Cross-Language Code Search using Static and Dynamic Analyses

Cross-Language Code-to-Code Search

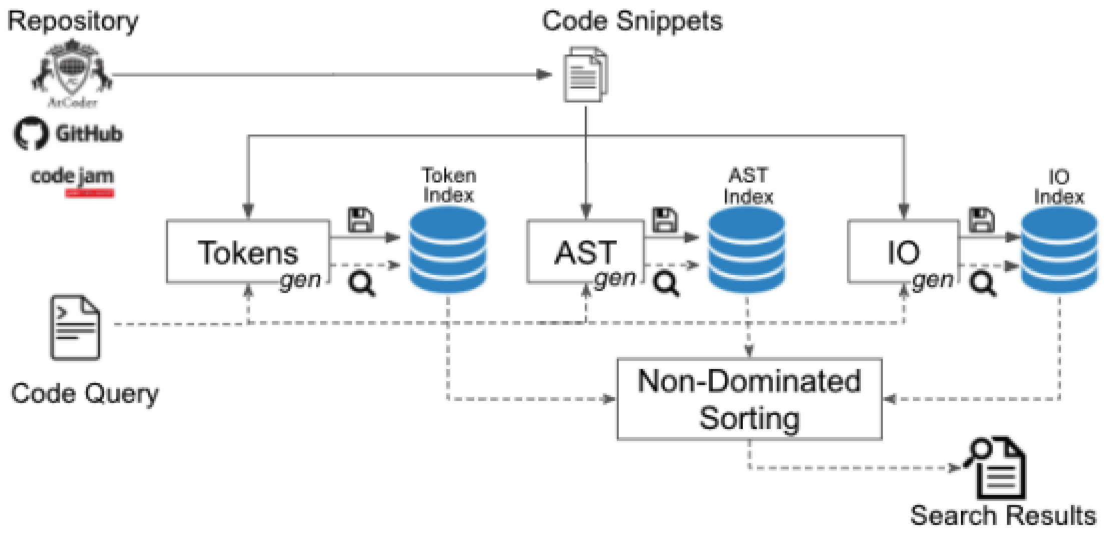
- Code-to-code search describes the task of **using a code query to search for similar code in a repository**.

- This task is particularly challenging when the query and results belong to different languages due to **syntactic and semantic differences between the languages.**

- This task is involved in identifying code clones, finding translations of code in a different language, program repair, and supporting students in learning a new programming language

CODE-TO-CODE SEARCH ACROSS LANGUAGES (COSAL)

1. Token-Based Search
2. AST-Based Search
3. Input-Output Based Search
4. Non-Dominated Sorting for extracting results

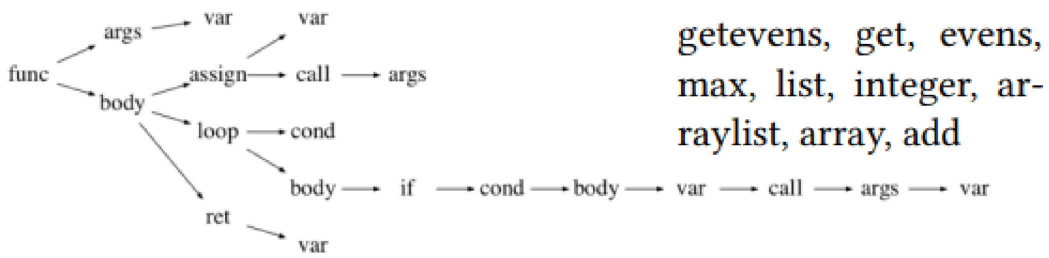


Token-Based Search

- Remove language-specific keywords based on the documentation
- Remove frequently-used words used in a language based on common coding conventions.
- Remove common stopwords from the English vocabulary.
- Split tokens to address language-specific nomenclature.
- Remove tokens of length less than MIN_TOK_LEN.
- Convert all the tokens to lower case

```java
List<Integer> getEvens(int max) {
    List<Integer> evens = new ArrayList<>();
    for(int i = 0; i < max; i++)
        if (i % 2 == 0)
            evens.add(i);
    return evens;
}
```

(a) Java: for loop to populate an array of even numbers

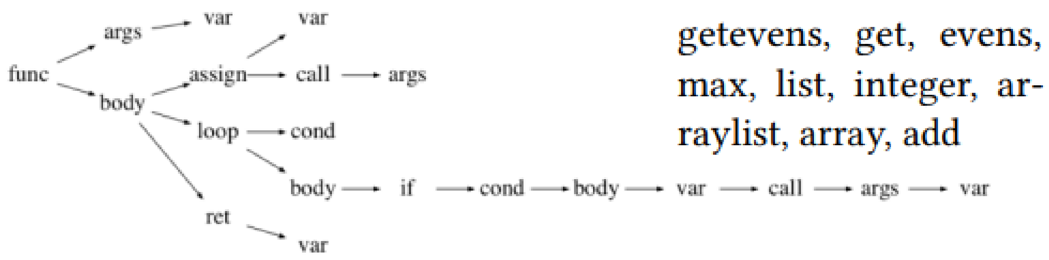


AST-Based Search

- **Common control structures**: Control structures are simplified and clustered.
- **Normalizing Variable**: Variables are denoted as var nodes.
- **Normalizing Literals**: Literals are denoted as lit nodes.
- **Normalizing Operators**: Operators are denoted as op nodes.
- **Language specific features**: If a feature is implemented in only one language, a custom node is created.

```java
List<Integer> getEvens(int max) {
    List<Integer> evens = new ArrayList<>();
    for(int i = 0; i < max; i++)
        if (i % 2 == 0)
            evens.add(i);
    return evens;
}
```

(a) Java: for loop to populate an array of even numbers



Input-Output Based Search

- SLACC segments code into executable snippets of size greater than MIN_STMTS and executed on ARGS_MAX arguments generated using a grey-box strategy.

- The executed functions are then clustered using a similarity measure (sim) based on the inputs and outputs of the functions.

$$d_{IO}(q, s) = \frac{1}{|Q|} \sum_{q_i \in Q} \underset{s_k \in S}{maximize}\ sim(q_i, s_k)$$

Non-dominated Ranking

- A search result s is said to dominate a search result t, if s is no worse than t in any objective and is better than t in at least one objective. Otherwise, there is a tie.

- To break ties, we compute distances between each search result and the optimal value for each similarity measure.

scenario	d_A	d_B	d_C	winner
1	$\mathbf{s} > t$	$\mathbf{s} > t$	$\mathbf{s} > t$	s
2	$s = t$	$s = t$	$\mathbf{s} > t$	s
3	$s = t$	$s < \mathbf{t}$	$\mathbf{s} > t$	tie
4	$s < \mathbf{t}$	$s < \mathbf{t}$	$\mathbf{s} > t$	tie

Metrics

- *Precision@k* or *P@k* is the average percentage of relevant results in the top-k search results for a query.
- *SuccessRate@k* or *SR@k* is the percentage of queries for which one or more relevant result exists among the top-k search results.
- *MRR* is the Mean Reciprocal Rank of the relevant results for a query .

$$P@k = \frac{\sum_{q \in Q} \frac{|R_q^k|}{k}}{|Q|} \quad SR@k = \frac{\sum_{q \in Q} \delta_k(BR(q))}{|Q|} \quad MRR = \frac{\sum_{q \in Q} \frac{1}{BR(q)}}{|Q|}$$

RQ1: Single vs Multiple Search Similarity Measures

- Using non-dominated ranking with static and dynamic similarity measures improves the quality of results for code-to-code search compared to subsets or a weighted aggregation of measures

	Search	MRR	P@1/3/5/10	SR@1/3/5/10
SotP	ElasticSearch	29	27/25/23/24	27/44/57/75
	GitHub	37	32/36/38/39	32/49/60/73
Single Sim.	COSAL_{token}	31	27/31/40/42	27/48/58/72
	COSAL_{AST}	34	34/41/45/44	34/41/58/82
	COSAL_{SLACC}	45	42/42/35/27	42/45/47/47
Multi Sim.	COSAL_{static}	43	40/45/44/48	40/72/85/86
	$\text{KD}_{IO+AST+token}$	39	39/41/40/37	39/56/71/89
	COSAL	**64**	**58/64/65/61**	**58/88/91/94**

RQ2: State-of-the-Practice Cross-Language Code-to-Code Search

- COSAL obtains better Precision@k, SuccessRate@k and MRR compared to GitHub Search and ElasticSearch.

	Search	MRR	P@1/3/5/10	SR@1/3/5/10
SotP	ElasticSearch	29	27/25/23/24	27/44/57/75
	GitHub	37	32/36/38/39	32/49/60/73
Single Sim.	COSAL_{token}	31	27/31/40/42	27/48/58/72
	COSAL_{AST}	34	34/41/45/44	34/41/58/82
	COSAL_{SLACC}	45	42/42/35/27	42/45/47/47
Multi Sim.	COSAL_{static}	43	40/45/44/48	40/72/85/86
	$\text{KD}_{IO+AST+token}$	39	39/41/40/37	39/56/71/89
	COSAL	**64**	**58/64/65/61**	**58/88/91/94**

RQ3: State-of-the-Art Code-to-Code Search

- Compared to state-of-the-art Java code-to-code search FaCoY, using dynamic information helps COSAL obtains better search results when executable code snippets are present. In the absence of dynamic information, a combination of AST and token-based similarity measures still yields better results than FaCoY.

		Search	MRR	P@1/3/5/10	SR@1/3/5/10
AtCoder	*SotA*	FaCoY	51	37/35/33/32	37/40/49/63
	Single Sim.	COSAL_{tokens}	46	36/32/31/29	36/40/45/58
		COSAL_{AST}	40	38/33/31/28	38/42/51/69
		COSAL_{SLACC}	40	39/39/38/32	39/48/52/59
	Multi Sim.	COSAL_{static}	53	43/45/44/41	43/58/65/77
		COSAL	**57**	**50/53/54/48**	**50/63/75/88**
BigCloneBench	*SotA*	FaCoY	76	70/68/68/65	70/72/74/81
	Single Sim.	COSAL_{tokens}	75	69/65/61/59	69/72/74/81
		COSAL_{AST}	72	68/61/55/51	68/74/76/83
		COSAL_{SLACC}	07	06/02/01/01	06/07/07/09
	Multi Sim.	COSAL_{static}	**81**	76/**73/72**/67	76/**81/89/94**
		COSAL	**81**	**77/73/72/68**	**77/81/89/94**

RQ4: Cross-Language Code Clone Detection

- For code clone detection, COSAL obtains better precision, recall and F1 scores compared to ASTLearner and CLCDSA, without the need to build models. COSAL has lower precision to SLACC but much better recall and F1 score.

	Clone Detector	Precision	Recall	F1
Single Sim.	ASTLearner	25	80	38
	CLCDSA	49	83	62
	SLACC	**66**	19	30
Multi Sim.	COSAL_{static}	48	85	61
	COSAL	55	**89**	**68**