Effective API Recommendation without Historical Software Repositories

Xiaoyu Liu[1], LiGuo Huang[1], Vincent Ng[2]

[1] Dept. of Computer Science and Engineering, Southern Methodist University, Dallas, TX

[2] Human Language Technology Research Institute, University of Texas and Dallas, TX

Emails: {*xiaoyul, lghuang*}*@smu.edu* *vince@hlt.utdallas.edu*

Automated API Recommendation, Why?

- equals(Object anObject) : boolean - String
- length() : int - String
- equalsIgnoreCase(String anotherString) : boolean - String
- getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin) : void - String
- concat(String str) : String - String
- substring(int beginIndex, int endIndex) : String - String
- charAt(int index) : char - String
- chars() : IntStream - String
- codePointAt(int index) : int - String
- codePointBefore(int index) : int - String
- codePointCount(int beginIndex, int endIndex) : int - String
- codePoints() : IntStream - String
- compareTo(String anotherString) : int - String
- compareToIgnoreCase(String str) : int - String
- contains(CharSequence s) : boolean - String
- contentEquals(CharSequence cs) : boolean - String
- contentEquals(StringBuffer sb) : boolean - String
- endsWith(String suffix) : boolean - String
- getBytes() : byte[] - String
- getBytes(Charset charset) : byte[] - String
- getBytes(String charsetName) : byte[] - String
- getBytes(int srcBegin, int srcEnd, byte[] dst, int dstBegin) : void - String
- getClass() : Class<?> - Object
- hashCode() : int - String
- indexOf(int ch) : int - String

Existing API Recommendation Methods

Statistical Learning: *irrelevant* features (noises) and *overlapping* features

- Wing-Kwan Chan, Hong Cheng, and David Lo. 2012. Searching connected API subgraph via text phrases. In Proceedings of the 20th ACM SIGSOFT International Symposium on the Foundations of Software Engineering. ACM, 10.
- Muhammad Asaduzzaman, Chanchal K. Roy, Kevin A. Schneider, and Daqing Hou. 2016. A Simple, Efficient, Context-sensitive Approach for Code Completion. Journal of Software: Evolution and Process 28, 7 (2016), 512–541.

Code Structure: missing certain semantics of source code

- CollinMcMillan,DenysPoshyvanyk,andMarkGrechanik.2010.Recommending source code examples via API call usages and documentation. In Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering. ACM, 21–25.
- Evan Moritz, Mario Linares-Vásquez, Denys Poshyvanyk, Mark Grechanik, Collin McMillan, and Malcom Gethers. 2013. Export: Detecting and visualizin API usages in large source code repositories. In Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering. IEEE Press, 646–651.

Mining Historical Software Repositories: maintain tremendous amount of historical data

- MarcelBruch,MartinMonperrus,andMiraMezini.2009.Learningfromexamples to improve code completion systems. In Proceedings of the 7th ACM SIGSOFT Symposium on the Foundations of Software Engineering. ACM, 213–222.
- Romain Robbes and Michele Lanza. 2008. How program history can improve code completion. In Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering. IEEE Computer Society, 317–326.

State-of-the Art

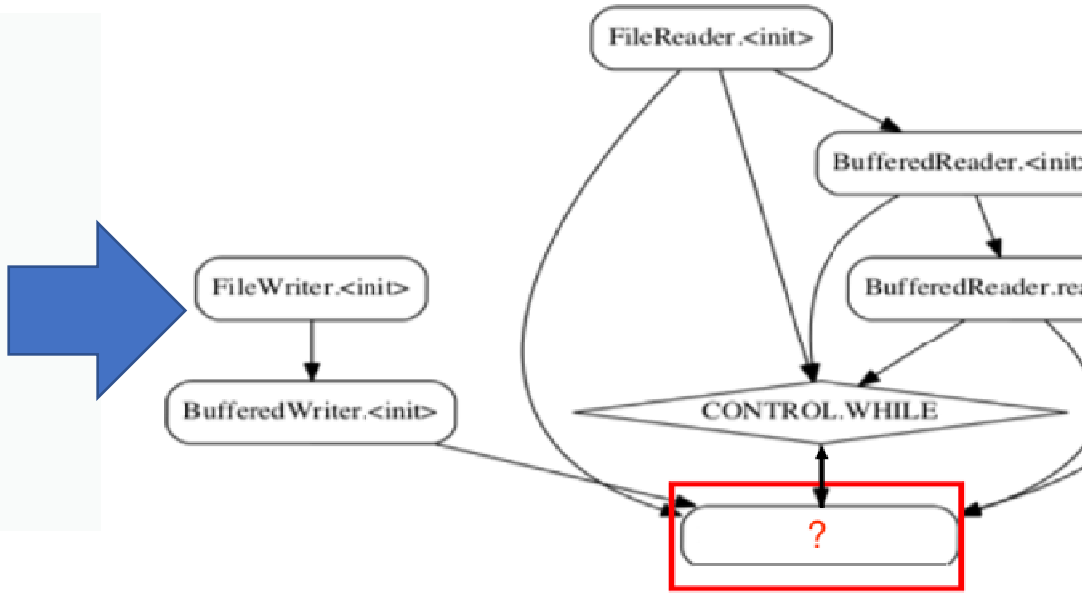
❖ ***Gralan*:** A. T. Nguyen and T. Nguyen. 2015. “Graph-based statistical language model for code”. In Proceedings of the 37th IEEE International Conference on Software Engineering. IEEE, 858–868.

❖ ***APIREC*:** A. Nguyen, M. Hilton, M. Codoban, H. A. Nguyen, L. Mast, E. Rademacher, T. Nguyen, and D. Dig. 2016. “API code recommendation using statistical learning from fine-grained changes”. In Proceedings of the 24th ACM SIGSOFT International Symposium on the Foundations of Software Engineering. ACM, 511–522.

How does *Gralan* Recommend APIs?

```
ublic void readAndWrite() throws IOException{
  BufferedReader in = new BufferedReader(new FileReader("Input.txt"));
  BufferedWriter out = new BufferedWriter(new FileWriter("Output.txt"));

  String i = null;
  char c;
  while((i = in.readLine())!=null){
      ?
  }
```



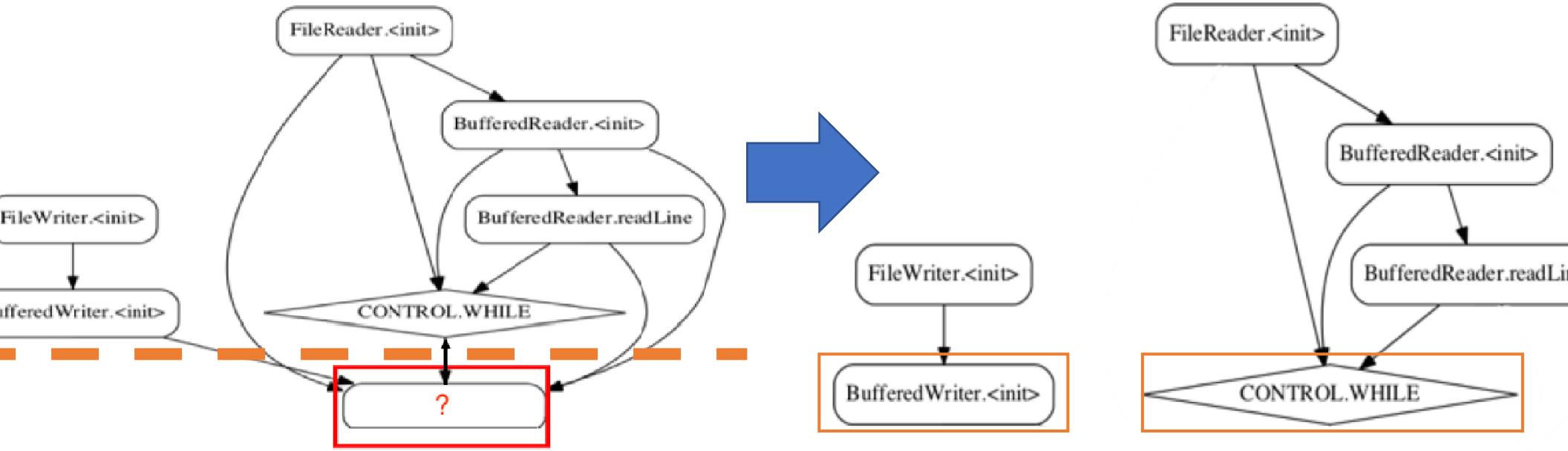
2 Extract Child Graph, Parent Graph and Subgraphs

3 Compute the probability of candidate APIs

4 Predict API wit the highest probability

roceeding Context: enerate API Usage raph

ow does *Gralan* Recommend APIs?



2

3

4

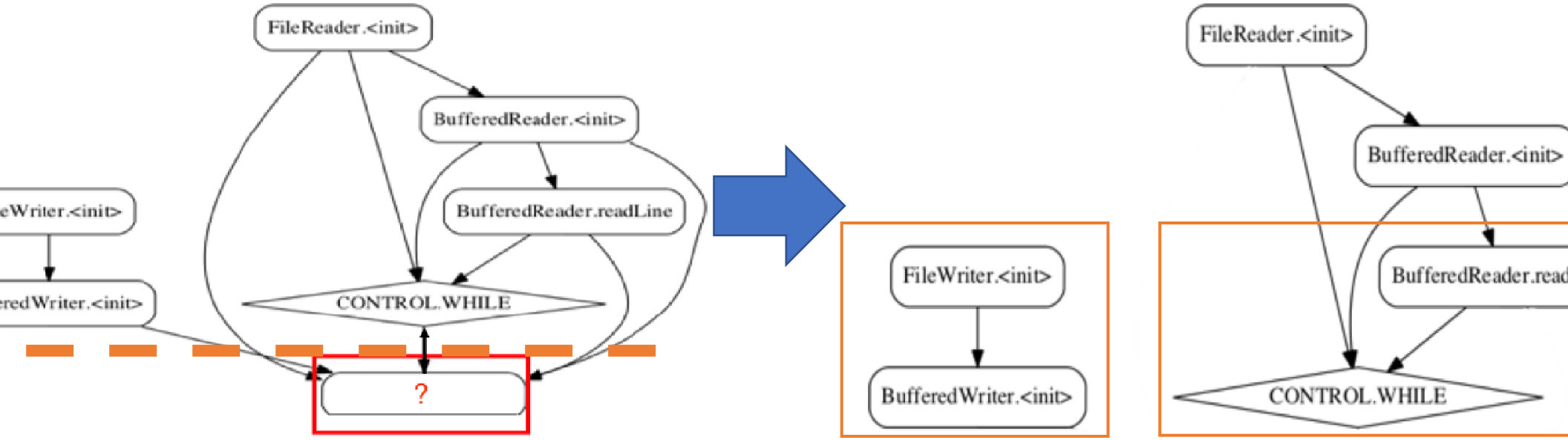
roceeding Context: enerate API Usage raph

Extract Child Graph, Parent Graph and Subgraphs

Compute the probability of candidate APIs

Predict API wit the highest probability

How does *Gralan* Recommend APIs?



2

3

4

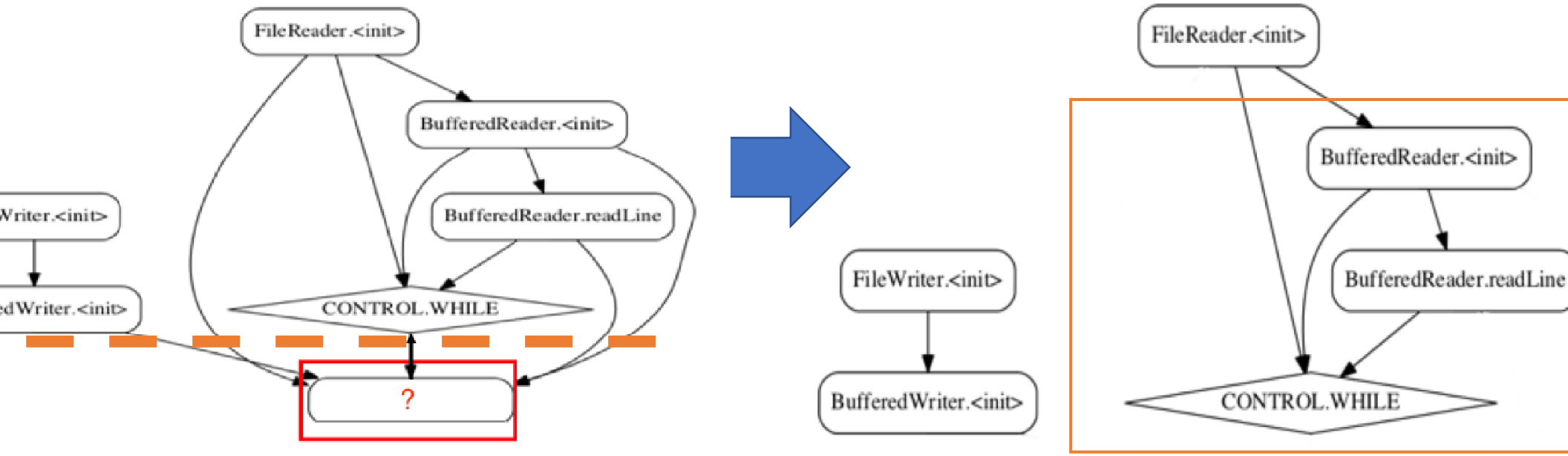
roceeding Context: enerate API Usage raph

Extract Child Graph, Parent Graph and Subgraphs

Compute the probability of candidate APIs

Predict API wit the highest probability

How does *Gralan* Recommend APIs?



2

3

4

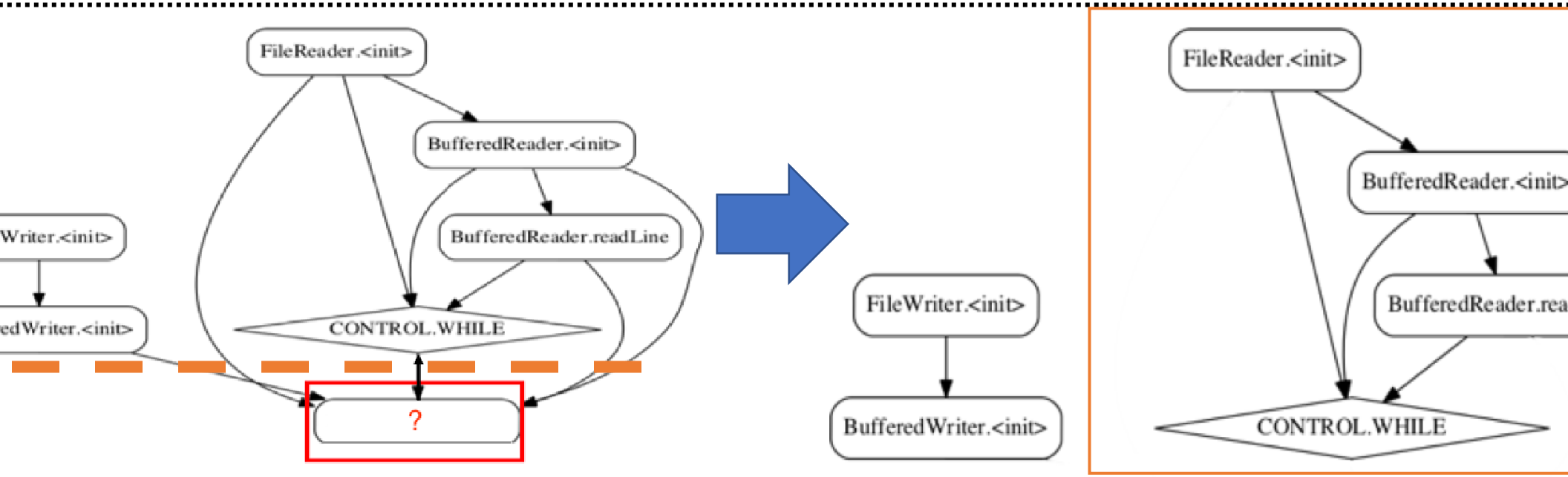
roceeding Context: enerate API Usage raph

Extract Child Graph, Parent Graph and Subgraphs

Compute the probability of candidate APIs

Predict API wit the highest probability

ow does *Gralan* Recommend APIs?



2

3

4

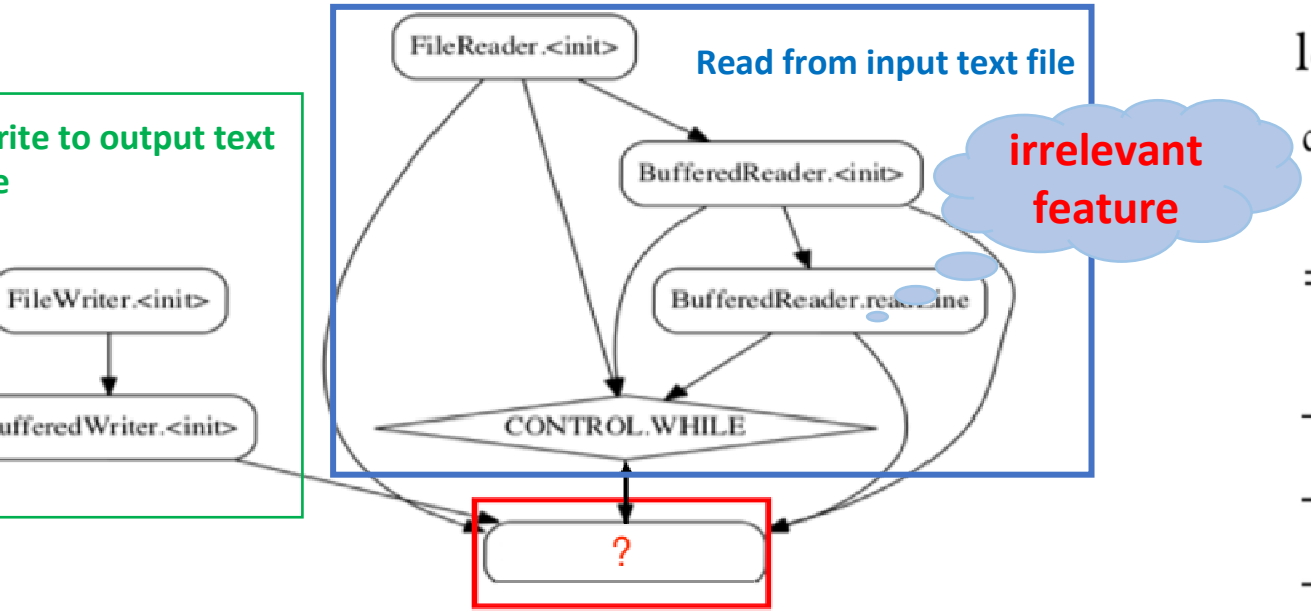
roceeding Context: enerate API Usage raph

Extract Child Graph, Parent Graph and Subgraphs

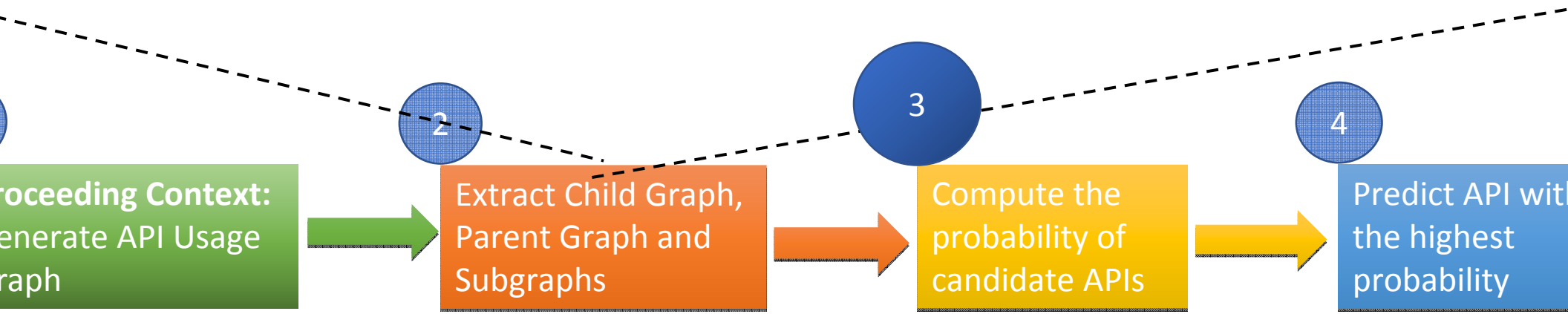
Compute the probability of candidate APIs

Predict API wit the highest probability

low does *Gralan* Recommend APIs?

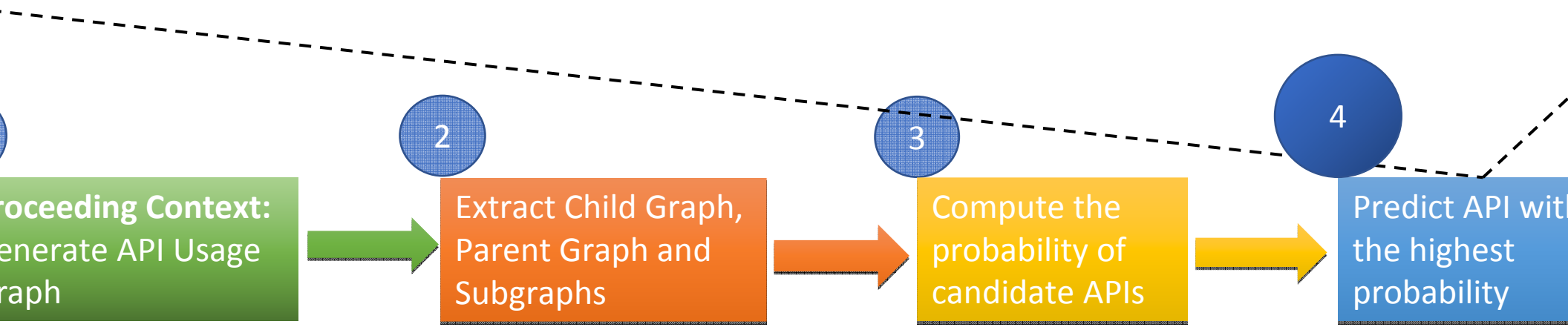


$$\log(Pr(C(g)|g_1, g_2, \ldots, g_n, g))$$
$$\propto \log(Pr(g_1|C(g))\ldots Pr(g_n|C(g))Pr(C(g$$
$$= \sum_{j=1}^{n} \log(\#methods(g_j, C(g)) + \alpha)$$
$$+ \log(\#methods(g, C(g)))$$
$$- (n-1)\log(\#methods(C(g)) + \alpha\#met$$
$$- \log(\#methods(g))$$



How does *Gralan* Recommend APIs?

g	C(g)	Candidate API	Score
Reader.<init>, BufferedReader.<init>, feredReader.readLine, CONTROL.WHILE	*FileReader.<init>, …, BufferedReader.close*	*BufferedReader.close (incorrect)*	**0.33**
	FileReader.<init>, …, CONTROL.WHILE, BufferedWriter.write	*BufferedWriter.write*	0.15
	…	…	…
feredWriter.<init>, CONTROL.WHILE	*BufferedWriter.<init>, CONTROL.WHILE, BufferedWriter.write*	*BufferedWriter.write*	0.25
	BufferedWriter.<init>, CONTROL.WHILE, BufferedReader.close	*BufferedReader.close*	0.02
	…	…	..



How does *APIREC* recommend APIs?

APIREC requires a long code change history of each project, which limits applicability to scenarios where long change history is unavailable or inaccessible.

APIREC uses all changes. BUT some them could be specific to a historica project and could therefore incur no the change patterns.

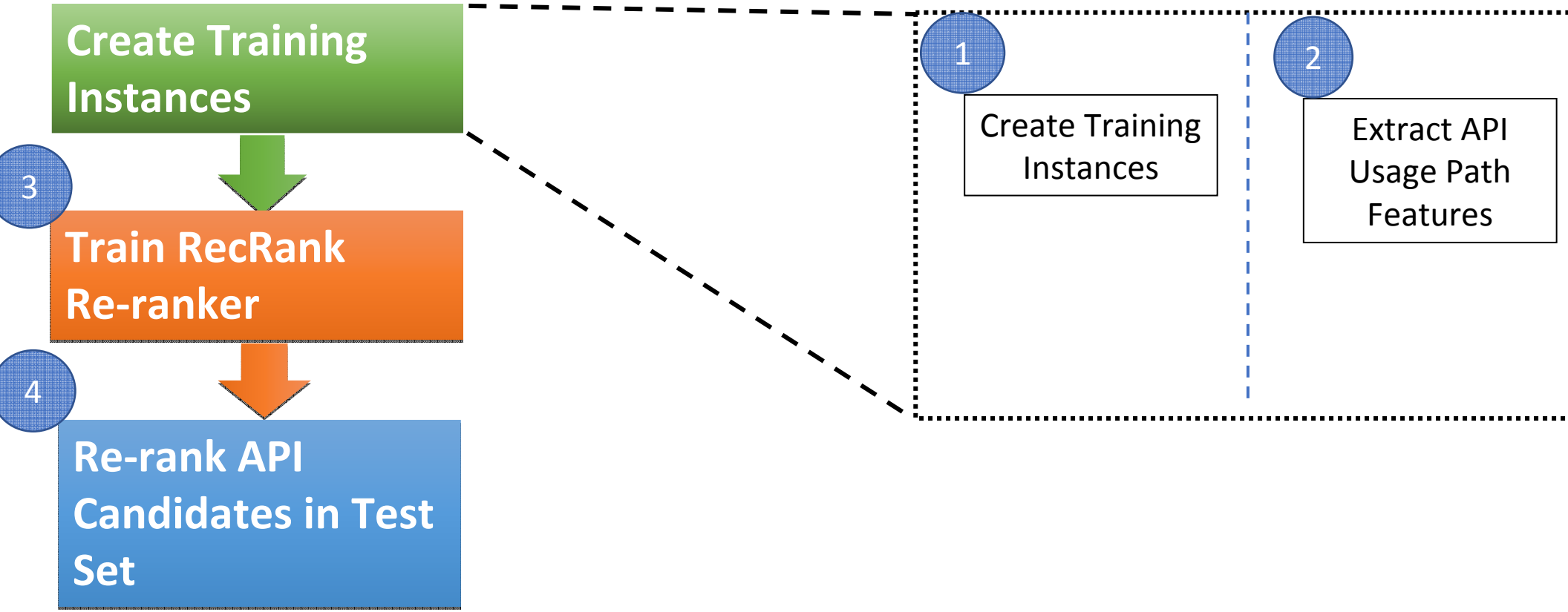
Objective and Major Contributions

Objective: To improve the **top-1 accuracy** of API recommendation.

RecRank: a novel discriminative ranking approach to automatically recommend top-1 APIs based on the top-10 API candidates suggested by *Gralan*.

Usage path features: a novel kind of features based on API usage paths

scriminative Re-Ranking for API Recommendation (*RecRa*



scriminative Re-Ranking for API Recommendation (*RecRa*

1

Create Training Instances

For each API recommendation point in the training set, training instances are created.

Create one training instance for each of the 10 API candidates recommended by *Gralan.*

Correct API labeled as “hit”, incorrect API labeled as “miss”

Each instance is represented using a set of **usage path features.**

2

Extract API Usage Path Features

- A sequence of APIs sequentially connected/listed API usage order with one entry and one exit API.
- Each usage path contains a candidate API (one of 10 candidate APIs recommended by *Gralan*) that appears either at the end or beginning of the pat
- Represent a data/control flow sequence of APIs
- Compared to context API usage graphs, it is bette captures the linguistic topic of the program expressing the intention of the developer withou including many irrelevant APIs

criminative Re-Ranking for API Recommendation (*RecRa*

1

Create Training Instances

For each API recommendation point in the training set, training instances are created.

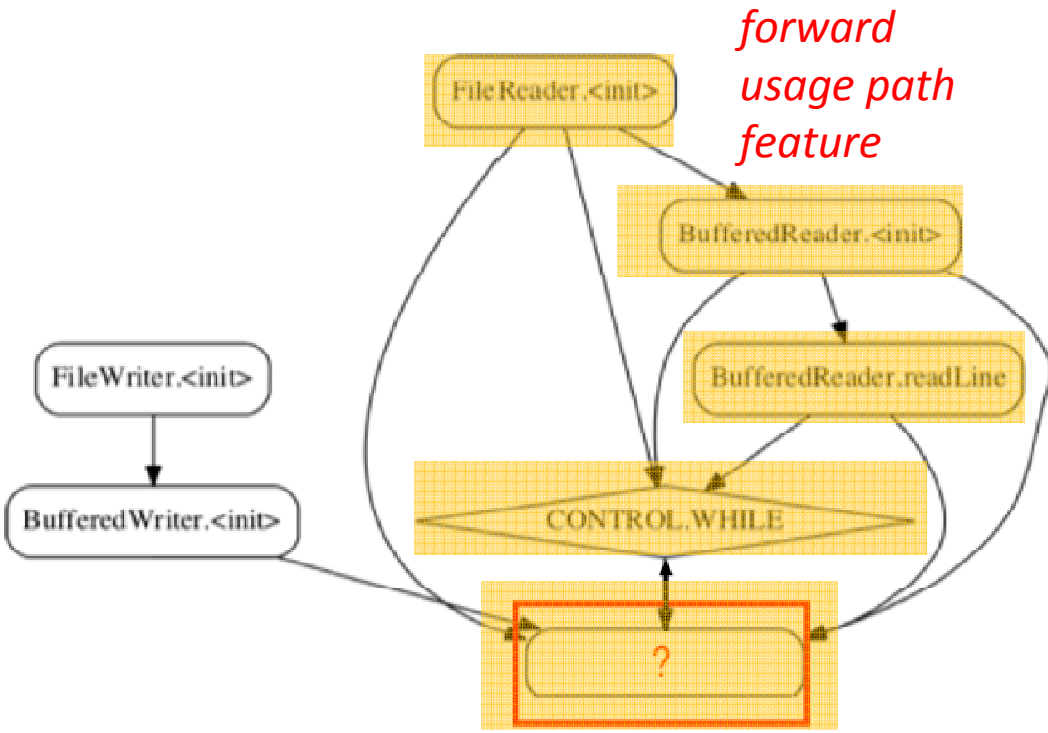
Create one training instance for each of the 10 API candidates recommended by *Gralan.*

Correct API labeled as “hit”, incorrect API labeled as “miss”

Each instance is represented using a set of **usage path features.**

2

Extract API Usage Path Features



[FileReader.<init> → BufferedReader.<init> → BufferedReader.readLine → CONTROL.WHILE → (candidate API)]

criminative Re-Ranking for API Recommendation (*RecRa*

1

Create Training Instances

For each API recommendation point in the training set, training instances are created.

Create one training instance for each of the 10 API candidates recommended by *Gralan.*

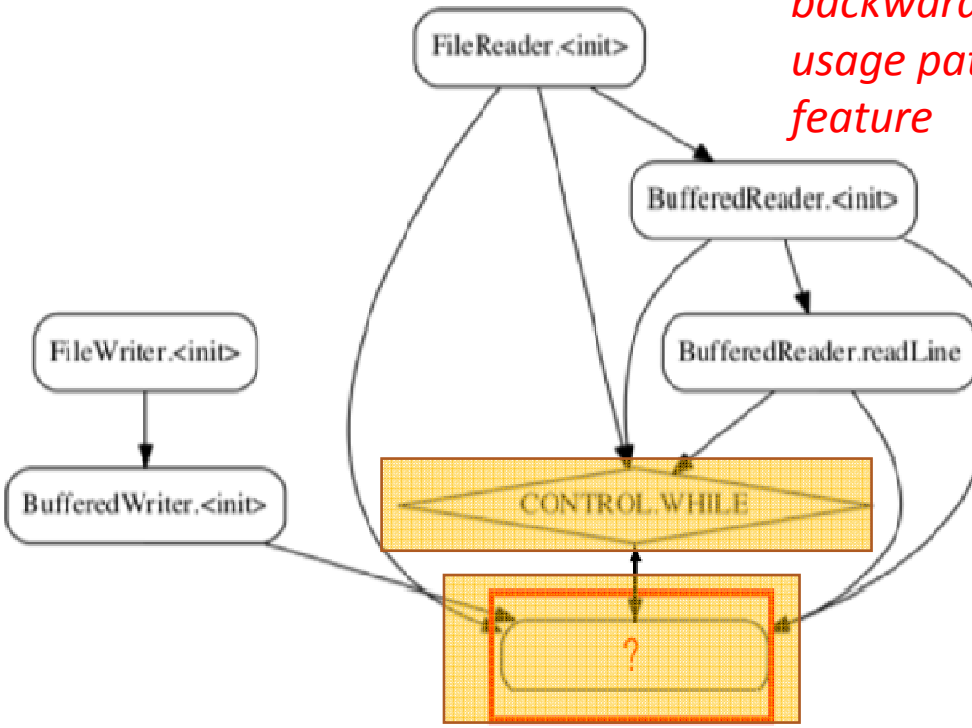
Correct API labeled as “hit”, incorrect API labeled as “miss”

Each instance is represented using a set of **usage path features.**

2

Extract API Usage Path Features

backward usage pat feature



[(candidate API) → CONTROL.WHILE]

criminative Re-Ranking for API Recommendation (*RecRa*

1

Create Training Instances

For each API recommendation point in the training set, training instances are created.

Create one training instance for each of the 10 API candidates recommended by *Gralan.*

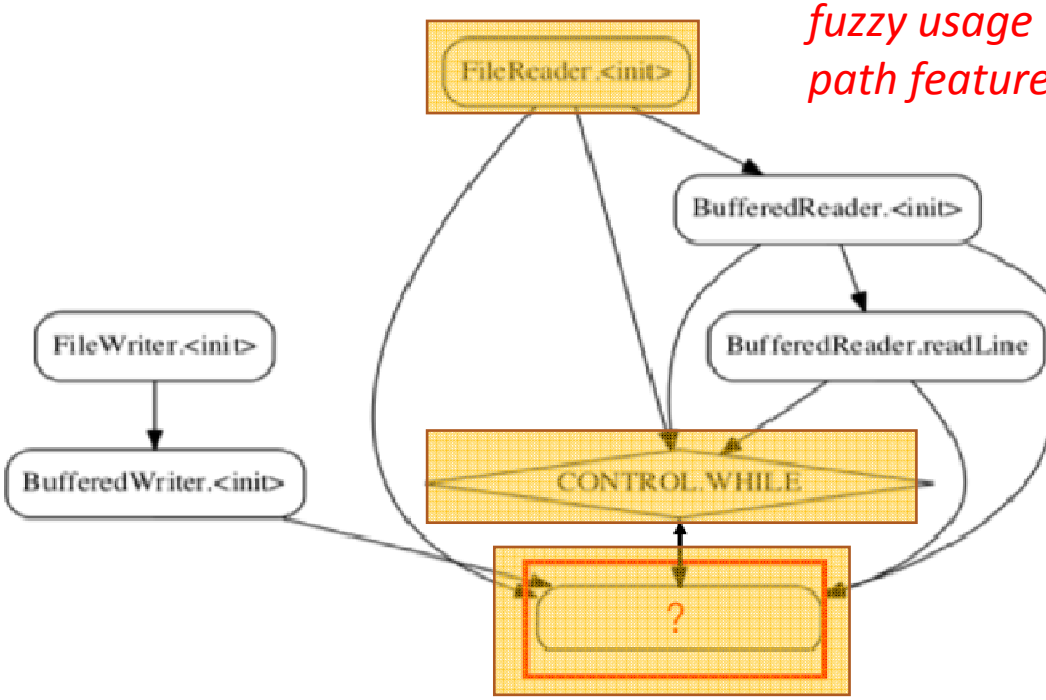
Correct API labeled as “hit”, incorrect API labeled as “miss”

Each instance is represented using a set of **usage path features.**

2

Extract API Usage Path Features

fuzzy usage path feature



[FileReader.<init> → *→ *→ CONTROL.WHILE → (candidate API)]

criminative Re-Ranking for API Recommendation (*NB*)

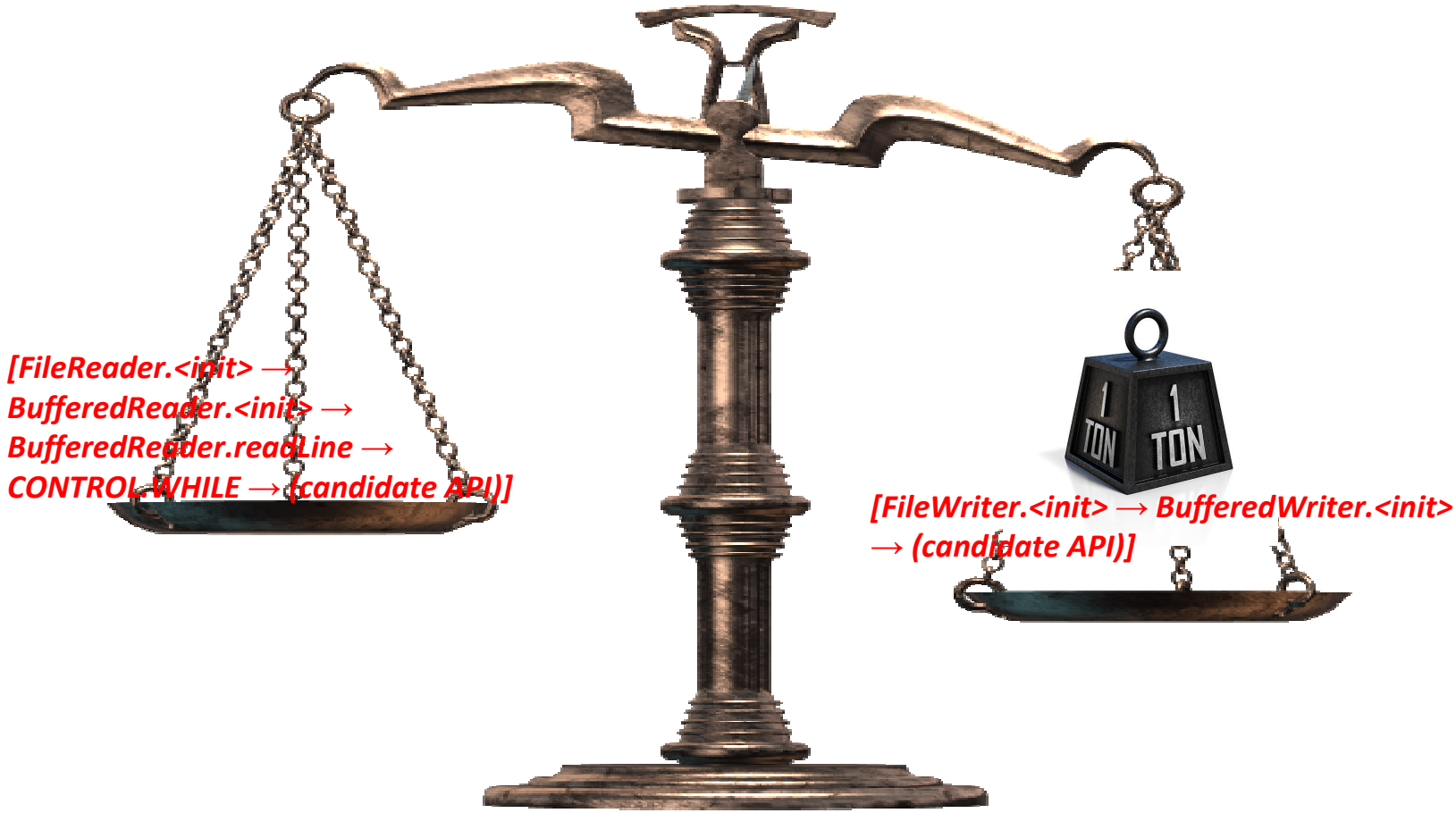
Generative Naïve Bayes Classifier

o investigate how generative model performs compared to discriminative model in API ecommendation.

enerative model assumes that the values of the usage path- based features are conditior ndependent of each other given the class.

andidate APIs are ranked using their associated probabilities, where higher probabilities orrespond to higher ranks.

scriminative Re-Ranking for API Recommendation



iscriminative Re-Ranking for API Recommendation (*SVC*

Discriminative SVC

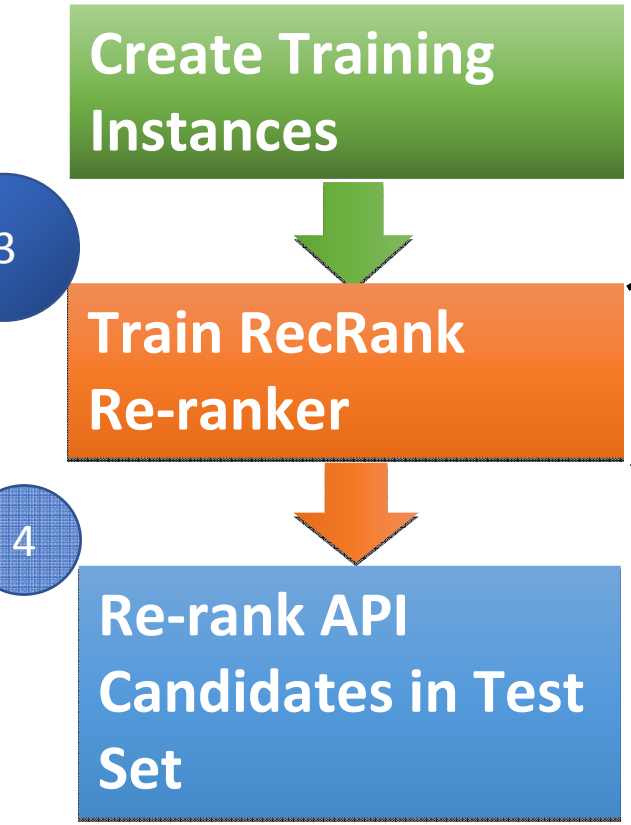
djust feature weights based on their relevance to the recommendation point.

- ✓ The more often the candidate API co-occurs with the rest of the path in the training set, the higher the feature value (more relevant features).

e-rank Top-10 candidate APIs based on their distances from the hyperplane

o investigate how discriminative classification model performs compared to scriminative ranking model in API recommendation

criminative Re-Ranking for API Recommendation (*RecRa*



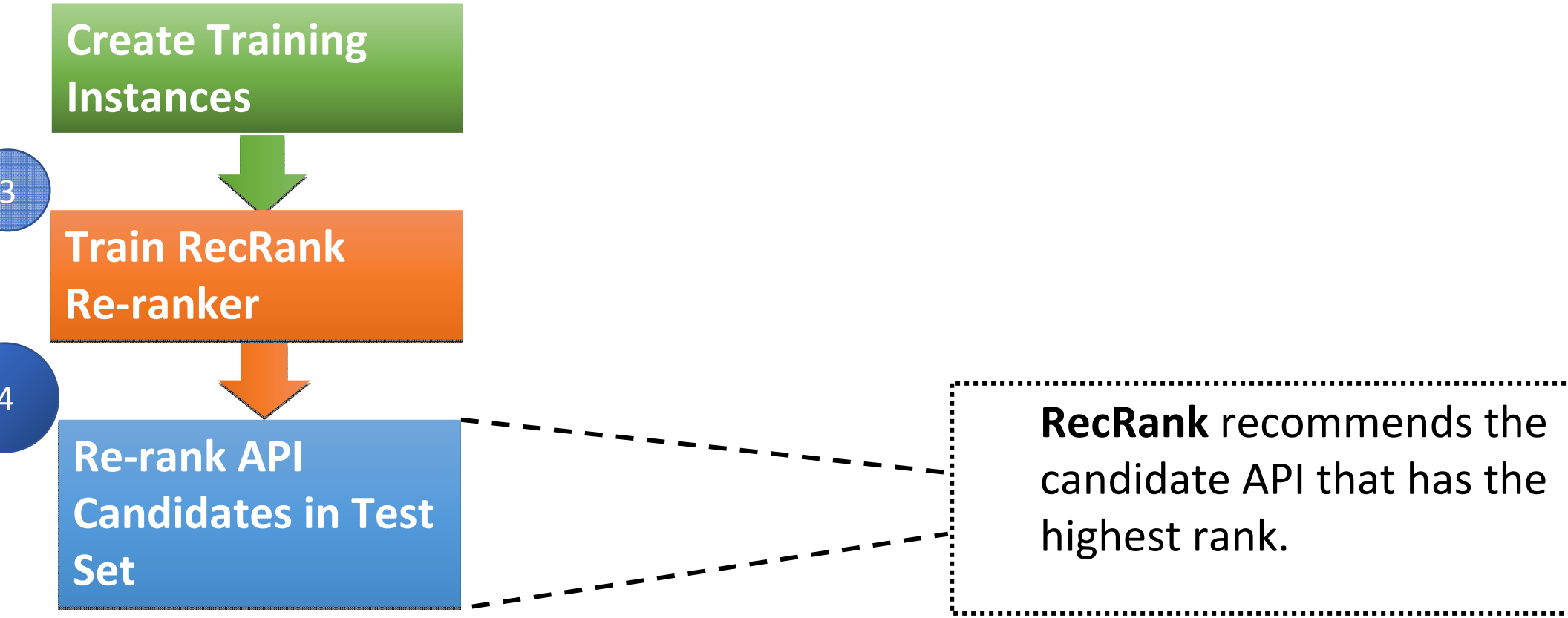
3. Discriminative RecRank

- **SVC** does not compare candidate APIs.

- **Train SVM Ranker:** Each ranking problem is composed of the 10 training instances corresponding to the top-10 candidate APIs for t recommendation point.

- **RecRank** learns a hyperplane (by adjusting the feature weights) to minimize the number of violations of pairwise ranking in the training set:
  - ✓ A violation occurs if a training instance labeled as "hit" is ranked below a training instance labeled as "miss" by the ranker.

scriminative Re-Ranking for API Recommendation (*RecR*

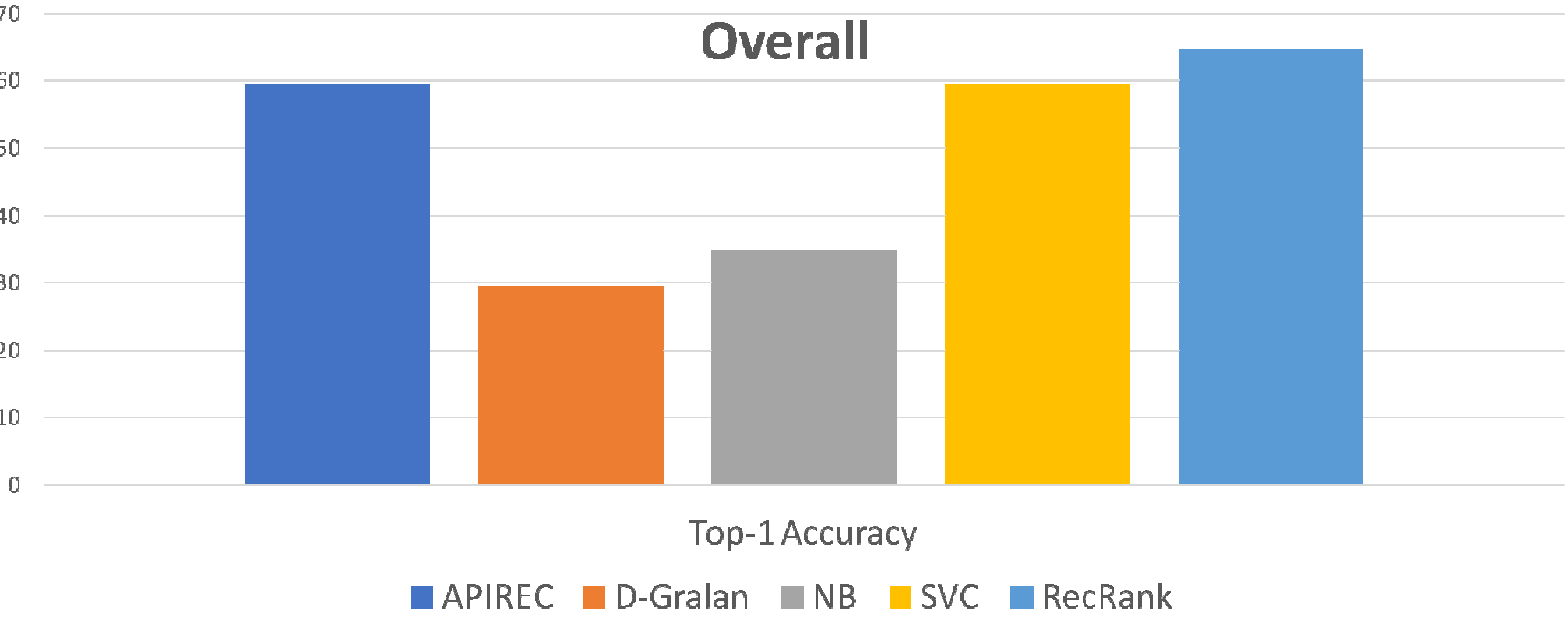


Empirical Evaluation

- **Datasets: Open Source Java Projects from GitHub**
  - ❖ 1385 Projects (Training)
  - ❖ 8 Projects (Evaluation)
- **Metrics:**
  - ❖ Top-1 Accuracy $= \frac{\#of\ correct\ API\ Recommendations}{Total\ \#\ of\ Recommendation\ Points}$
  - ❖ Mean Reciprocal Rank (MRR): Partial reward is inversely proportiona to the rank of the correct API
- **Two Baseline Systems**
  - ❖ *Gralan*: $d = 3$ for context diagram generation
  - ❖ *APIREC*

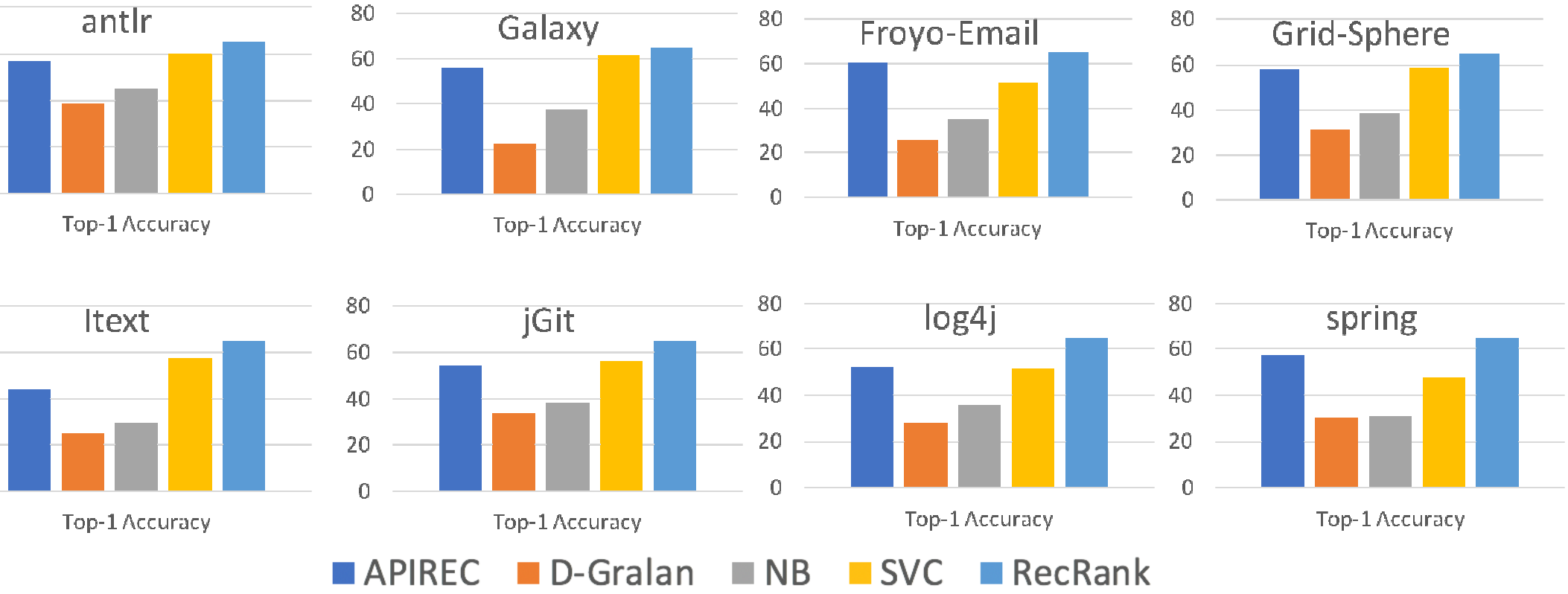
ow Well Does *RecRank* Perform?

1. *How accurate do RecRank, NB, and SVC recommend APIs in comparison to the o baselines?*



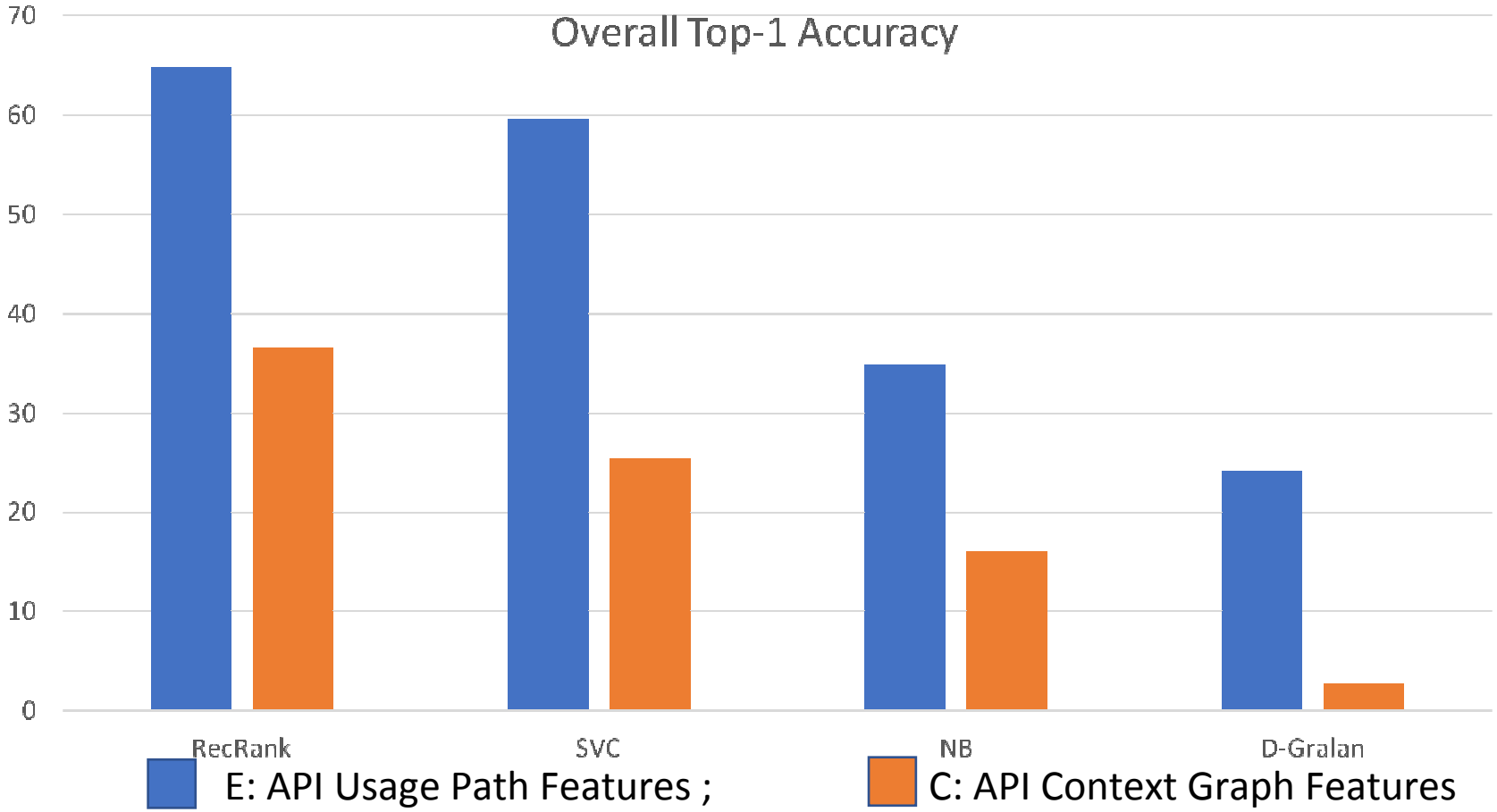
ow Well Does *RecRank* Perform?

1. *How accurate do RecRank, NB, and SVC recommend APIs in comparison to the o baselines?*



ow Well Does *RecRank* Perform?

***Q2**. How effective are usage path features for API recommendation ompared with context graphs?*



ow Well Does *RecRank* Perform?

3. *How effective are different classes of usage path features for API recommendat*

- **Performance drops highly significantly in three cases:**
  - ❖**when the length 2 forward features are removed**
  - ❖**when all forward features are removed**
  - ❖**when all length 2 features are removed**
- **This by no means implies that features of lengths 3 and 4 are not useful: these experiments only suggest that the feature group that is being removed is not useful in the presence of the remaining features**

Conclusions and Future Work

Compared with *Gralan*, discriminative re-ranking-based API recommendation system, *RecRank*, uses usage path-based features to significantly improved **top-1 accuracy** by 28.5%–50.0% and **MRR** by 0.32–0.49.

Compared with *APIREC*, **top-1 accuracy** is improved by as much as 23.7%.

Extend *RecRank* to a wider spectrum of API types and additional project domains.