Accelerating Code Search with Deep Hashing and Code Classification

王焱林
wangylin36@mail.sysu.edu.cn
2022.11.27 MLNLP

Accelerating Code Search with Deep Hashing and Code Classification

Wenchao Gu[1]*, Yanlin Wang[2,†], Lun Du[2], Hongyu Zhang[3], Shi Han[2], Dongmei Zhang[2], and Michael R. Lyu[1]

[1] Department of Computer Science and Engineering,
The Chinese University of Hong Kong, China.

[2] Microsoft Research Asia, Beijing, China

[3] The University of Newcastle, Australia

What is code search? (in a narrow sense)

- Code search aims to retrieve the code snippets according to the natural language query/description given by users.

Query: how to sort an array with bubble sort?

Answer:

```
void BubbleSort(int arr[], int n) {
    int i, j, flag,temp;
    for(i = 0; i < n-1; i++) {
        flag = 0;
        for(j = 0; j < (n-i-1); j++) {
            if(arr[j] > arr[j+1]) {
                flag = 1;
                temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
        if(flag == 0)
            break;
    }
}
```

What is code search?

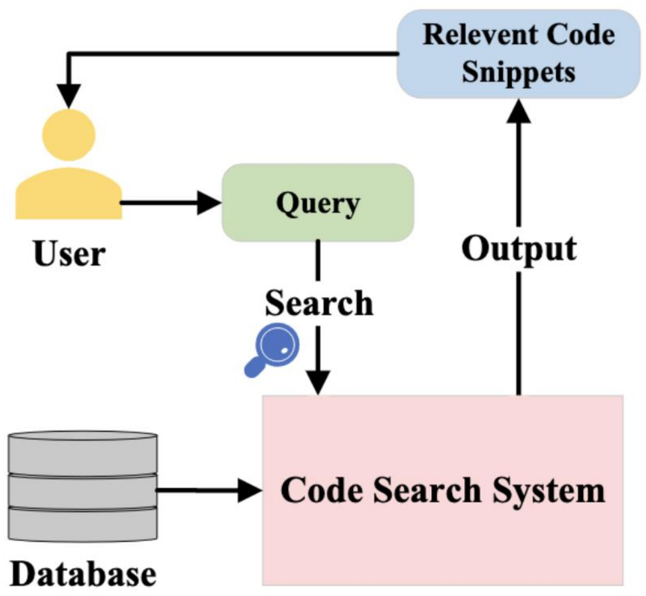
- Given a query, return its corresponding code

No.	Code Search Task	Query	Codebases	#Studies	Percent
1	Text-Based Code Search	Text	Source Code	34	51%
2	Code Clone Search	Source Code	Source Code	9	14%
3	I/O Example Code Search	Input/Output Example	Source Code	8	12%
4	API-Based Code Search	API	Source Code	7	11%
5	Binary Clone Search	Binary Code	Binary Code	5	7%
6	UI Code Search	UI Sketch	UI Code	3	4%
7	Programming Video Search	Text	Code in Video	1	1%
-	**Total**	-	-	67	100%

Opportunities and Challenges in Code Search Tools. Liu et. al. ACM Computing Surveys. 2020.

Why is it important?

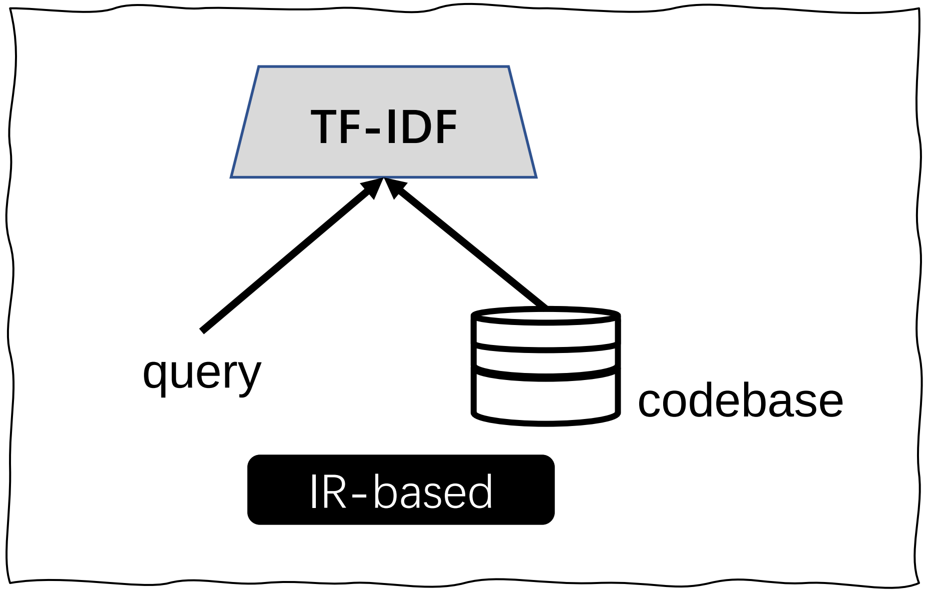
- Code search is a frequent developer activity in software development practices.
- It improves programming productivity as developers' time and energy can be saved by reusing existing code.
- On Google Code Search, a developer composes 12 search queries per weekday on average *[Sadowski et al.]*.

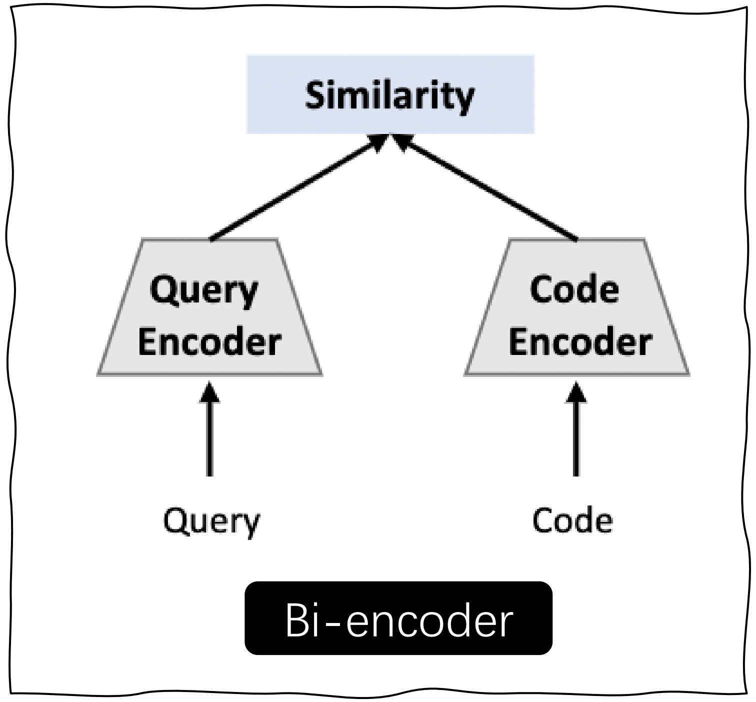


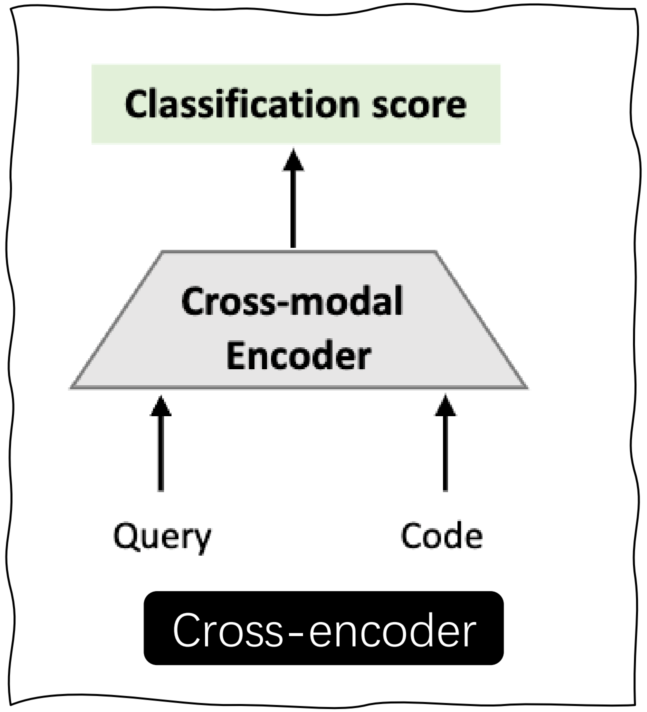
C. Sadowski, K. T. Stolee, and S. Elbaum, “How developers search for code: a case study,”FSE 2015.

Existing approaches

- Information Retrieval-based
  - BOW, Jaccard, TF-IDF, BM25, etc
- Deep Learning-based
  - Cross-encoder paradigm
    - CODEnn
    - biRNN, SelfAtt
    - CodeBERT
    - …
  - Bi-encoder paradigm
    - GraphCodeBERT
    - CoCLR
    - …



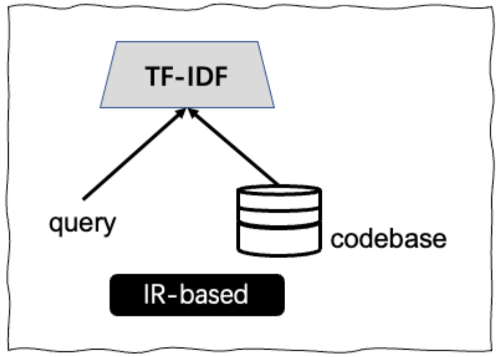


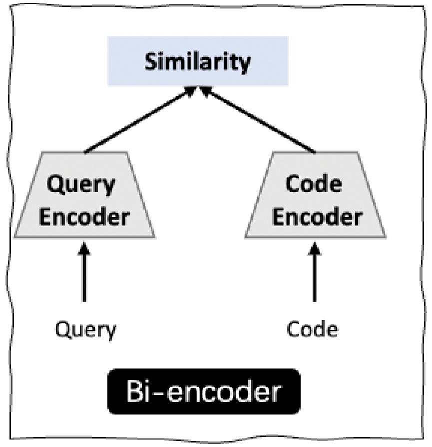


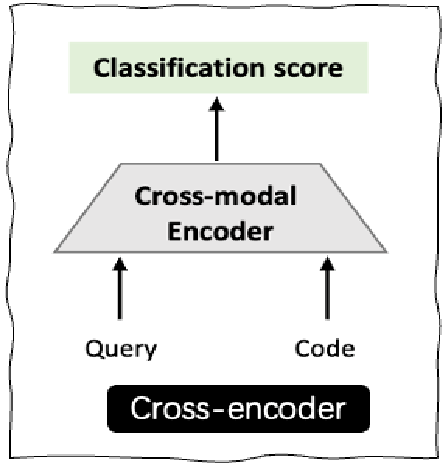
Limitations

IR-based:
Fast but not accurate

DL-based:
Accurate but slow, especially for the cross-encoder paradigm!





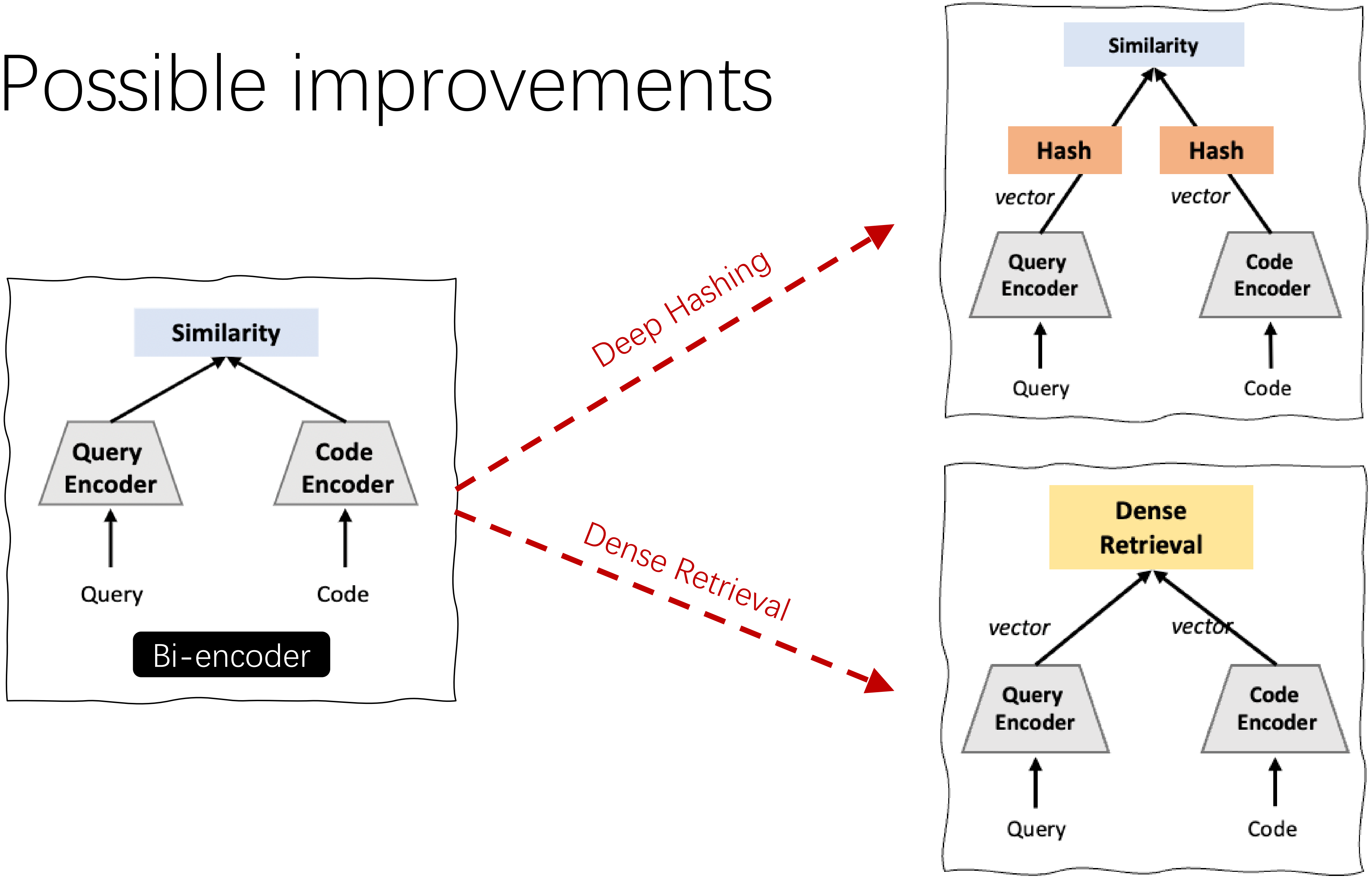


Model	MRR	Need training	Per query time / s	R@1	R@5	R@10	R@100	R@1000
Text matching IR model								
Jaccard	0.2425	No	0.0130 ± 0.0004	17.7	30.7	36.7	59.4	83.0
BOW	0.2220	No	0.0011 ± 0.0000	16.1	28.1	33.6	56.6	82.6
TFIDF	0.2397	No	0.0011 ± 0.0001	16.9	30.8	37.1	62.9	87.1
BM25	0.4523	No	0.0062 ± 0.0003	35.6	56.4	63.4	81.0	92.0
Bi-encoder DL model								
CODEnn	0.1775	Yes	0.0033 ± 0.0001	11.1	23.9	30.7	57.3	82.3
CodeBERT-bi	0.6669	Yes	0.0021 ± 0.0003	57.4	77.9	83.3	94.6	98.8
GraphCodeBERT	0.6948	Yes	0.0048 ± 0.0002	59.3	82.1	87.3	96.5	99.1
Cross-encoder DL model								
CodeBERT	**0.7015**	Yes	802.43 ± 51.29	62.4	79.2	83.7	94.5	98.7
CoCLR	0.6349	Yes	766.27 ± 47.79	51.6	78.3	84.6	95.7	99.0

Revisiting Code Search in a Two-Stage Paradigm, Hu et. al., WSDM 2023.

Previous research on code search mainly focuses on accuracy but neglects the retrieval efficiency.

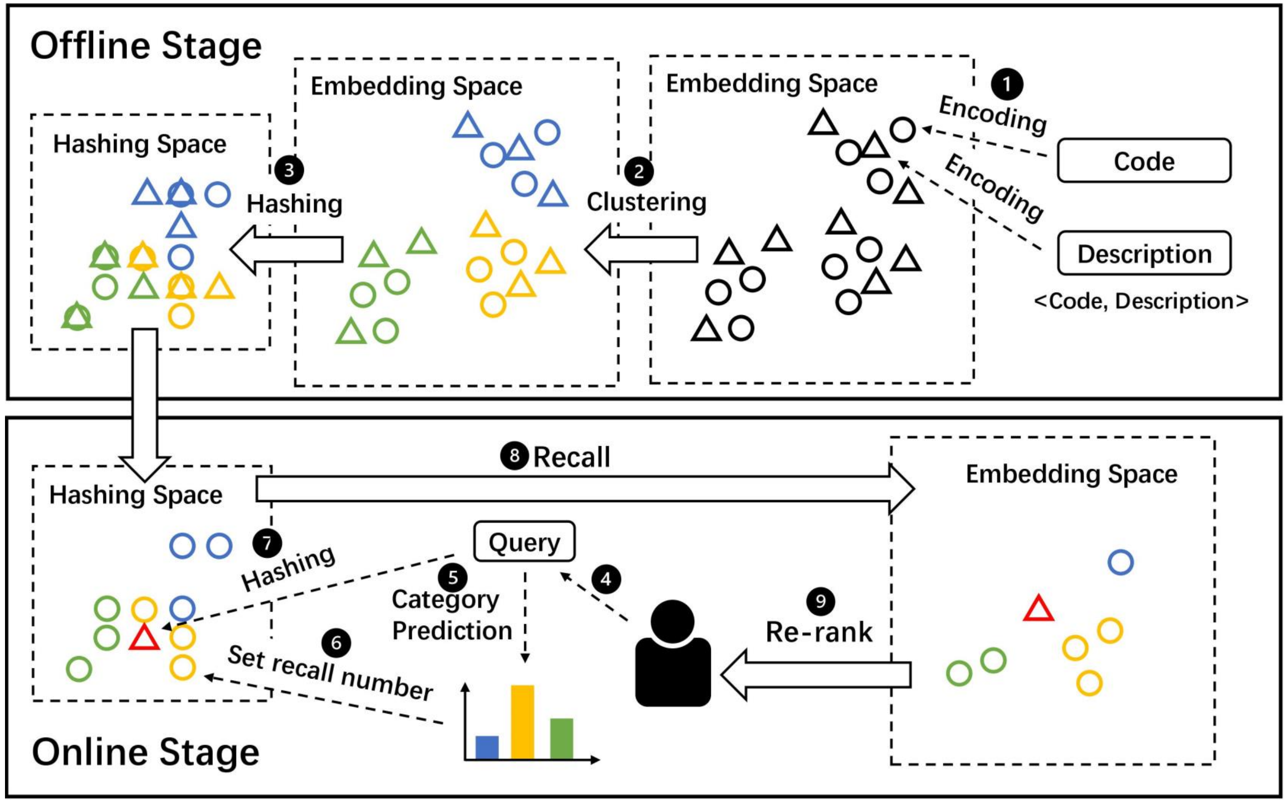
Possible improvements



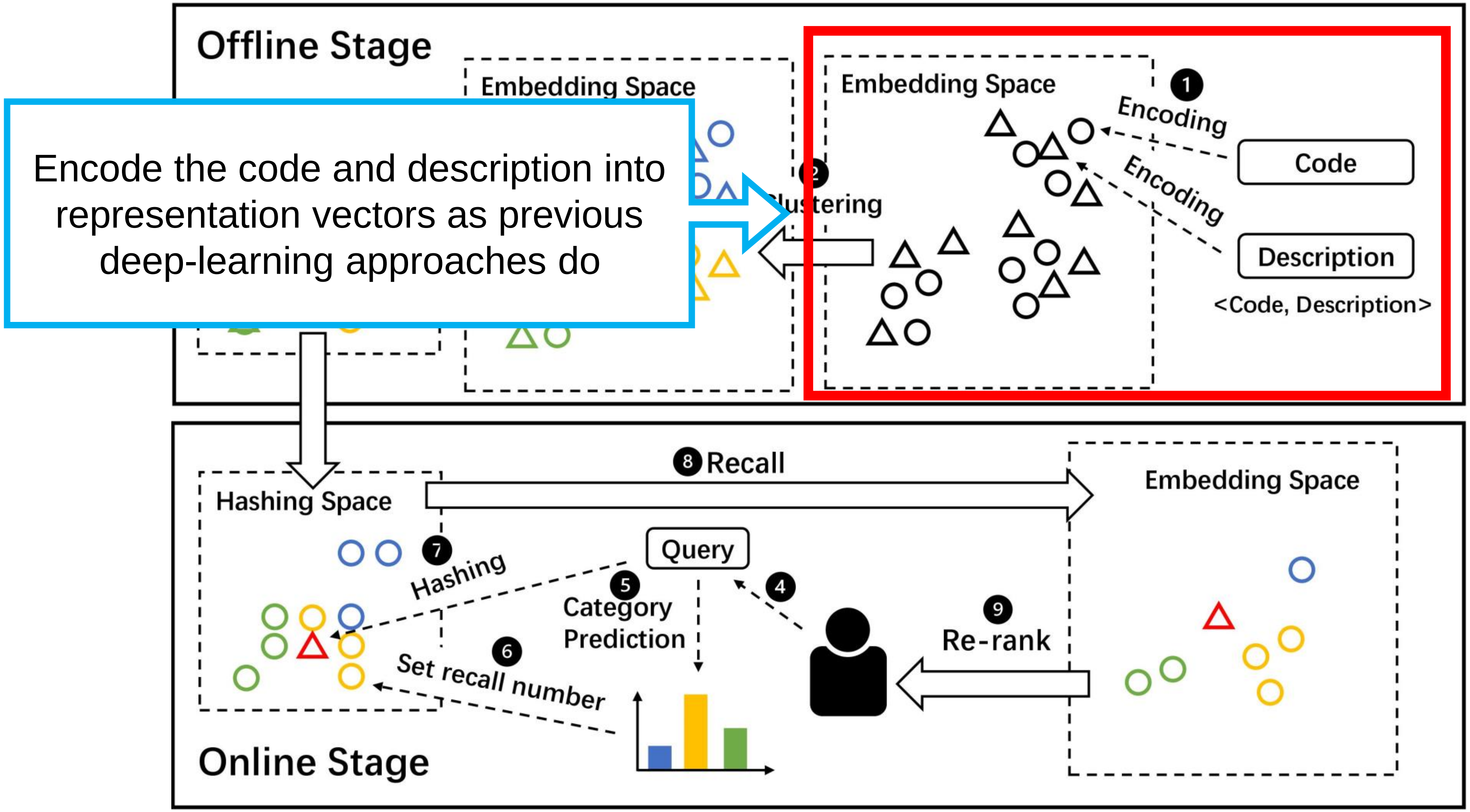
Our Exploration on the Deep Hashing Approach:

CoSHC: Accelerating Code Search with Deep Hashing and Code Classification (ACL'22)

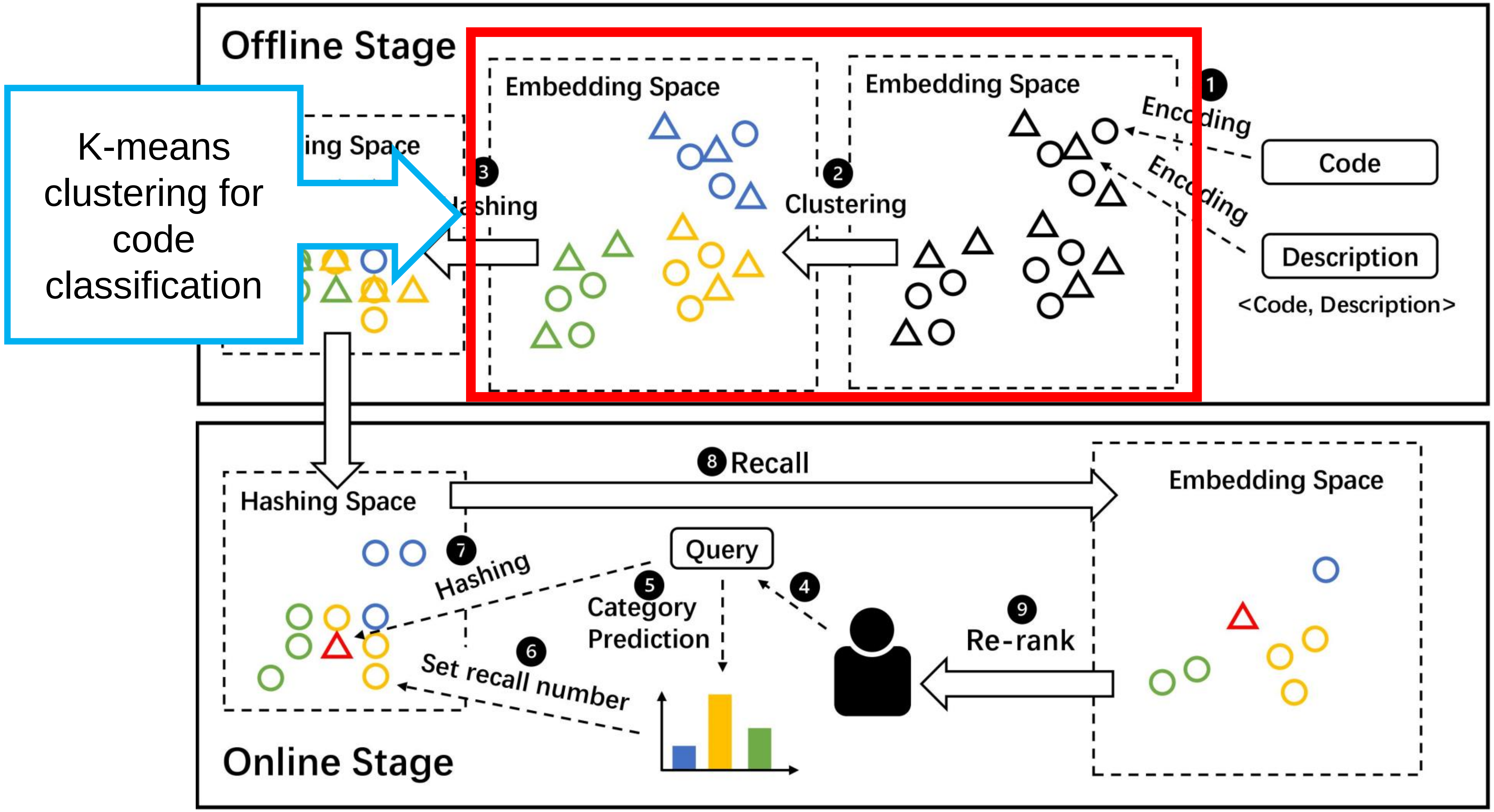
- We adopt the recall and re-rank mechanism with the integration of code clustering and deep hashing to improve the retrieval efficiency.
- Offline stage + Online stage



Framework - Offline Stage

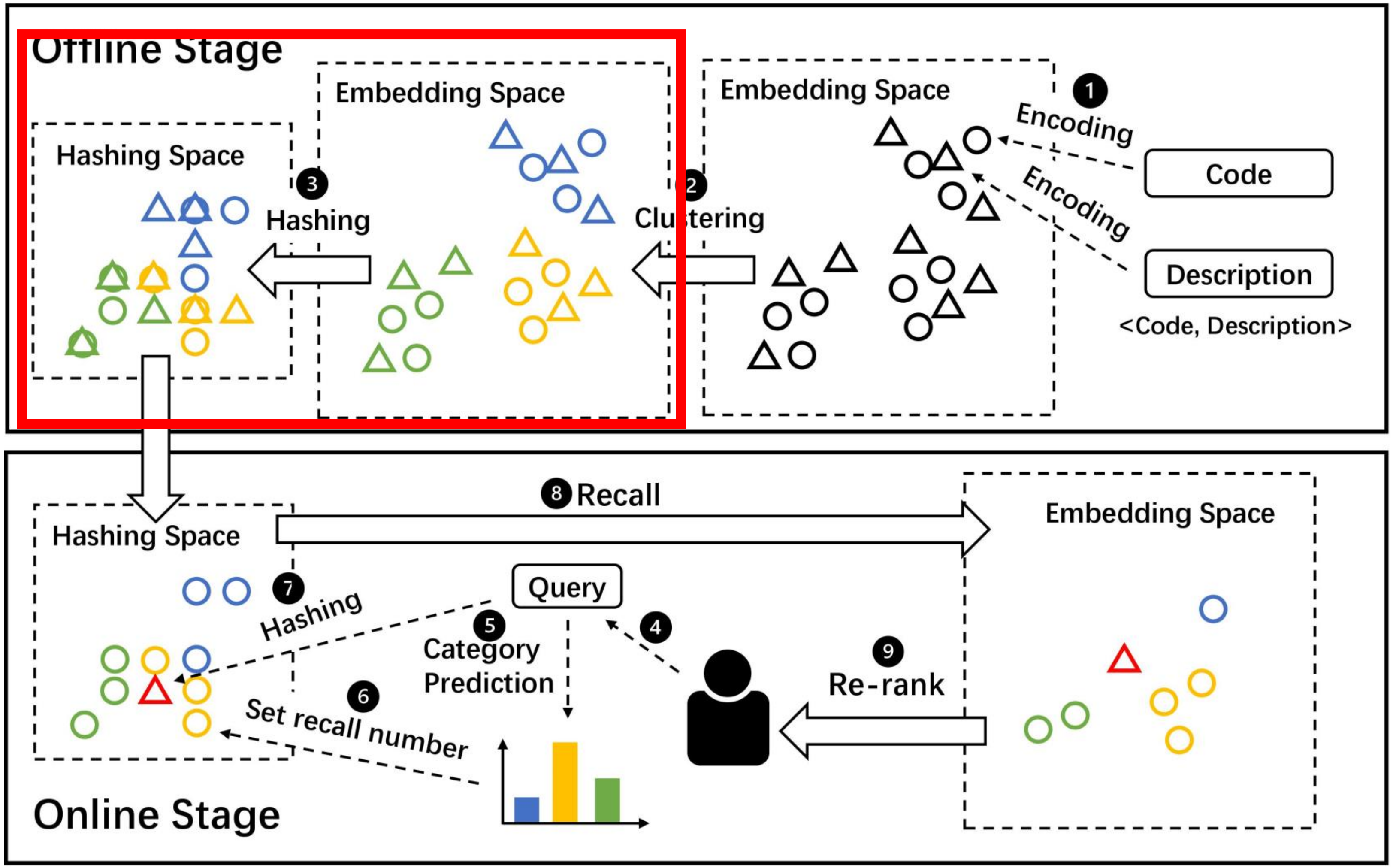


Framework



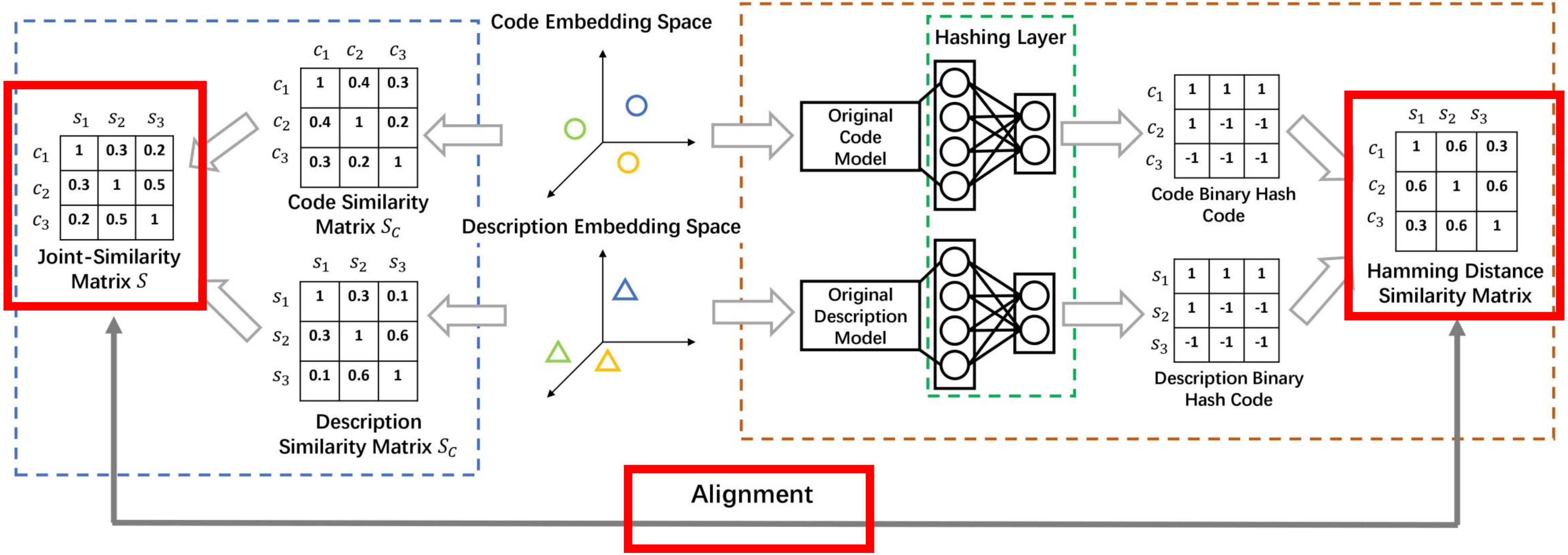
K-means clustering for code classification
Offline Stage
ing Space
Embedding Space
Embedding Space
3 Hashing
2 Clustering
1 Encoding
Encoding
Code
Description
<Code, Description>
8 Recall
Hashing Space
Embedding Space
7 Hashing
Query
4
5 Category Prediction
6 Set recall number
9 Re-rank
Online Stage

Framework

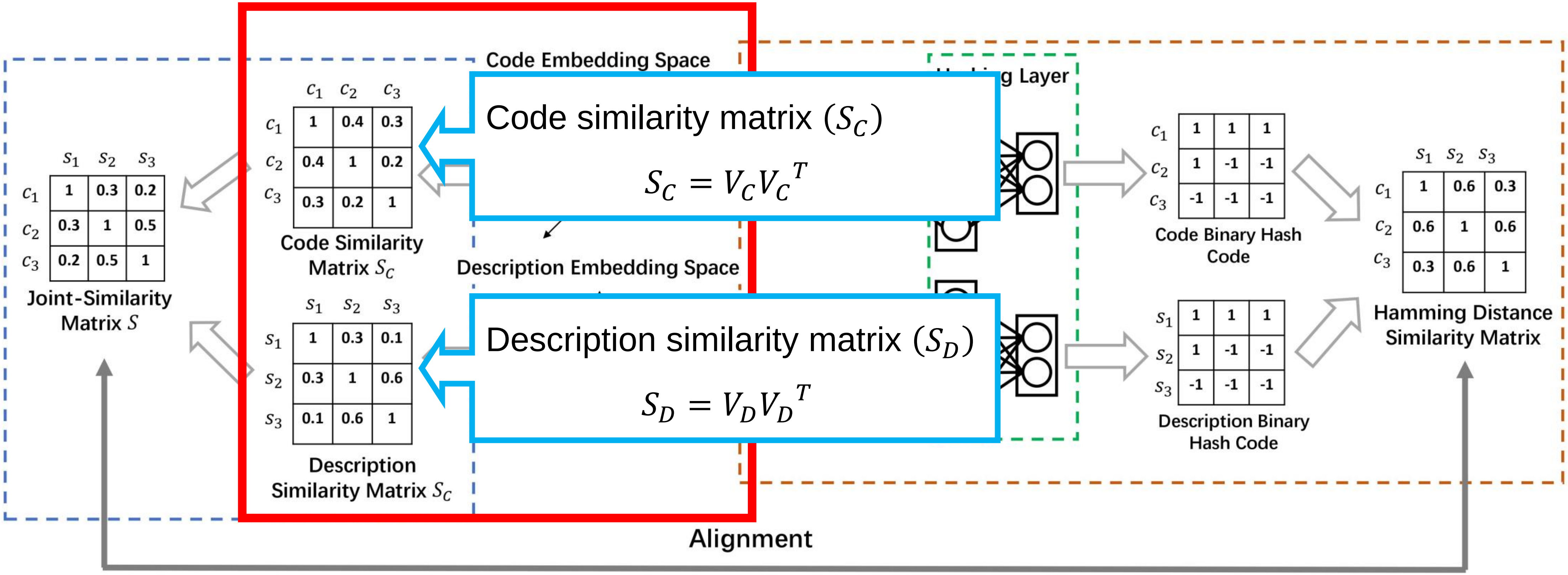


Offline Stage
Embedding Space
Hashing Space
3
Hashing
2
Clustering
Embedding Space
1
Encoding
Encoding
Code
Description
<Code, Description>
8 Recall
Hashing Space
Embedding Space
Query
7
Hashing
5
Category
Prediction
4
9
Re-rank
6
Set recall number
Online Stage

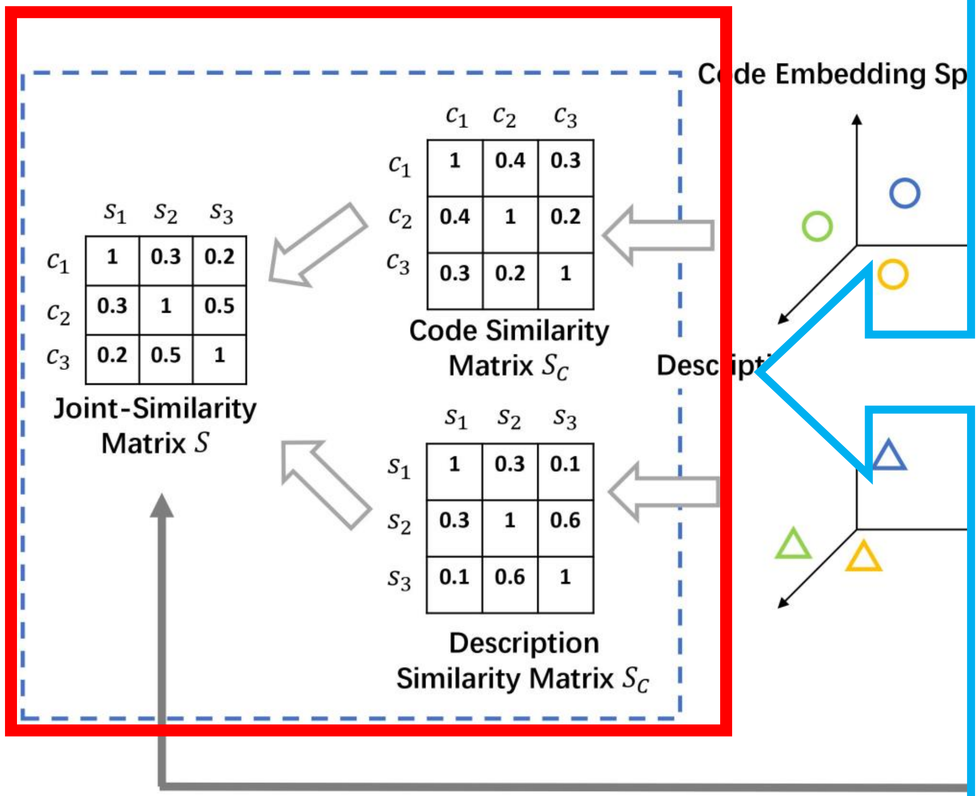
Architecture of the hashing module



Architecture of the hashing module



Architecture of the hashing module



Combines both code and description similarity matrix:

$$\tilde{S} = \beta S_C + (1-\beta) S_D$$

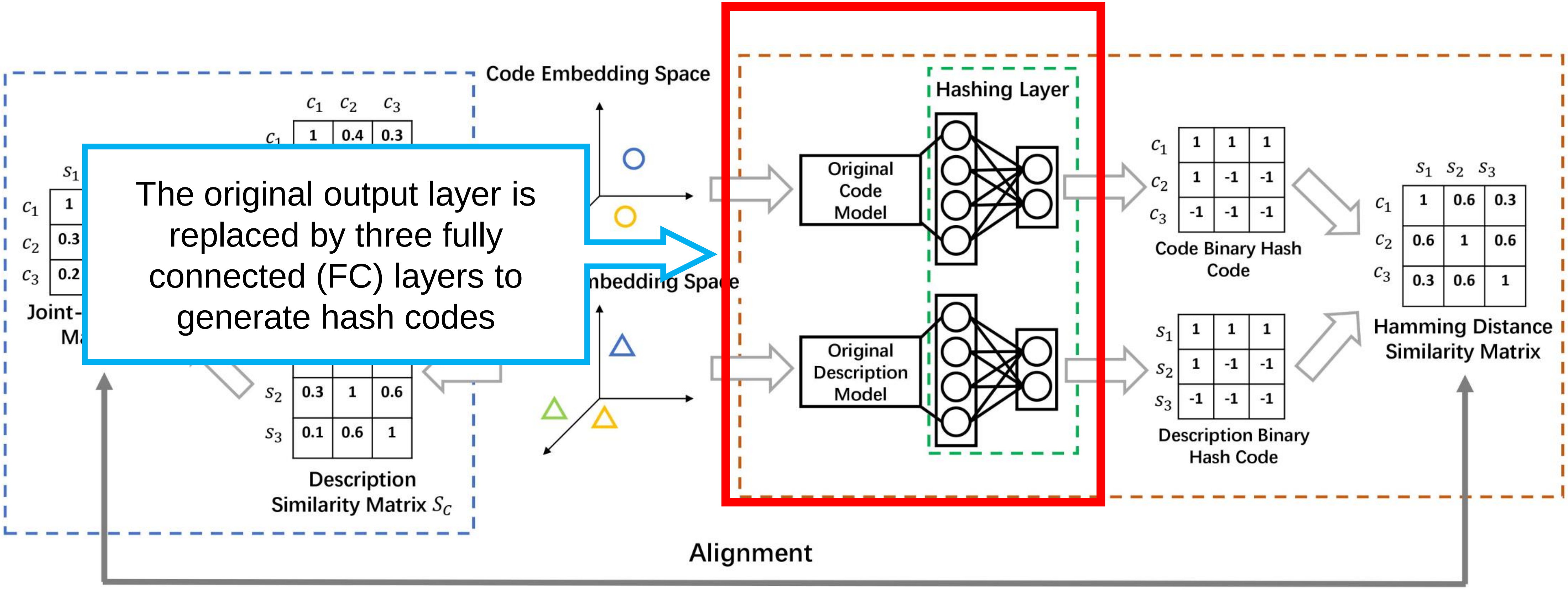
Involves $\tilde{S}\tilde{S}^T$ to describe a high order neighborhood similarity information:

$$S = (1-\eta)\tilde{S} + \frac{\tilde{S}\tilde{S}^T}{m}$$

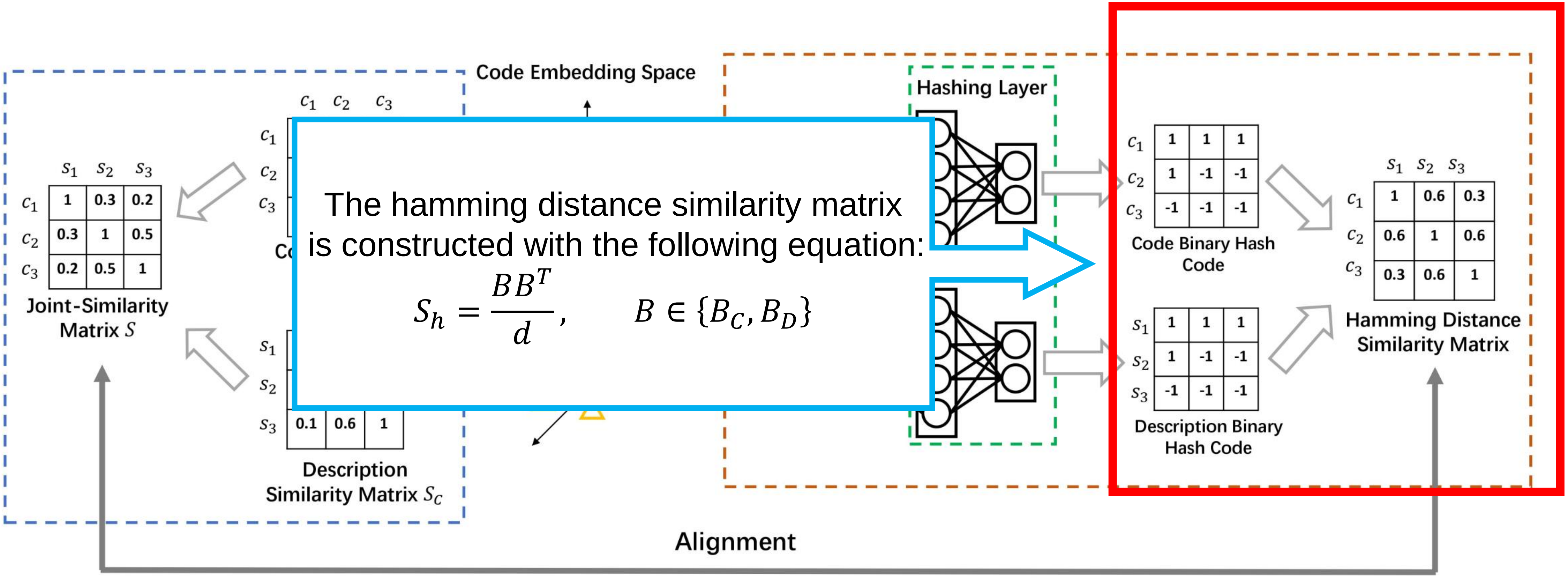
Sets diagonal elements of the joint-similarity matrix to be 1:

$$S_{F_{ij}} = \begin{cases} 1, & i = j \\ S_{ij}, & \text{otherwise} \end{cases}$$

Architecture of the hashing module



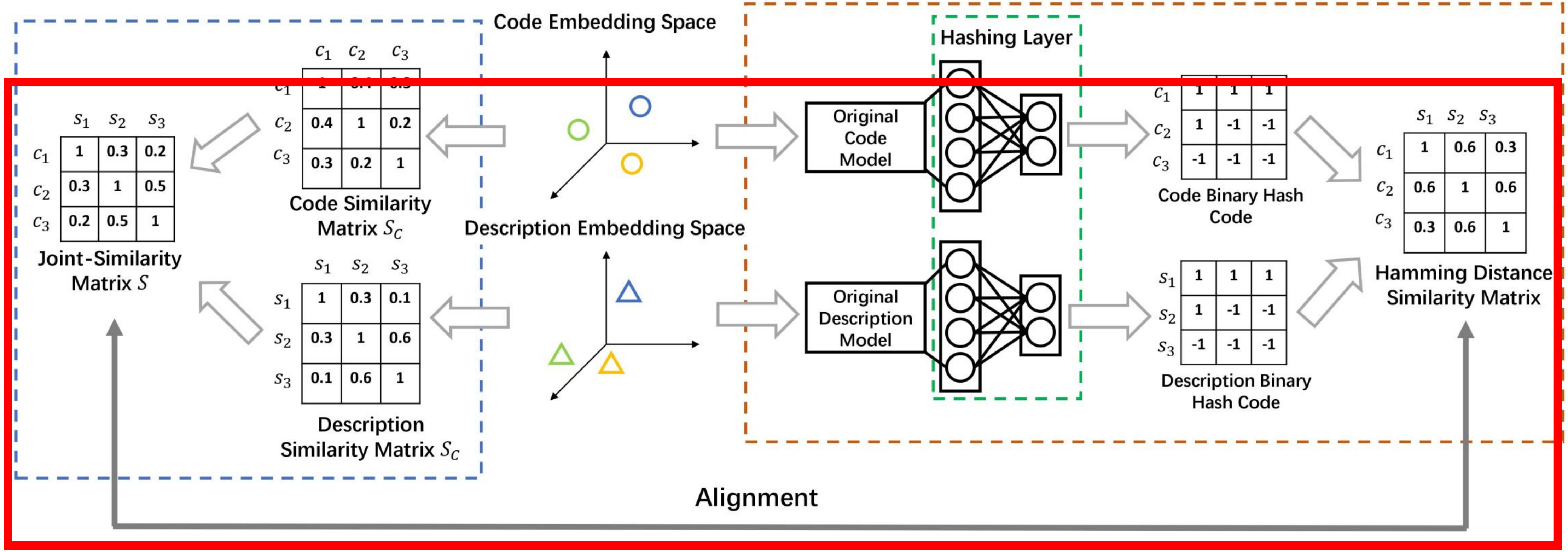
Architecture of the hashing module



The hamming distance similarity matrix is constructed with the following equation:

$$S_h = \frac{BB^T}{d}, \qquad B \in \{B_C, B_D\}$$

Architecture of the hashing module



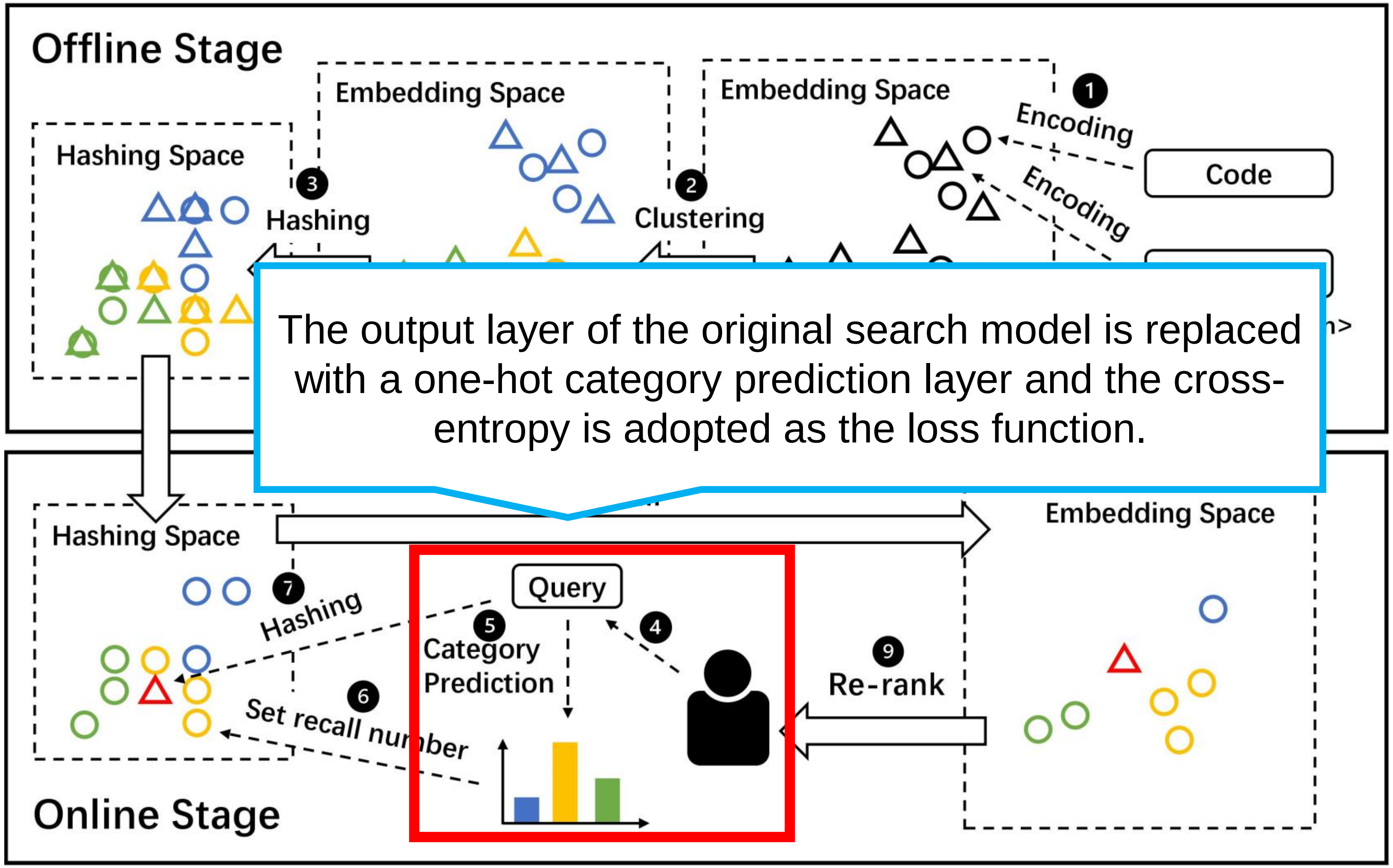
Hash code alignment

- The loss function of alignment is

$$\mathcal{L}(\theta) = \min_{B_C,B_D} \boxed{\left\| \min(\mu S_F, 1) - \frac{B_C B_D^T}{d} \right\|_F^2} + \lambda_1 \boxed{\left\| \min(\mu S_F, 1) - \frac{B_C B_C^T}{d} \right\|_F^2} + \lambda_2 \boxed{\left\| \min(\mu S_F, 1) - \frac{B_D B_D^T}{d} \right\|_F^2}$$

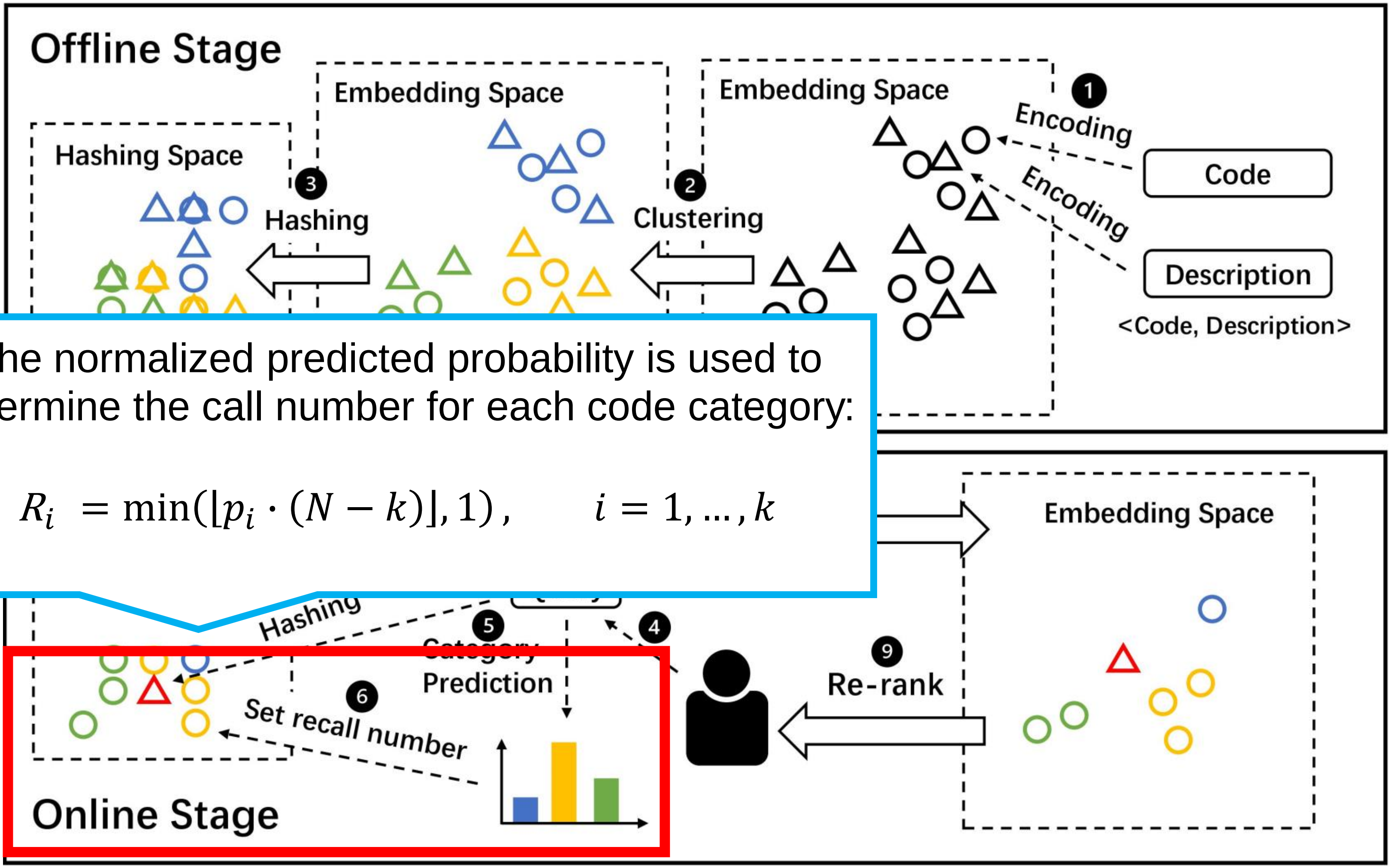
$$s.t.\ B_C, B_D \in \{-1, +1\}^{m\times d}$$

Framework – Online Stage



The output layer of the original search model is replaced with a one-hot category prediction layer and the cross-entropy is adopted as the loss function.

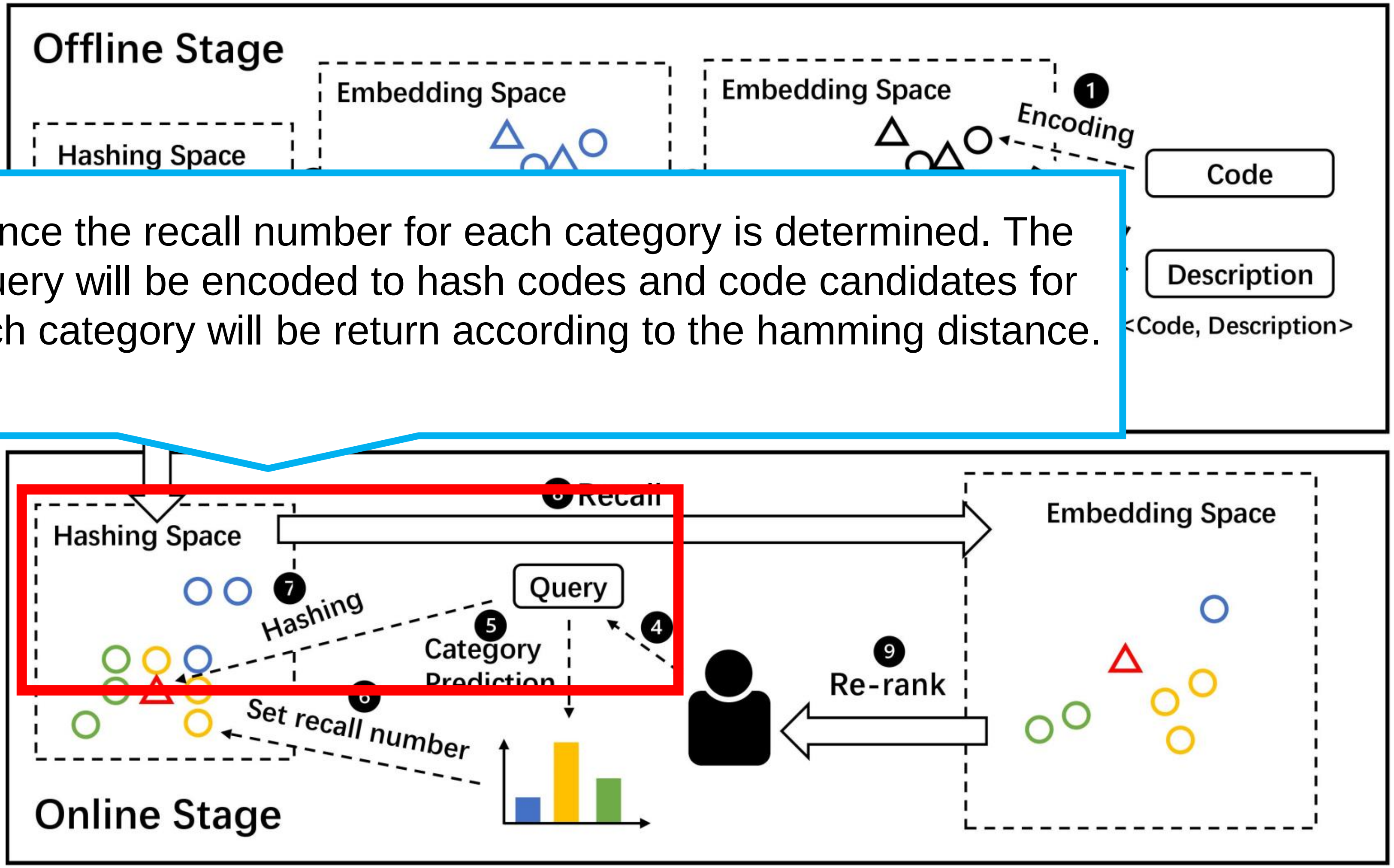
Framework



The normalized predicted probability is used to determine the call number for each code category:

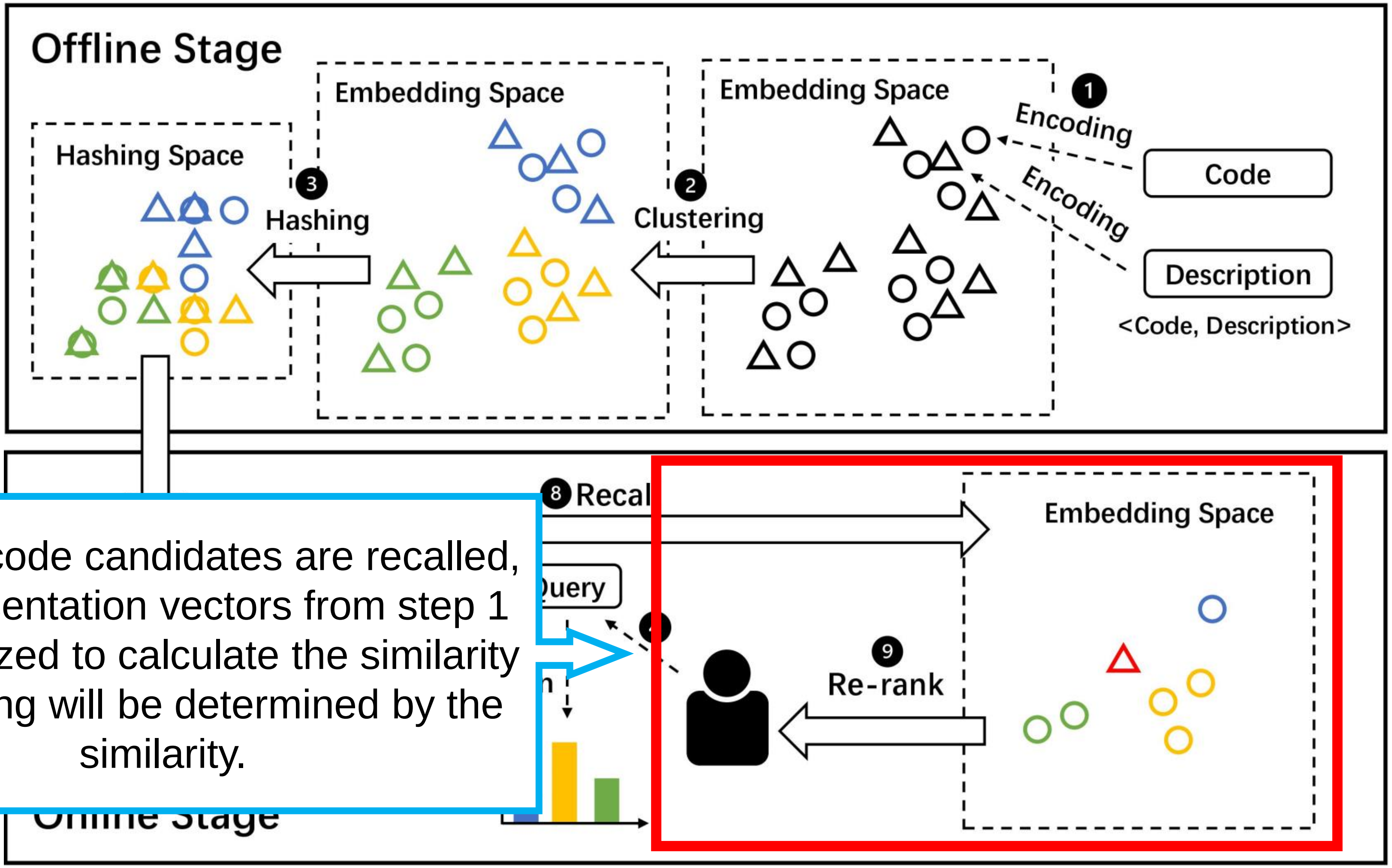
$$R_i = \min(\lfloor p_i \cdot (N-k) \rfloor, 1), \qquad i = 1, \dots, k$$

Framework



Once the recall number for each category is determined. The query will be encoded to hash codes and code candidates for each category will be return according to the hamming distance.

Framework



Once the code candidates are recalled, the representation vectors from step 1 will be utilized to calculate the similarity and ranking will be determined by the similarity.

Evaluation 1: Time efficiency

- Dataset: CodeSearchNet

	Python	Java
	Total Time	
CodeBERT	572.97s	247.78s
CoSHC	33.87s (↓94.09%)	15.78s (↓93.51%)
	(1) Vector Similarity Calculation	
CodeBERT	531.95s	234.08s
CoSHC	14.43s (↓97.29%)	7.25s (↓96.90%)
	(2) Array Sorting	
CodeBERT	41.02s	13.70s
CoSHC	19.44s (↓53.61%)	8.53s (↓37.74%)

COSHC can greatly reduce the cost of similarity calculation. Besides, COSHC can also reduce the cost of sorting by dividing a large code dataset into several small code datasets

Evaluation 2: Overall performance

Model	Python			Java		
	R@1	**R@5**	**R@10**	**R@1**	**R@5**	**R@10**
UNIF	0.071	0.173	0.236	0.084	0.193	0.254
CoSHC$_{\text{UNIF}}$	**0.072 (↑1.4%)**	**0.177 (↑2.3%)**	**0.241 (↑2.1%)**	**0.086 (↑2.4%)**	**0.198 (↑2.6%)**	**0.264 (↑3.9%)**
−w/o classification	0.071 (0.0%)	0.174 (↑0.6%)	0.236 (0.0%)	0.085 (↑1.2%)	0.193 (0.0%)	0.254 (0.0%)
−one classification	0.069 (↓2.8%)	0.163 (↓5.8%)	0.216 (↓8.5%)	0.083 (↓1.2%)	0.183 (↓5.2%)	0.236 (↓7.1%)
−ideal classification	0.077 (↑6.9%)	0.202 (↑16.8%)	0.277 (↑17.4%)	0.093 (↑10.7%)	0.222 (↑15.0%)	0.296 (↑16.5%)
RNN	0.111	0.253	0.333	0.073	0.184	0.250
CoSHC$_{\text{RNN}}$	**0.112 (↑0.9%)**	**0.259 (↑2.4%)**	**0.343 (↑5.0%)**	**0.076 (↑4.1%)**	**0.194 (↑5.4%)**	**0.265 (↑6.0%)**
−w/o classification	**0.112 (↑0.9%)**	0.254 (↑0.4%)	0.335 (↑0.6%)	0.073 (0.0%)	0.186 (↑1.1%)	0.253 (↑1.2%)
−one classification	**0.112 (↑0.9%)**	0.243 (↓4.0%)	0.311 (↓6.6%)	0.075 (↑2.7%)	0.182 (↓1.1%)	0.240 (↓4.0%)
−ideal classification	0.123 (↑10.8%)	0.289 (↑14.2%)	0.385 (↑15.6%)	0.084 (↑15.1%)	0.221 (↑20.1%)	0.302 (↑20.8%)
CodeBERTa	0.124	0.250	0.314	0.089	0.203	0.264
CoSHC$_{\text{CodeBERTa}}$	**0.123 (↓0.8%)**	**0.247 (↓1.2%)**	**0.309 (↓1.6%)**	**0.090 (↑1.1%)**	**0.210 (↑3.4%)**	**0.272 ((↑3.0%)**
−w/o classification	0.122 (↓1.6%)	0.242 (↓3.2%)	0.302 (↓3.8%)	0.089 (0.0%)	0.201 (↓1.0%)	0.258 (↓2.3%)
−one classification	0.116 (↓6.5%)	0.221 (↓11.6%)	0.271 (↓13.7%)	0.085 (↓4.5%)	0.189 (↓6.9%)	0.238 (↓9.8%)
−ideal classification	0.135 (↑8.9%)	0.276 (↑10.4%)	0.346 (↑10.2%)	0.100 (↑12.4%)	0.235 (↑15.8%)	0.305 (↑15.5%)
CodeBERT	0.451	0.683	0.759	0.319	0.537	0.608
CoSHC$_{\text{CodeBERT}}$	**0.451 (0.0%)**	**0.679 (↓0.6%)**	**0.750 (↓1.2%)**	**0.318 (↓0.3%)**	**0.533 (↓0.7%)**	**0.602 (↓1.0%)**
−w/o classification	0.449 (↓0.4%)	0.673 (↓1.5%)	0.742 (↓2.2%)	0.316 (↓0.9%)	0.527 (↓1.9%)	0.593 (↓2.5%)
−one classification	0.425 (↓5.8%)	0.613 (↓10.2%)	0.665 (↓12.4%)	0.304 (↓4.7%)	0.483 (↓10.1%)	0.532 (↓12.5%)
−ideal classification	0.460 (↑2.0%)	0.703 (↑2.9%)	0.775 (↑2.1%)	0.329 (↑3.1%)	0.555 (↑3.4%)	0.627 (↑3.1%)
GraphCodeBERT	0.485	0.726	0.792	0.353	0.571	0.640
CoSHC$_{\text{GraphCodeBERT}}$	**0.483 (↓0.4%)**	**0.719 (↓1.0%)**	**0.782 (↓1.3%)**	**0.350 (↓0.8%)**	**0.561 (↓1.8%)**	**0.625 (↓2.3%)**
−w/o classification	0.481 (↓0.8%)	0.713 (↓1.8%)	0.774 (↓2.3%)	0.347 (↓1.7%)	0.553 (↓3.2%)	0.616 (↓3.7%)
−one classification	0.459 (↓5.4%)	0.653 (↓10.1%)	0.698 (↓11.9%)	0.329 (↓7.8%)	0.505 (↓11.6%)	0.551 (↓13.9%)
−ideal classification	0.494 (↑1.9%)	0.741 (↑2.1%)	0.803 (↑1.4%)	0.361 (↑2.3%)	0.585 (↑2.5%)	0.649 (↑1.4%)

CoSHC can retain original model performance well.

(e.g., at least 99.2%, 98.2% and 97.7% accuracy of GraphCodeBERT on Java in terms of R@1, R@5 and R@10, respectively).

Evaluation 2: Overall performance

Model	**Python**			**Java**		
	R@1	**R@5**	**R@10**	**R@1**	**R@5**	**R@10**
UNIF	0.071	0.173	0.236	0.084	0.193	0.254
CoSHC$_{\mathrm{UNIF}}$	**0.072 (↑1.4%)**	**0.177 (↑2.3%)**	**0.241 (↑2.1%)**	**0.086 (↑2.4%)**	**0.198 (↑2.6%)**	**0.264 (↑3.9%)**
−w/o classification	0.071 (0.0%)	0.174 (↑0.6%)	0.236 (0.0%)	0.085 (↑1.2%)	0.193 (0.0%)	0.254 (0.0%)
−one classification	0.069 (↓2.8%)	0.163 (↓5.8%)	0.216 (↓8.5%)	0.083 (↓1.2%)	0.183 (↓5.2%)	0.236 (↓7.1%)
−ideal classification	0.077 (↑6.9%)	0.202 (↑16.8%)	0.277 (↑17.4%)	0.093 (↑10.7%)	0.222 (↑15.0%)	0.296 (↑16.5%)
RNN	0.111	0.253	0.333	0.073	0.184	0.250
CoSHC$_{\mathrm{RNN}}$	**0.112 (↑0.9%)**	**0.259 (↑2.4%)**	**0.343 (↑5.0%)**	**0.076 (↑4.1%)**	**0.194 (↑5.4%)**	**0.265 (↑6.0%)**
−w/o classification	**0.112 (↑0.9%)**	0.254 (↑0.4%)	0.335 (↑0.6%)	0.073 (0.0%)	0.186 (↑1.1%)	0.253 (↑1.2%)
−one classification	**0.112 (↑0.9%)**	0.243 (↓4.0%)	0.311 (↓6.6%)	0.075 (↑2.7%)	0.182 (↓1.1%)	0.240 (↓4.0%)
−ideal classification	0.123 (↑10.8%)	0.289 (↑14.2%)	0.385 (↑15.6%)	0.084 (↑15.1%)	0.221 (↑20.1%)	0.302 (↑20.8%)
CodeBERTa	0.124	0.250	0.314	0.089	0.203	0.264
CoSHC$_{\mathrm{CodeBERTa}}$	**0.123 (↓0.8%)**	**0.247 (↓1.2%)**	**0.309 (↓1.6%)**	**0.090 (↑1.1%)**	**0.210 (↑3.4%)**	**0.272 (↑3.0%)**
−w/o classification	0.122 (↓1.6%)	0.242 (↓3.2%)	0.302 (↓3.8%)	0.089 (0.0%)	0.201 (↓1.0%)	0.258 (↓2.3%)
−one classification	0.116 (↓6.5%)	0.221 (↓11.6%)	0.271 (↓13.7%)	0.085 (↓4.5%)	0.189 (↓6.9%)	0.238 (↓9.8%)
−ideal classification	0.135 (↑8.9%)	0.276 (↑10.4%)	0.346 (↑10.2%)	0.100 (↑12.4%)	0.235 (↑15.8%)	0.305 (↑15.5%)
CodeBERT	0.451	0.683	0.759	0.319	0.537	0.608
CoSHC$_{\mathrm{CodeBERT}}$	**0.451 (0.0%)**	**0.679 (↓0.6%)**	**0.750 (↓1.2%)**	**0.318 (↓0.3%)**	**0.533 (↓0.7%)**	**0.602 (↓1.0%)**
−w/o classification	0.449 (↓0.4%)	0.673 (↓1.5%)	0.742 (↓2.2%)	0.316 (↓0.9%)	0.527 (↓1.9%)	0.593 (↓2.5%)
−one classification	0.425 (↓5.8%)	0.613 (↓10.2%)	0.665 (↓12.4%)	0.304 (↓4.7%)	0.483 (↓10.1%)	0.532 (↓12.5%)
−ideal classification	0.460 (↑2.0%)	0.703 (↑2.9%)	0.775 (↑2.1%)	0.329 (↑3.1%)	0.555 (↑3.4%)	0.627 (↑3.1%)
GraphCodeBERT	0.485	0.726	0.792	0.353	0.571	0.640
CoSHC$_{\mathrm{GraphCodeBERT}}$	**0.483 (↓0.4%)**	**0.719 (↓1.0%)**	**0.782 (↓1.3%)**	**0.350 (↓0.8%)**	**0.561 (↓1.8%)**	**0.625 (↓2.3%)**
−w/o classification	0.481 (↓0.8%)	0.713 (↓1.8%)	0.774 (↓2.3%)	0.347 (↓1.7%)	0.553 (↓3.2%)	0.616 (↓3.7%)
−one classification	0.459 (↓5.4%)	0.653 (↓10.1%)	0.698 (↓11.9%)	0.329 (↓7.8%)	0.505 (↓11.6%)	0.551 (↓13.9%)
−ideal classification	0.494 (↑1.9%)	0.741 (↑2.1%)	0.803 (↑1.4%)	0.361 (↑2.3%)	0.585 (↑2.5%)	0.649 (↑1.4%)

CoSHC even outperforms the original baseline model when the performance of the baselines are not so good.

Evaluation 2: Effectiveness of Code Classification

Model	Python			Java		
	R@1	R@5	R@10	R@1	R@5	R@10
UNIF	0.071	0.173	0.236	0.084	0.193	0.254
CoSHC$_{\mathrm{UNIF}}$	**0.072 (↑1.4%)**	**0.177 (↑2.3%)**	**0.241 (↑2.1%)**	**0.086 (↑2.4%)**	**0.198 (↑2.6%)**	**0.264 (↑3.9%)**
−w/o classification	0.071 (0.0%)	0.174 (↑0.6%)	0.236 (0.0%)	0.085 (↑1.2%)	0.193 (0.0%)	0.254 (0.0%)
−one classification	0.069 (↓2.8%)	0.163 (↓5.8%)	0.216 (↓8.5%)	0.083 (↓1.2%)	0.183 (↓5.2%)	0.236 (↓7.1%)
−ideal classification	0.077 (↑6.9%)	0.202 (↑16.8%)	0.277 (↑17.4%)	0.093 (↑10.7%)	0.222 (↑15.0%)	0.296 (↑16.5%)
RNN	0.111	0.253	0.333	0.073	0.184	0.250
CoSHC$_{\mathrm{RNN}}$	**0.112 (↑0.9%)**	**0.259 (↑2.4%)**	**0.343 (↑5.0%)**	**0.076 (↑4.1%)**	**0.194 (↑5.4%)**	**0.265 (↑6.0%)**
−w/o classification	**0.112 (↑0.9%)**	0.254 (↑0.4%)	0.335 (↑0.6%)	0.073 (0.0%)	0.186 (↑1.1%)	0.253 (↑1.2%)
−one classification	**0.112 (↑0.9%)**	0.243 (↓4.0%)	0.311 (↓6.6%)	0.075 (↑2.7%)	0.182 (↓1.1%)	0.240 (↓4.0%)
−ideal classification	0.123 (↑10.8%)	0.289 (↑14.2%)	0.385 (↑15.6%)	0.084 (↑15.1%)	0.221 (↑20.1%)	0.302 (↑20.8%)
CodeBERTa	0.124	0.250	0.314	0.089	0.203	0.264
CoSHC$_{\mathrm{CodeBERTa}}$	**0.123 (↓0.8%)**	**0.247 (↓1.2%)**	**0.309 (↓1.6%)**	**0.090 (↑1.1%)**	**0.210 (↑3.4%)**	**0.272 ((↑3.0%)**
−w/o classification	0.122 (↓1.6%)	0.242 (↓3.2%)	0.302 (↓3.8%)	0.089 (0.0%)	0.201 (↓1.0%)	0.258 (↓2.3%)
−one classification	0.116 (↓6.5%)	0.221 (↓11.6%)	0.271 (↓13.7%)	0.085 (↓4.5%)	0.189 (↓6.9%)	0.238 (↓9.8%)
−ideal classification	0.135 (↑8.9%)	0.276 (↑10.4%)	0.346 (↑10.2%)	0.100 (↑12.4%)	0.235 (↑15.8%)	0.305 (↑15.5%)
CodeBERT	0.451	0.683	0.759	0.319	0.537	0.608
CoSHC$_{\mathrm{CodeBERT}}$	**0.451 (0.0%)**	**0.679 (↓0.6%)**	**0.750 (↓1.2%)**	**0.318 (↓0.3%)**	**0.533 (↓0.7%)**	**0.602 (↓1.0%)**
−w/o classification	0.449 (↓0.4%)	0.673 (↓1.5%)	0.742 (↓2.2%)	0.316 (↓0.9%)	0.527 (↓1.9%)	0.593 (↓2.5%)
−one classification	0.425 (↓5.8%)	0.613 (↓10.2%)	0.665 (↓12.4%)	0.304 (↓4.7%)	0.483 (↓10.1%)	0.532 (↓12.5%)
−ideal classification	0.460 (↑2.0%)	0.703 (↑2.9%)	0.775 (↑2.1%)	0.329 (↑3.1%)	0.555 (↑3.4%)	0.627 (↑3.1%)
GraphCodeBERT	0.485	0.726	0.792	0.353	0.571	0.640
CoSHC$_{\mathrm{GraphCodeBERT}}$	**0.483 (↓0.4%)**	**0.719 (↓1.0%)**	**0.782 (↓1.3%)**	**0.350 (↓0.8%)**	**0.561 (↓1.8%)**	**0.625 (↓2.3%)**
−w/o classification	0.481 (↓0.8%)	0.713 (↓1.8%)	0.774 (↓2.3%)	0.347 (↓1.7%)	0.553 (↓3.2%)	0.616 (↓3.7%)
−one classification	0.459 (↓5.4%)	0.653 (↓10.1%)	0.698 (↓11.9%)	0.329 (↓7.8%)	0.505 (↓11.6%)	0.551 (↓13.9%)
−ideal classification	0.494 (↑1.9%)	0.741 (↑2.1%)	0.803 (↑1.4%)	0.361 (↑2.3%)	0.585 (↑2.5%)	0.649 (↑1.4%)

With the ideal category labels, CoSHC can even outperform all baseline models.

Evaluation 3: Ablation Study

Model	**Python**			**Java**		
	R@1	**R@5**	**R@10**	**R@1**	**R@5**	**R@10**
UNIF	0.071	0.173	0.236	0.084	0.193	0.254
$\text{CoSHC}_{\text{UNIF}}$	**0.072 (↑1.4%)**	**0.177 (↑2.3%)**	**0.241 (↑2.1%)**	**0.086 (↑2.4%)**	**0.198 (↑2.6%)**	**0.264 (↑3.9%)**
−w/o classification	0.071 (0.0%)	0.174 (↑0.6%)	0.236 (0.0%)	0.085 (↑1.2%)	0.193 (0.0%)	0.254 (0.0%)
−one classification	0.069 (↓2.8%)	0.163 (↓5.8%)	0.216 (↓8.5%)	0.083 (↓1.2%)	0.183 (↓5.2%)	0.236 (↓7.1%)
−ideal classification	0.077 (↑6.9%)	0.202 (↑16.8%)	0.277 (↑17.4%)	0.093 (↑10.7%)	0.222 (↑15.0%)	0.296 (↑16.5%)
RNN	0.111	0.253	0.333	0.073	0.184	0.250
$\text{CoSHC}_{\text{RNN}}$	**0.112 (↑0.9%)**	**0.259 (↑2.4%)**	**0.343 (↑5.0%)**	**0.076 (↑4.1%)**	**0.194 (↑5.4%)**	**0.265 (↑6.0%)**
−w/o classification	**0.112 (↑0.9%)**	0.254 (↑0.4%)	0.335 (↑0.6%)	0.073 (0.0%)	0.186 (↑1.1%)	0.253 (↑1.2%)
−one classification	**0.112 (↑0.9%)**	0.243 (↓4.0%)	0.311 (↓6.6%)	0.075 (↑2.7%)	0.182 (↓1.1%)	0.240 (↓4.0%)
−ideal classification	0.123 (↑10.8%)	0.289 (↑14.2%)	0.385 (↑15.6%)	0.084 (↑15.1%)	0.221 (↑20.1%)	0.302 (↑20.8%)
CodeBERTa	0.124	0.250	0.314	0.089	0.203	0.264
$\text{CoSHC}_{\text{CodeBERTa}}$	**0.123 (↓0.8%)**	**0.247 (↓1.2%)**	**0.309 (↓1.6%)**	**0.090 (↑1.1%)**	**0.210 (↑3.4%)**	**0.272 ((↑3.0%)**
−w/o classification	0.122 (↓1.6%)	0.242 (↓3.2%)	0.302 (↓3.8%)	0.089 (0.0%)	0.201 (↓1.0%)	0.258 (↓2.3%)
−one classification	0.116 (↓6.5%)	0.221 (↓11.6%)	0.271 (↓13.7%)	0.085 (↓4.5%)	0.189 (↓6.9%)	0.238 (↓9.8%)
−ideal classification	0.135 (↑8.9%)	0.276 (↑10.4%)	0.346 (↑10.2%)	0.100 (↑12.4%)	0.235 (↑15.8%)	0.305 (↑15.5%)
CodeBERT	0.451	0.683	0.759	0.319	0.537	0.608
$\text{CoSHC}_{\text{CodeBERT}}$	**0.451 (0.0%)**	**0.679 (↓0.6%)**	**0.750 (↓1.2%)**	**0.318 (↓0.3%)**	**0.533 (↓0.7%)**	**0.602 (↓1.0%)**
−w/o classification	0.449 (↓0.4%)	0.673 (↓1.5%)	0.742 (↓2.2%)	0.316 (↓0.9%)	0.527 (↓1.9%)	0.593 (↓2.5%)
−one classification	0.425 (↓5.8%)	0.613 (↓10.2%)	0.665 (↓12.4%)	0.304 (↓4.7%)	0.483 (↓10.1%)	0.532 (↓12.5%)
−ideal classification	0.460 (↑2.0%)	0.703 (↑2.9%)	0.775 (↑2.1%)	0.329 (↑3.1%)	0.555 (↑3.4%)	0.627 (↑3.1%)
GraphCodeBERT	0.485	0.726	0.792	0.353	0.571	0.640
$\text{CoSHC}_{\text{GraphCodeBERT}}$	**0.483 (↓0.4%)**	**0.719 (↓1.0%)**	**0.782 (↓1.3%)**	**0.350 (↓0.8%)**	**0.561 (↓1.8%)**	**0.625 (↓2.3%)**
−w/o classification	0.481 (↓0.8%)	0.713 (↓1.8%)	0.774 (↓2.3%)	0.347 (↓1.7%)	0.553 (↓3.2%)	0.616 (↓3.7%)
−one classification	0.459 (↓5.4%)	0.653 (↓10.1%)	0.698 (↓11.9%)	0.329 (↓7.8%)	0.505 (↓11.6%)	0.551 (↓13.9%)
−ideal classification	0.494 (↑1.9%)	0.741 (↑2.1%)	0.803 (↑1.4%)	0.361 (↑2.3%)	0.585 (↑2.5%)	0.649 (↑1.4%)

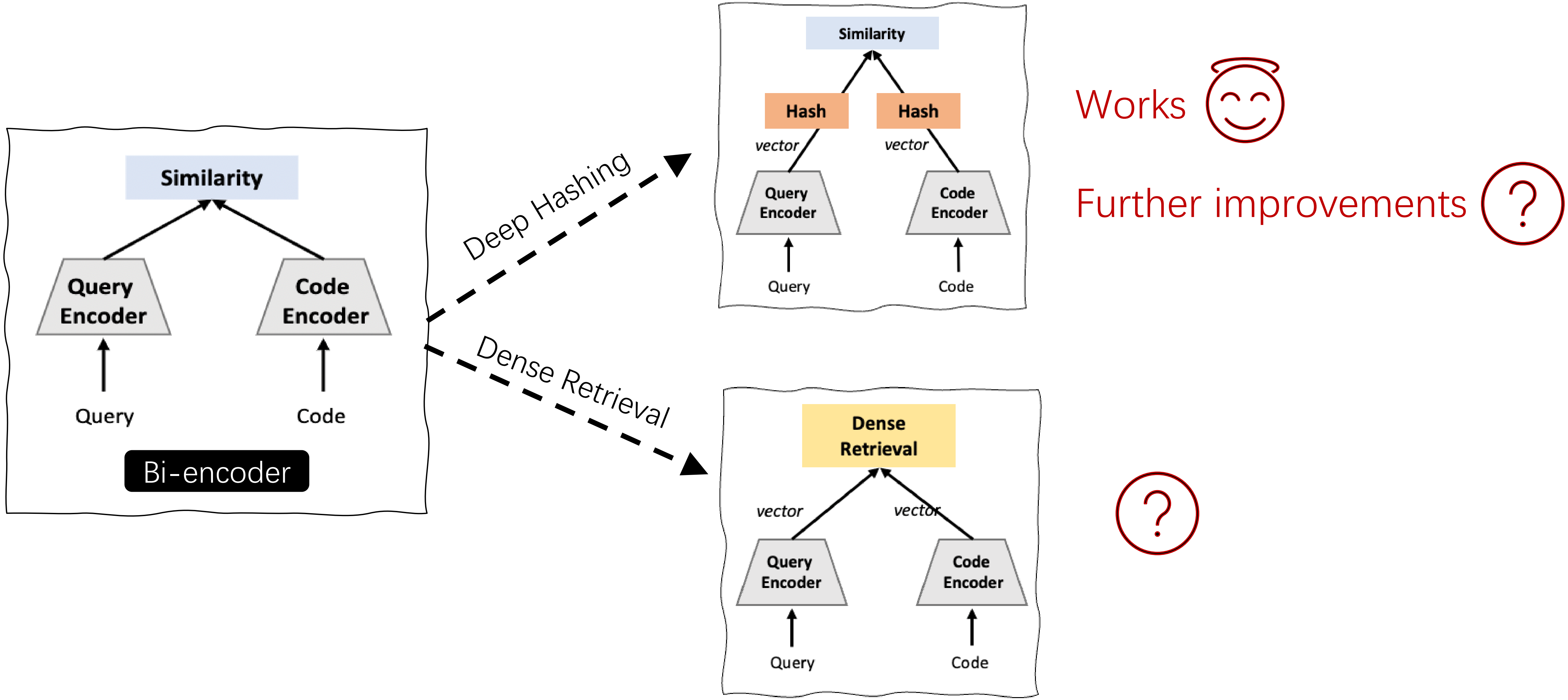
CoSHC outperforms all the variants

Evaluation 4: Classification Accuracy

Model	Python Acc.	Java Acc.
$\text{CoSHC}_{\text{UNIF}}$	0.558	0.545
$\text{CoSHC}_{\text{RNN}}$	0.610	0.535
$\text{CoSHC}_{\text{CodeBERTa}}$	0.591	0.571
$\text{CoSHC}_{\text{CodeBERT}}$	0.694	0.657
$\text{CoSHC}_{\text{GraphCodeBERT}}$	0.713	0.653

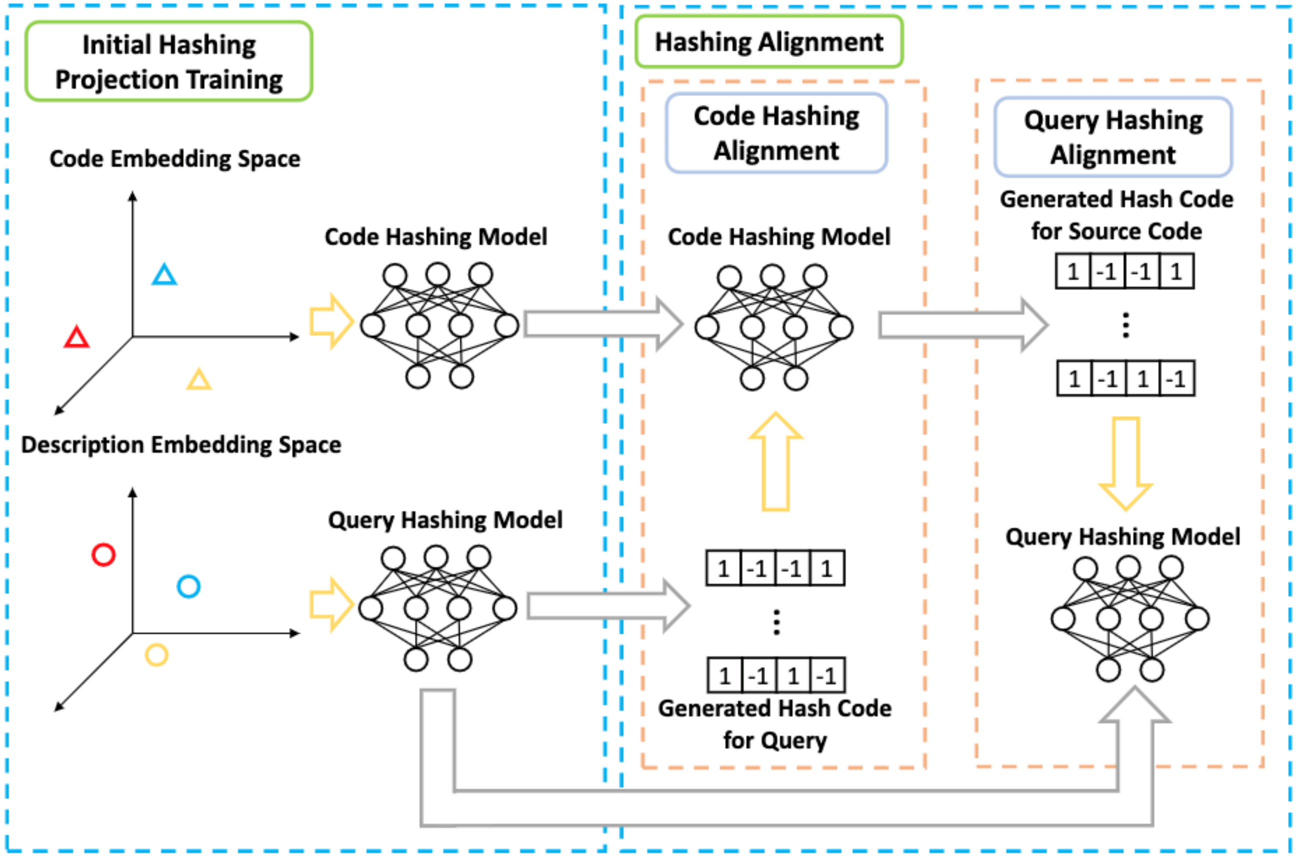
The retrieval accuracy of the original code search models is higher, the prediction accuracy also tends to be higher.

Ongoing/Future work



Onging/Future work

1. Improvement on CoSHC with segmented hashing

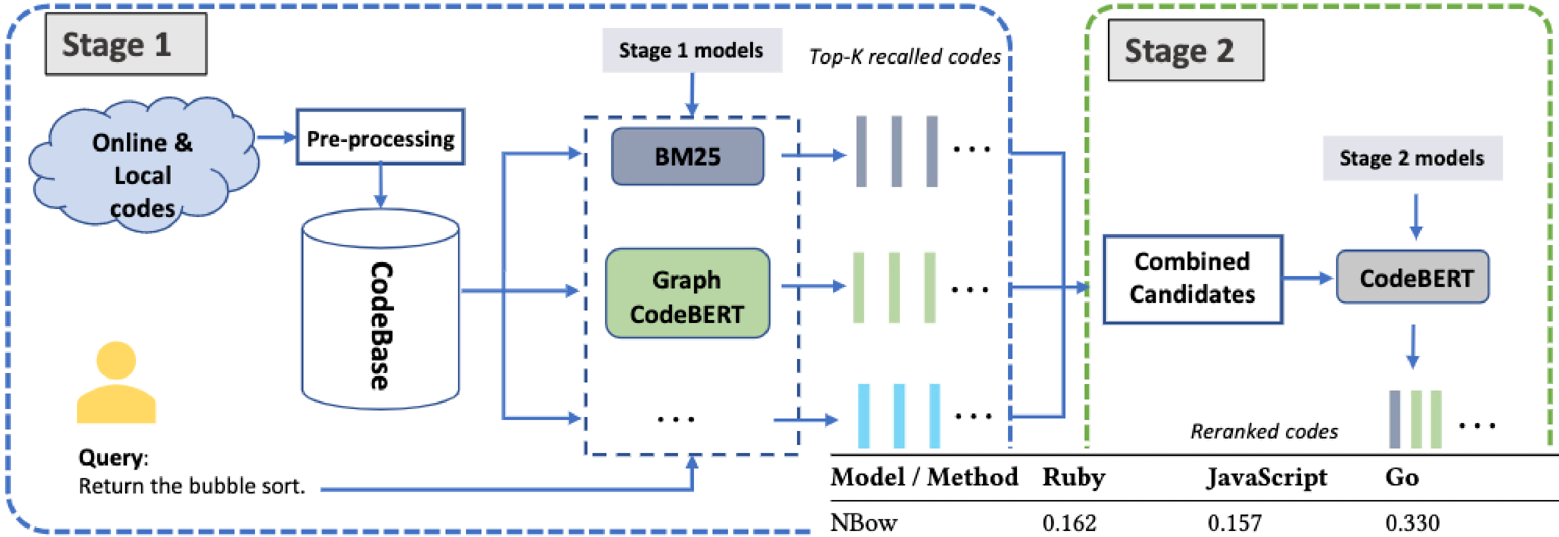


Promising preliminary results

Onging/Future work

2. Combination of IR & DL based approaches

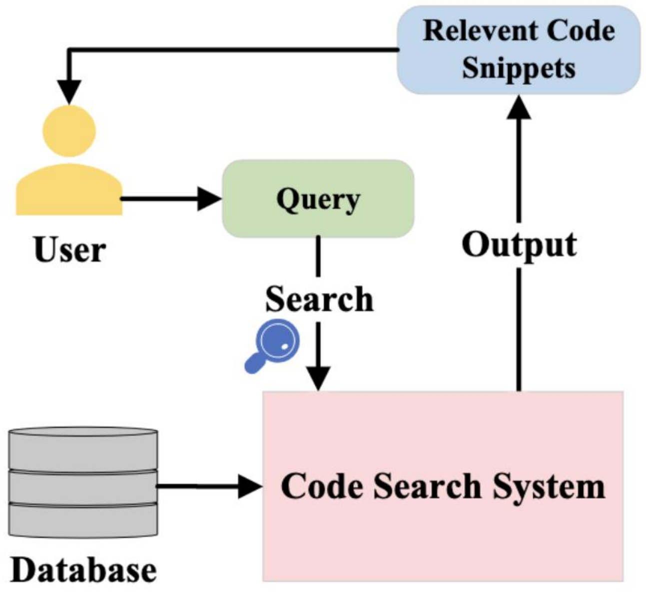
- Preliminary results are promising, both in accuracy and efficiency.



Model / Method	Ruby	JavaScript	Go	Python	Java	PHP	Overall
NBow	0.162	0.157	0.330	0.161	0.171	0.152	0.189
CNN	0.276	0.224	0.680	0.242	0.263	0.260	0.324
BiRNN	0.213	0.193	0.688	0.290	0.304	0.338	0.338
selfAtt	0.275	0.287	0.723	0.398	0.404	0.426	0.419
RoBERTa	0.587	0.517	0.850	0.587	0.599	0.560	0.617
RoBERTa (code)	0.628	0.562	0.859	0.610	0.620	0.579	0.643
CodeBERT	0.679	0.620	0.882	0.672	0.676	0.628	0.693
GraphCodeBERT	0.703	0.644	0.897	0.692	0.691	0.649	0.713
TOSS	**0.763** (8.9%↑)	**0.729** (13.7%↑)	**0.898** (0.01%↑)	**0.786** (14.0%↑)	**0.800** (16.2%↑)	**0.730** (12.8%↑)	**0.784** (10.4%↑)

Onging/Future work

- Multi-stage retrieval
- Better evaluation metrics for code search
- Larger and more practical datasets
- Human in the loop: Interactive multi-turn code search



Thanks!

Open for discussion and collaboration:

wangylin36@mail.sysu.edu.cn
https://yanlin.info