{percentile}(\mathcal{R}, 100 - \alpha/2)$
return $[L, U]$

Figure 5. Inversion-based Estimation via Monte Carlo Simulation


Confidence Intervals (P5 to P95) for Class Forecasts
C3-Multimodal
C2-Classical
C1-Agent
C5-Prepro-TN
C6-RAG
C4-Abstraction
C7-Tools
Pattern Class
0
50
100
150
200
250
300
Occurrence

So What?

- The last two classes have wide confidence bounds, but 5/7 provide usable bounds.
- Existence of bounds means we know when estimates can’t be used.
- Provide relative availability - For example, even using given confidence levels, we can argue that pattern classes C2 and C3 are much more common than C1, C5, and C6. Such understanding can help us focus our attention on teaching these patterns.

Concrete examples: based on the model's forecast

- **Agent Architecture** - multi-agent debate, multi-agent orchestration, communication through an agent bus, coordination through group chat.
- **LLM-based Multimodal Generative Prompting** - Predicting object centers, dimensions, and local offsets, inferring an object's 3D position, bounding box prediction, motion prediction, sequential detections of an object from Lidar data, resizing and padding 3D tensors, generating synthetic training data
- **Model Abstraction** - abstracting multiple AI backends or local/ cloud producers, routing, adopting Prompts via template, and managing API keys
- **Preprocessing Text and Numerical Data** - anonymizes, binning, type conversions, data dictionary, extracting metadata, text normalization, categorical feature encoding
- **Retrieval Augmented Generation(RAG)** - search with vector database, using custom knowledge bases, caching, and retriever component

Contributions

- By combining methods active learning and human-in-the-loop annotations, we propose a robust methodology to detect patterns without pre-existing annotated datasets, thereby addressing a significant bottleneck in software engineering research: labeled data for niche or emerging patterns is rare.
- We propose a new chunking method to generate code embeddings useful for pattern detection by applying the Louvain method [54] to call graphs to identify "code communities" that serve as chunks.
- We propose applying prevalence estimation techniques (matrix inversion and Monte Carlo simulations) to estimate the true frequency of patterns with useful error bounds.
- We synthesized 769 pattern candidates into a refined taxonomy of 14 pattern classes. We provided one of the first empirical validations of AI pattern prevalence in real-life code by analyzing 100 open-source repositories.

Questions?

Confusion Matrix

- Strong diagonal
- None is common source of error