Active Inductive Logic Programming for Code Search

Aishwarya Sivaraman, Tianyi Zhang, Guy Van den Broeck, Miryung Kim

University of California, Los Angeles

Tool and dataset: https://github.com/AishwaryaSivaraman/ALICE-ILP-for-Code-Search

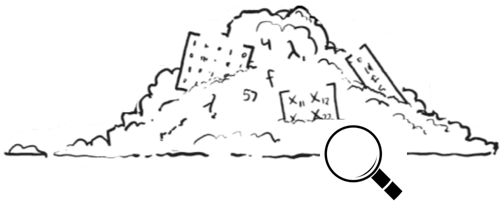
Developers Often Search For Similar Code

- Bug fix [Kim *et al.*, 2006]
- API-related refactoring [Dig and Johnson, 2006]
- Optimization [Ahmad and Cheung, 2018]

Existing Code Search

- Internet code search engines [Krugle, S6, CodeGenie]
  - Lacks expressiveness and query refinement is tedious
- Clone detection techniques [CCFinder, Deckard]
  - Threshold metric insufficient to capture the abstract search intent
- Interactive template based code search [Critics]
  - Interaction is tedious

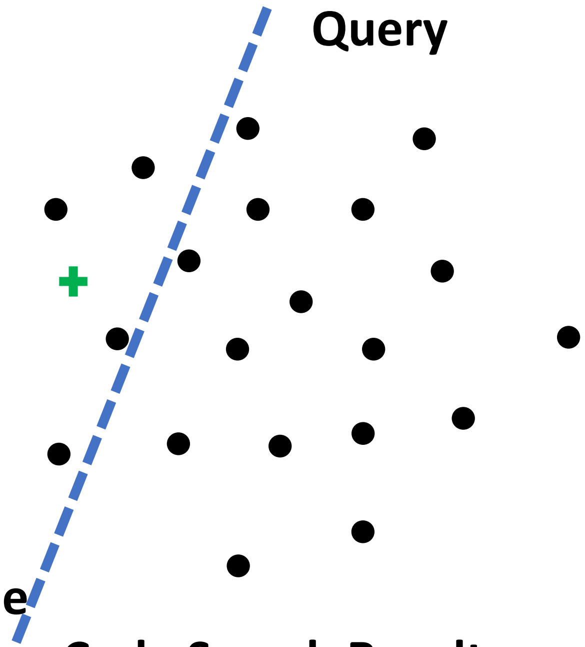
ALICE: Interactive Code Search via Active Inductive Logic Programming



Input: **One code example**

ALICE: **Generates a query (a search pattern)**

Output: **Set of method locations that match the query**

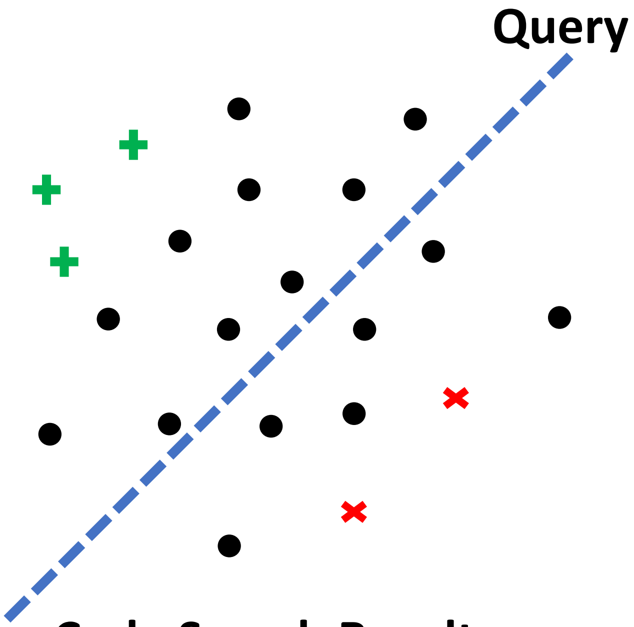


Query

Code Search Results (Iteration 0)

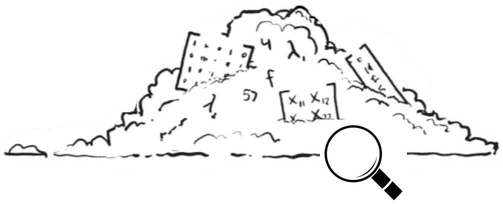
ALICE: Interactive Code Search via Active Inductive Logic Programming

Input: More labels



Code Search Results (Iteration 1)

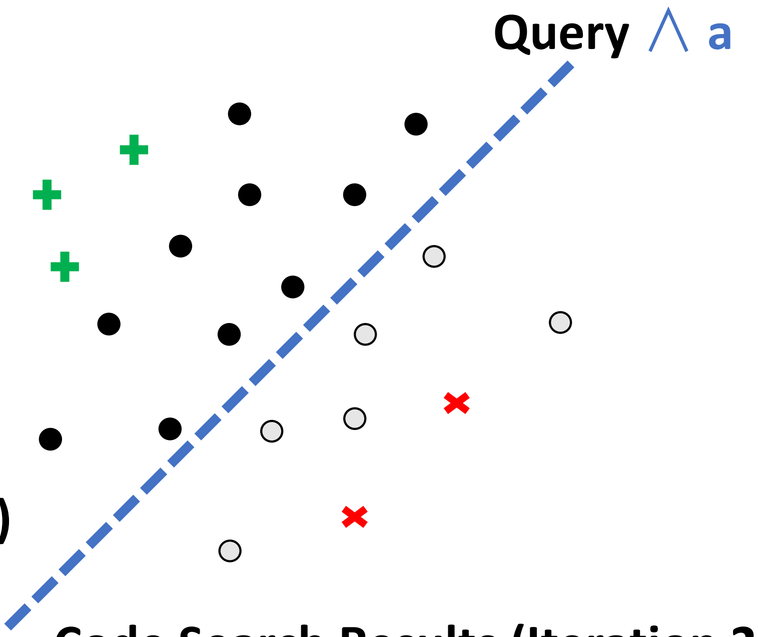
ALICE: Interactive Code Search via Active Inductive Logic Programming



Input: **More labels**

ALICE: **Refines the initial query (search pattern)**

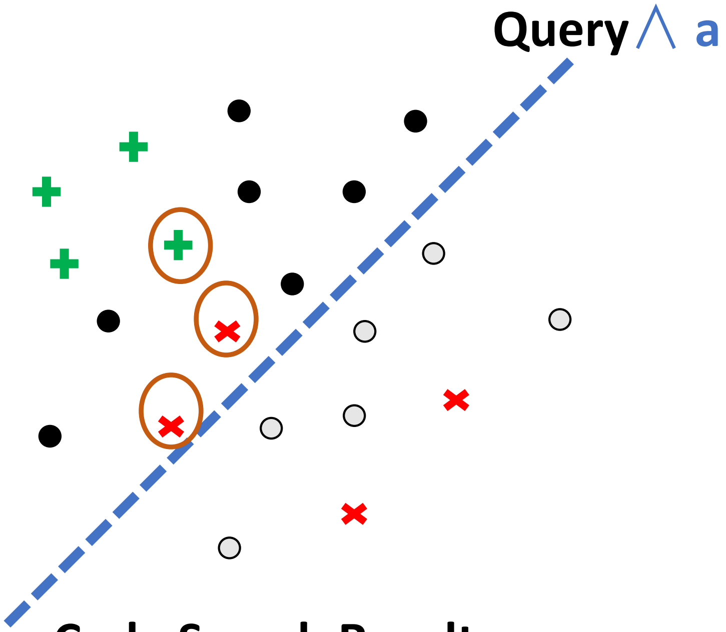
Output: **A smaller set of method locations that match the new query**



Code Search Results (Iteration 2)

ALICE: Interactive Code Search via Active Inductive Logic Programming

Input: More labels



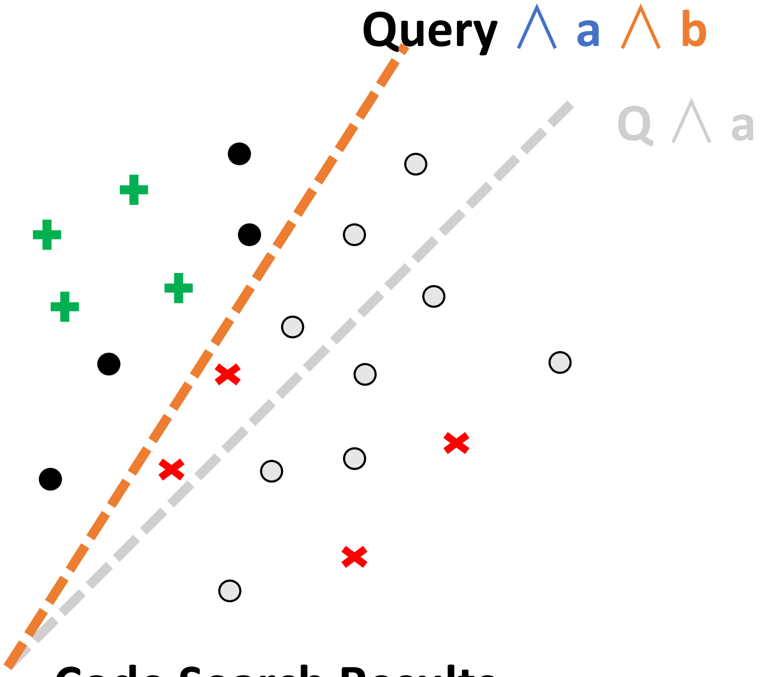
Code Search Results (Iteration 2)

ALICE: Interactive Code Search via Active Inductive Logic Programming

Input: More labels

ALICE: Keep refining the query

Output: A smaller set of method locations that match the new query



Code Search Results (Iteration 3)

Active Learning

- Obtaining labels is time consuming and expensive

Inductive Logic Programming

- Data as feature vectors cannot easily express the structure of code
- ILP: Positive examples + negative examples + background knowledge as rules

Represent Code as Logic Facts

Fact Predicate
if (ID, CONDITION)
loop (ID, CONDITION)
parent (ID, ID)
next (ID, ID)
methodCall (ID, NAME)
type (ID, NAME)
exception (ID, NAME)
methodDec (ID, NAME)

Represent Code as Logic Facts

Fact Predicate
if (ID, CONDITION)
loop (ID, CONDITION)
parent (ID, ID)
next (ID, ID)
methodCall (ID, NAME)
type (ID, NAME)
exception (ID, NAME)
methodDec (ID, NAME)

```
public void queryDB() {
   try {
        Connection con = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/db","root","root");
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("select * from emp");
        while (rs.next())  {
           System.out.println(rs.getInt(1));
        }
        con.close();
   } catch (SQLException e) {
       System.out.println(e);
   }
}
```

Extracted Logic Facts

methodDec (0, queryDB)

Represent Code as Logic Facts

Fact Predicate
if (ID, CONDITION)
loop (ID, CONDITION)
parent (ID, ID)
next (ID, ID)
methodCall (ID, NAME)
type (ID, NAME)
exception (ID, NAME)
methodDec (ID, NAME)

```
public void queryDB() {
   try {
        Connection con = DriverManager.getConnection(
               "jdbc:mysql://localhost:3306/db","root","root");
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("select * from emp");
        while (rs.next())  {
           System.out.println(rs.getInt(1));
        }
        con.close();
   } catch (SQLException e) {
       System.out.println(e);
   }
}
```

Extracted Logic Facts

methodDec (0, queryDB),
type (1, Connection),
parent (0, 1)

Represent Code as Logic Facts

Fact Predicate
if (ID, CONDITION)
loop (ID, CONDITION)
parent (ID, ID)
next (ID, ID)
methodCall (ID, NAME)
type (ID, NAME)
exception (ID, NAME)
methodDec (ID, NAME)

```
public void queryDB() {
   try {
        Connection con = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/db","root","root");
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("select * from emp");
        while (rs.next())  {
           System.out.println(rs.getInt(1));
        }
        con.close();
   } catch (SQLException e) {
       System.out.println(e);
   }
}
```

Extracted Logic Facts

methodDec (0, queryDB),
type (1, Connection),
parent (0, 1),
methodCall(2, getConnection),
parent (0, 2),
next (2, 1)

Represent Code as Logic Facts

Fact Predicate
if (ID, CONDITION)
loop (ID, CONDITION)
parent (ID, ID)
next (ID, ID)
methodCall (ID, NAME)
type (ID, NAME)
exception (ID, NAME)
methodDec (ID, NAME)

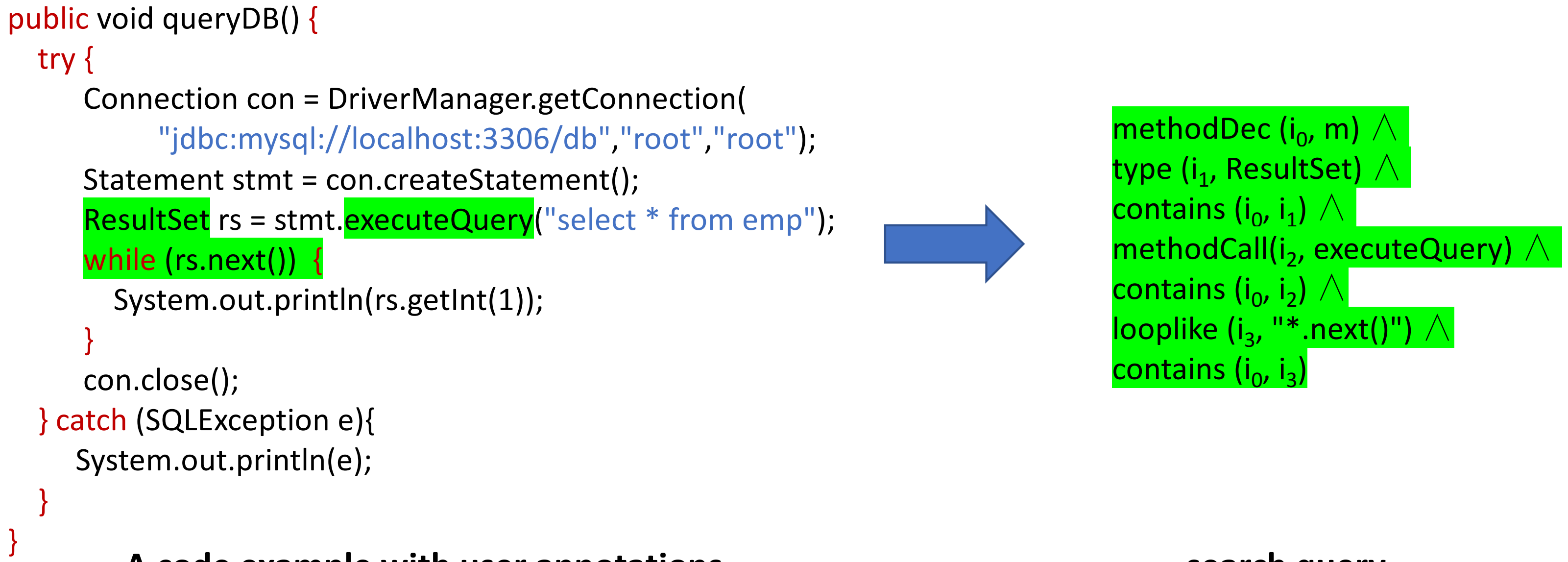
```
public void queryDB() {
   try {
        Connection con = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/db","root","root");
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("select * from emp");
        while (rs.next())  {
           System.out.println(rs.getInt(1));
        }
        con.close();
   } catch (SQLException e) {
       System.out.println(e);
   }
}
```

Extracted Logic Facts

```
methodDec (0, queryDB),
type (1, Connection),
parent (0, 1),
methodCall(2, getConnection),
parent (0, 2),
next (2, 1),
…
loop (7, "rs.next()"),
methodCall (8, getInt),
parent (7, 8),
…
exception (10, SQLException),
parent (0, 10),
…
```

Formulate a Search Query

- A user selects a code example and annotate important features.



```
public void queryDB() {
   try {
        Connection con = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/db","root","root");
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("select * from emp");
        while (rs.next())  {
           System.out.println(rs.getInt(1));
        }
        con.close();
   } catch (SQLException e){
       System.out.println(e);
   }
}
```

A code example with user annotations

methodDec (i_0, m) $\wedge$
type (i_1, ResultSet) $\wedge$
contains (i_0, i_1) $\wedge$
methodCall(i_2, executeQuery) $\wedge$
contains (i_0, i_2) $\wedge$
looplike (i_3, "*.next()") $\wedge$
contains (i_0, i_3)

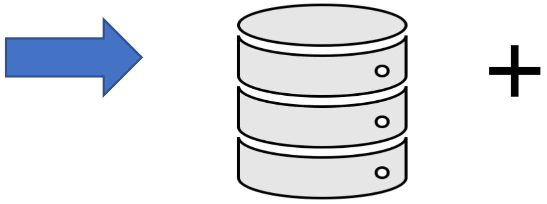
search query

Logic-based Code Search

Search Query

methodDec (i_0, m) $\wedge$
type (i_1, ResultSet) $\wedge$
contains (i_0, i_1) $\wedge$
methodCall(i_2, executeQuery) $\wedge$
contains (i_0, i_2) $\wedge$
looplike (i_3, "*.next()") $\wedge$
contains (i_0, i_3)

Fact Base



Rules
iflike (ID, regex) :- if (ID, cond), match (cond, regex)
looplike (ID, regex) :- loop (ID, cond), match (cond, regex)
contains (ID_1, ID_2) :- parent (ID_1, ID_2)
contains (ID_1, ID_3) :- parent (ID_1, ID_2), contains (ID_2, ID_3)
before(ID_1, ID_2) :- next(ID_2, ID_1)
before(ID_1, ID_3) : - next(ID_2, ID_1), before(ID_2, ID_3).

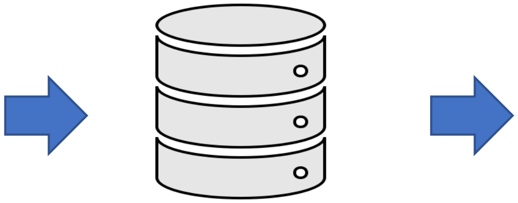
Costa et al., “The yap prolog system,” Theory and Practice of Logic Programming, 2012

Logic-based Code Search

Search Query

methodDec (i_0, m) $\wedge$
type (i_1, ResultSet) $\wedge$
contains (i_0, i_1) $\wedge$
methodCall(i_2, executeQuery) $\wedge$
contains (i_0, i_2) $\wedge$
looplike (i_3, "*.next()") $\wedge$
contains (i_0, i_3)

Fact Base



+

Fact Rules

Matched Code

```
public void getUserName(String id) {
   try {
         ResultSet set = db.executeQuery(
                  "select name from users where id=" + id);
         while (set.next()) { … }
   } catch (SQLException e) { …}
}

public void queryDatabase() {
   try {
         ResultSet result = s.executeQuery("select * from customers");
         while (result.next()) { … }
   } catch (SQLException e) { …}
}

public List get() {
    ResultSet set = stmt.executeQuery("select * from t");
    List l = new List();
    while (set.next()) { … }
    return l;
}
```

and 32 other matched locations

Partial Feedback

```
public void getUserName(String id) {
  try {
     ResultSet set = db.executeQuery(
           "select name from users where id=” + id);
     while (set.next()) { … }
  } catch (SQLException e) { …}
}
```

✓

```
public void queryDatabase() {
  try {
     ResultSet result = s.executeQuery("select * from customers”);
     while (result.next()) { … }
  } catch (SQLException e) { …}
}
```

✓

```
public List get() {
   ResultSet set = stmt.executeQuery("select * from t”);
   List l = new List();
   while (set.next()) { … }
   return l;
}
```

✗

Search Query

methodDec (i_0, m) $\wedge$
type (i_1, ResultSet) $\wedge$
contains (i_0, i_1) $\wedge$
methodCall(i_2, executeQuery) $\wedge$
contains (i_0, i_2) $\wedge$
looplike (i_3, "*.next()") $\wedge$
contains (i_0, i_3)

Query Refinement via Active Learning

```
public void getUserName(String id) {
   try {
        ResultSet set = db.executeQuery(
                  "select name from users where id=” + id);
        while (set.next()) { … }
   } catch (SQLException e) { …}
}
```

✓

```
public void queryDatabase() {
   try {
        ResultSet result = s.executeQuery("select * from customers”);
        while (result.next()) { … }
   } catch (SQLException e) { …}
}
```

✓

```
public List get() {
    ResultSet set = stmt.executeQuery("select * from t”);
    List l = new List();
    while (set.next()) { … }
    return l;
}
```

✗

Refined Query

methodDec (i_0, m) $\wedge$
type (i_1, ResultSet) $\wedge$
contains (i_0, i_1) $\wedge$
methodCall(i_2, executeQuery) $\wedge$
contains (i_0, i_2) $\wedge$
looplike (i_3, "*.next()") $\wedge$
contains (i_0, i_3)

Query Refinement Optimization

$$\texttt{Specialize}(h_{i-1}, P, N) = \underset{h_i}{\text{argmax}} \sum_{p \in P} [p \models h_i]$$

$$\text{such that } h_i \models h_{i-1} \text{ and } \forall n \in N, n \not\models h_i$$

Query Refinement via Active Learning

```
public void getUserName(String id) {
   try {
        ResultSet set = db.executeQuery(
                  "select name from users where id=” + id);
        while (set.next()) { … }
   } catch (SQLException e) { …}
}
```

✓

```
public void queryDatabase() {
   try {
        ResultSet result = s.executeQuery("select * from customers”);
        while (result.next()) { … }
   } catch (SQLException e) { …}
}
```

✓

```
public List get() {
    ResultSet set = stmt.executeQuery("select * from t”);
    List l = new List();
    while (set.next()) { … }
    return l;
}
```

✗

Refined Query

methodDec (i_0, m) $\wedge$
type (i_1, ResultSet) $\wedge$
contains (i_0, i_1) $\wedge$
methodCall(i_2, executeQuery) $\wedge$
contains (i_0, i_2) $\wedge$
looplike (i_3, "*.next()") $\wedge$
contains (i_0, i_3) $\wedge$
exception (i_4, SQLException),
contains (i_0, i_4)

Query Refinement Optimization

$$\texttt{Specialize}(h_{i-1}, P, N) = \operatorname*{argmax}_{h_i} \sum_{p \in P} [p \models h_i]$$

such that $h_i \models h_{i-1}$ and $\forall n \in N, n \not\models h_i$

How To Pick a Discriminatory Atom?

A code example with user annotations

```
public void queryDB() {
  try {
     Connection con = DriverManager.getConnection(
          "jdbc:mysql://localhost:3306/db","root","root");
     Statement stmt = con.createStatement();
     ResultSet rs = stmt.executeQuery("select * from emp");
     while (rs.next())  {
       System.out.println(rs.getInt(1));
     }
     con.close();
  } catch (SQLException e){
     System.out.println(e);
  }
}
```

🟩 User annotations

🟨 Potential Candidate Features

Inductive Bias

1. ***Feature Vector* considers source code has a flat structure**
2. *Nested Structure* prioritizes code elements with containment relationship
3. *Sequential Code Order* prioritizes code elements with sequential ordering

A code example with user annotations

```
public void queryDB() {
  try {
      Connection con = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/db","root","root");
      Statement stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery("select * from emp");
      while (rs.next())  {
         System.out.println(rs.getInt(1));
      }
      con.close();
  } catch (SQLException e){
      System.out.println(e);
  }
}
```

Inductive Bias

1. *Feature Vector* considers source code has a flat structure
2. ***Nested Structure* prioritizes code elements with containment relationship**
3. *Sequential Code Order* prioritizes code elements with sequential ordering

A code example with user annotations

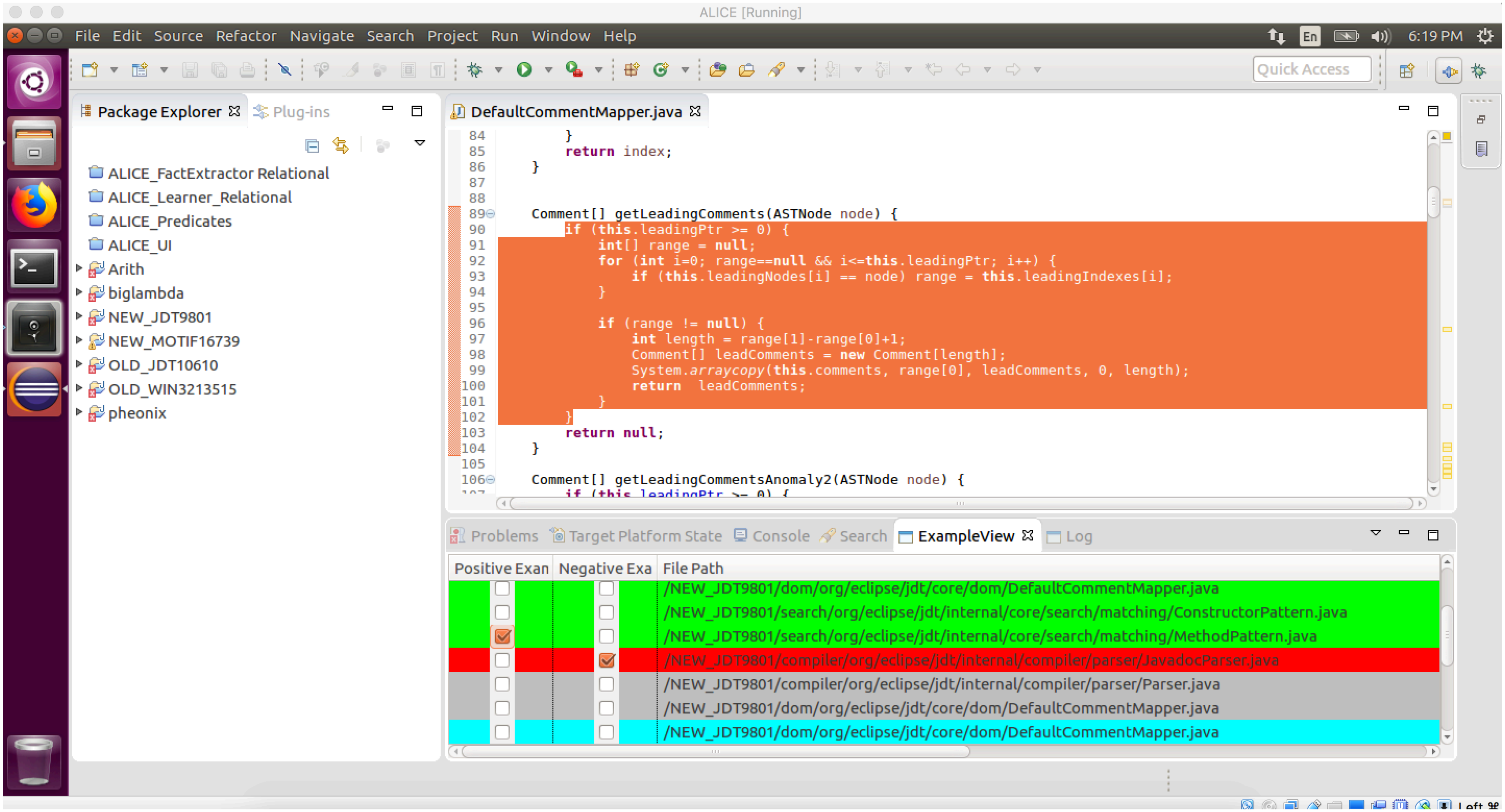
```
public void queryDB() {
  try {
      Connection con = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/db","root","root");
      Statement stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery("select * from emp");
      while (rs.next())  {
        System.out.println(rs.getInt(1));
      }
      con.close();
  } catch (SQLException e){
      System.out.println(e);
  }
}
```

Inductive Bias

1. *Feature Vector* considers source code has a flat structure
2. *Nested Structure* prioritizes code elements with containment relationship
3. ***Sequential Code Order* prioritizes code elements with sequential ordering**

A code example with user annotations

```
public void queryDB() {
   try {
        Connection con = DriverManager.getConnection(
              "jdbc:mysql://localhost:3306/db","root","root");
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("select * from emp");
        while (rs.next())  {
           System.out.println(rs.getInt(1));
        }
        con.close();
   } catch (SQLException e){
       System.out.println(e);
   }
}
```



```
        }
        return index;
    }


    Comment[] getLeadingComments(ASTNode node) {
        if (this.leadingPtr >= 0) {
            int[] range = null;
            for (int i=0; range==null && i<=this.leadingPtr; i++) {
                if (this.leadingNodes[i] == node) range = this.leadingIndexes[i];
            }

            if (range != null) {
                int length = range[1]-range[0]+1;
                Comment[] leadComments = new Comment[length];
                System.arraycopy(this.comments, range[0], leadComments, 0, length);
                return  leadComments;
            }
        }
        return null;
    }

    Comment[] getLeadingCommentsAnomaly2(ASTNode node) {
        if (this.leadingPtr >= 0) {
```

Positive Exan	Negative Exa	File Path
□	□	/NEW_JDT9801/dom/org/eclipse/jdt/core/dom/DefaultCommentMapper.java
□	□	/NEW_JDT9801/search/org/eclipse/jdt/internal/core/search/matching/ConstructorPattern.java
☑	□	/NEW_JDT9801/search/org/eclipse/jdt/internal/core/search/matching/MethodPattern.java
□	☑	/NEW_JDT9801/compiler/org/eclipse/jdt/internal/compiler/parser/JavadocParser.java
□	□	/NEW_JDT9801/compiler/org/eclipse/jdt/internal/compiler/parser/Parser.java
□	□	/NEW_JDT9801/dom/org/eclipse/jdt/core/dom/DefaultCommentMapper.java
□	□	/NEW_JDT9801/dom/org/eclipse/jdt/core/dom/DefaultCommentMapper.java

Evaluation

- Simulation Experiments
- A Comparison with Critics
- A Case Study with Real Users

Evaluation

Simulation Experiments

A Comparison with Critics

A Case Study with Real Users

Experiment Benchmarks

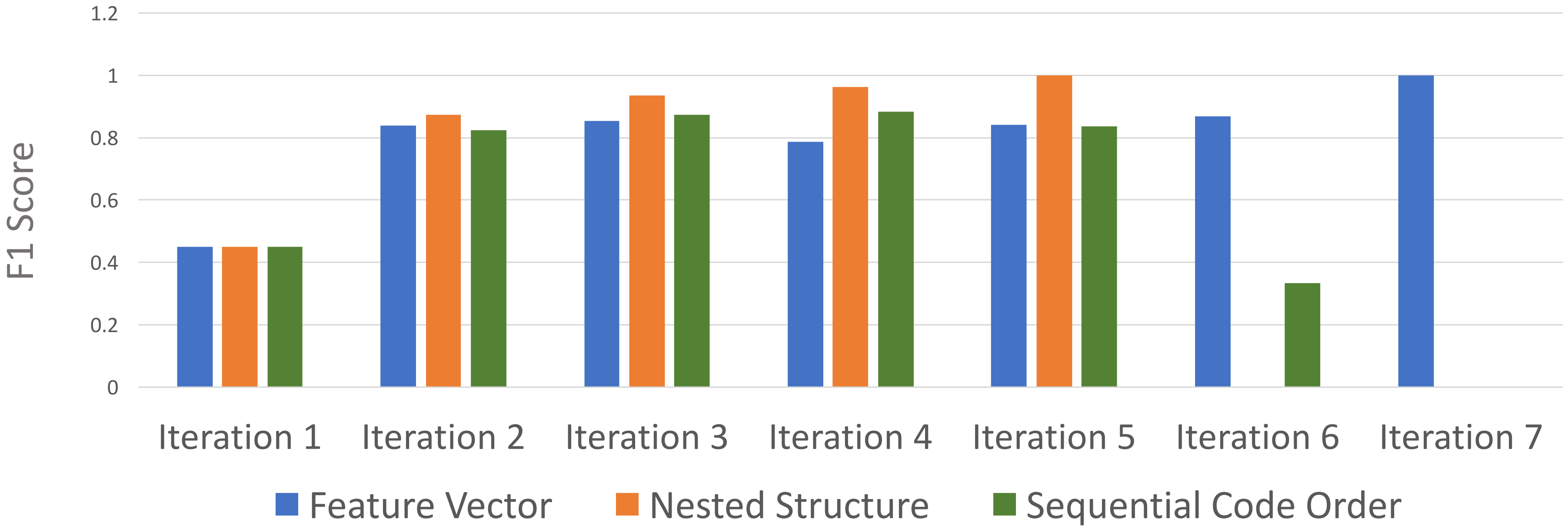
Simulation

- Similar locations to update [Meng *et al.*, 2013]
  - 14 groups of syntactically similar code fragments from Eclipse JDT and SWT
- Code optimization [Ahmad *et al.*, 2018]
  - 6 groups of similar programs that follow the same code pattern

(RQ1) Which inductive bias is effective?

Simulation

- Nested structure bias is the most effective.



* Averaged over 10 runs.

(RQ2) How much does a user should annotate?

Simulation

- **Method:** Randomly annotate important code elements in an example
- **Result:** Annotating more features increases precision but not recall.

	1 Feature	2 Features	3 Features	4 Features
Precision	0.16	0.47	0.68	0.80
Recall	0.91	0.86	0.80	0.78

* Averaged over 10 runs.

(RQ3) How many labels should a user provide?

Simulation

- **Method:** Label randomly selected search results w.r.t. the ground truth.
- **Results:** Labeling three examples is optimal.

	2 Labels	3 Labels	4 Labels	5 Labels
Precision	1.0	1.0	1.0	1.0
Recall	1.0	0.88	0.81	0.75
# Iterations	7	6	5	5
# Total Labels	14	18	20	25

* Averaged over 10 runs.

(RQ4) What if a user makes mistakes?

Simulation

- **Method:** Flip a label (e.g., positive -> negative) with a probability.
- **Result:** Report contradictory labels immediately and behave robustly when no inconsistencies are found.

	Error Rate		
	10%	20%	40%
Precision	1.0	1.0	1.0
Recall	0.95	0.90	0.93
% of Inconsistency feedback	33%	60%	54%

* Averaged over 10 runs.

Overall Performance

Simulation

- Simulate user behavior
  - Randomly select a code fragment in each group as a seed example
  - Randomly tag two important features
  - Randomly label three examples w.r.t. the ground truth
- 93% precision and 96% recall in 3 search iterations

* Averaged over 10 runs.

Evaluation

Simulation Experiment

Comparison with Critics

Case Study with Real Users

Comparison with Critics [Zhang et al., ICSE 2015]

Comparison

- Critics supports interactive code search via template refinement.

```
try {
   Connection con = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/db","root","root");
   Statement stmt = con.createStatement();
   ResultSet rs = stmt.executeQuery("select * from emp");
   while (rs.next())  {
       System.out.println(rs.getInt(1));
   }
   con.close();
} catch (SQLException e) {
   System.out.println(e);
}
```

A concrete code example

```
try {
   $EXCLUDE
   Statement stmt = con.createStatement();
   ResultSet rs = stmt.executeQuery("select * from emp");
   while (rs.next())  {
       System.out.println(rs.getInt(1));
   }
   $v0.close();
} catch (SQLException e) {
   System.out.println(e);
}
```

A search template

Comparison with Critics [Zhang et al., ICSE 2015]

Comparison

- Critics supports interactive code search via template refinement.

```
try {
   Connection con = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/db","root","root");
   Statement stmt = con.createStatement();
   ResultSet rs = stmt.executeQuery("select * from emp");
   while (rs.next())  {
      System.out.println(rs.getInt(1));
   }
   con.close();
} catch (SQLException e) {
   System.out.println(e);
}
```

A concrete code example

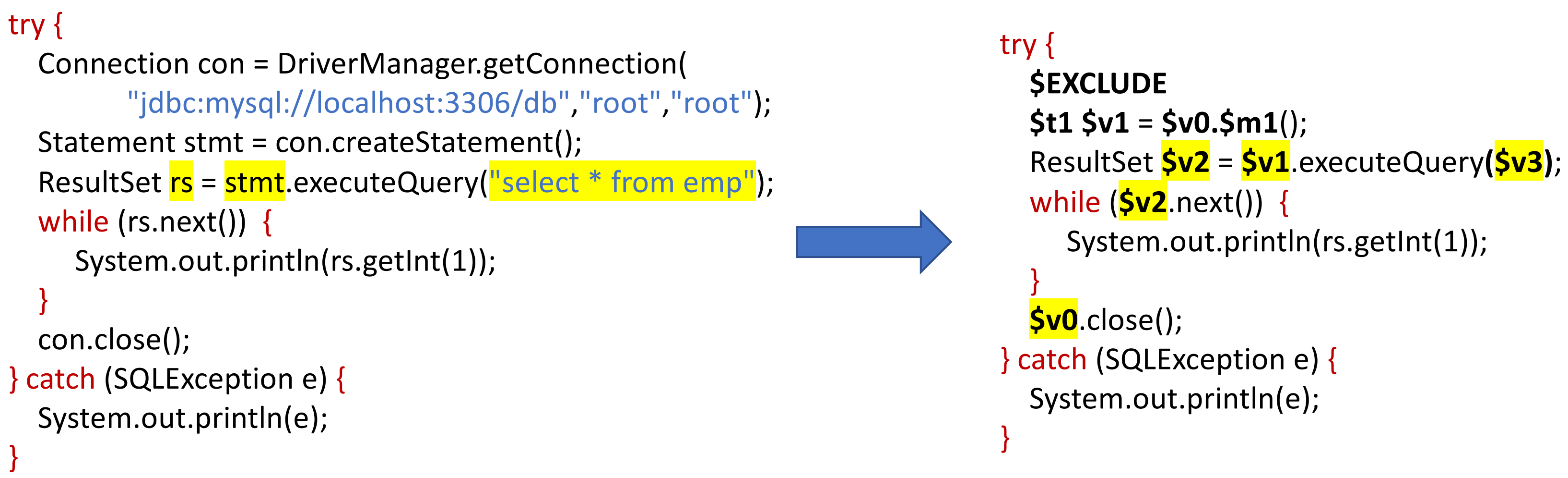
```
try {
   $EXCLUDE
   $t1 $v1 = $v0.$m1();
   ResultSet rs = $v1.executeQuery("select * from emp");
   while (rs.next())  {
      System.out.println(rs.getInt(1));
   }
   $v0.close();
} catch (SQLException e) {
   System.out.println(e);
}
```

A search template

Comparison with Critics [Zhang et al., ICSE 2015]

Comparison

- Critics supports interactive code search via template refinement.



```
try {
   Connection con = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/db","root","root");
   Statement stmt = con.createStatement();
   ResultSet rs = stmt.executeQuery("select * from emp");
   while (rs.next())  {
       System.out.println(rs.getInt(1));
   }
   con.close();
} catch (SQLException e) {
   System.out.println(e);
}
```

```
try {
   $EXCLUDE
   $t1 $v1 = $v0.$m1();
   ResultSet $v2 = $v1.executeQuery($v3);
   while ($v2.next())  {
       System.out.println(rs.getInt(1));
   }
   $v0.close();
} catch (SQLException e) {
   System.out.println(e);
}
```

A concrete code example

A search template

Comparison with Critics [Zhang et al., ICSE 2015]

Comparison

- Critics supports interactive code search via template refinement.

```
try {
   Connection con = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/db","root","root");
   Statement stmt = con.createStatement();
   ResultSet rs = stmt.executeQuery("select * from emp");
   while (rs.next())  {
       System.out.println(rs.getInt(1));
   }
   con.close();
} catch (SQLException e) {
   System.out.println(e);
}
```

A concrete code example

→

```
try {
   $EXCLUDE
   $t1 $v1 = $v0.$m1();
   ResultSet $v2 = $v1.executeQuery($v3);
   while ($v2.next())  {
      $EXCLUDE
   }
   $v0.close();
} catch (SQLException $v4) {
   $EXCLUDE
}
```

A search template

Comparison with Critics

Comparison

- ALICE achieves comparable or better accuracy with fewer iterations.

Group ID	ALICE			Critics		
	Precision	Recall	Iteration	Precision	Recall	Iteration
1	1.0	1.0	2	1.0	1.0	4
2	1.0	1.0	2	1.0	0.9	6
3	1.0	1.0	1	1.0	0.88	6
4	0.0	1.0	1	1.0	1.0	0
5	1.0	1.0	3	1.0	1.0	7
6	1.0	1.0	3	1.0	1.0	4
7	1.0	1.0	3	1.0	0.33	3
Average	0.86	1.0	2.1	1.0	0.87	4.3

Evaluation

Simulation Experiment

Comparison with Critics

Case Study with Real Users

Case Study: Eclipse SWT Revision 16379

User Study

- Recruit three graduate students to perform a code search task
- Participants can
  - easily recognize important features to annotate
  - distinguish positive and negative examples without much effort

Participant	#Examples	#Positives	#Negatives	Time Taken(s)
P1	8	1	1	20
P2	437	0	2	55
P3	8	1	0	25

Summary

- A novel learning based paradigm that lets ***users to express search intent*** via annotation and labelling.
- Our inductive bias ***eliminates tedious labelling effort*** by requiring a user to label a partial dataset.
- Our ***active learning*** engine enables an ***easy query refinement*** by leverage both positive and negative examples.
- A comprehensive simulation and a case study with real users indicate that ***interactivity pays off***.

Tool and dataset: https://github.com/AishwaryaSivaraman/ALICE-ILP-for-Code-Search

Q & A