Using Knowledge Units of Programming Languages to Recommend Reviewers for Pull Requests: An Empirical Study

Md Ahasanuzzaman
Gustavo A. Oliva
Ahmed E. Hassan
Finding the right reviewer for reviewing a piece of code is a difficult task
Finding the right reviewer for reviewing a piece of code is a difficult task.

The quality of a code review inherently depends on the selection of the reviewer.
Finding the right reviewer for reviewing a piece of code is a difficult task.

The quality of a code review inherently depends on the selection of the reviewer.

Finding the right reviewer for a set of code changes is always a nontrivial task, especially for a large-scale, distributed software development.
Mapping different expertise to individual developers is a key requirement for effective code review.
Mapping different expertise to individual developers is a key requirement for effective code review

Recent research studies have attempted to develop approaches to detect experts in specific topics [1]

[1] Identifying experts in software libraries and frameworks among GitHub users, MSR, 2016
We focus on a key facet of expertise, which is programming language (PL) expertise.
We focus on a key facet of expertise, which is programming language (PL) expertise.

Our rationale is that a piece of code involving concurrency is suitable to be reviewed by someone who has demonstrated experience in dealing with such type of code.
To capture the PL expertise of developers, we introduce the notion of Knowledge Units (KUs) of PL
To capture the PL expertise of developers, we introduce the notion of Knowledge Units (KUs) of PL.

Knowledge Unit (KU):

A Knowledge Unit (KU) is a cohesive set of key capabilities that are offered by one or more building blocks of a given programming language.
Knowledge Unit (KU):

A Knowledge Unit (KU) is a cohesive set of **key capabilities** that are offered by one or more building blocks of a given
Knowledge Unit (KU):

A Knowledge Unit (KU) is a cohesive set of **key capabilities** that are offered by one or more building blocks of a given
Knowledge Unit (KU):

A Knowledge Unit (KU) is a cohesive set of **key capabilities** that are offered by one or more building blocks of a given
Knowledge Unit (KU): A Knowledge Unit (KU) is a cohesive set of key capabilities that are offered by one or more building blocks of a given...
Knowledge Unit (KU):

A Knowledge Unit (KU) is a cohesive set of **key capabilities** that are offered by one or more building blocks of a given
Knowledge Unit (KU):

A Knowledge Unit (KU) is a cohesive set of **key capabilities** that are offered by one or more building blocks of a given
Knowledge Unit (KU):

A Knowledge Unit (KU) is a cohesive set of **key capabilities** that are offered by one or more building blocks of a given.
Knowledge Unit (KU):

A Knowledge Unit (KU) is a cohesive set of **key capabilities** that are offered by one or more building blocks of a given
We operationalize our KUs via certification exams for the Java programming language
We operationalize our KUs via certification exams for the Java programming language.

Oracle Java SE and Java EE certification exams for the Java
We operationalize our KUs via certification exams for the Java programming language.

Certification exams of a programming language aim to determine the skills and knowledge of a developer in using the key capabilities offered by the building blocks of that language.

Oracle Java SE and Java EE certification exams for the Java
We operationalize our KUs via certification exams for the Java programming language. Certification exams of a programming language aim to determine the skills and knowledge of a developer in using the key capabilities offered by the building blocks of that language. Thus, certification exams capture the KUs of a programming language.
We map the topics and subtopics of the Java Certification Exam into KUs

Java Concurrency

- Create worker threads using Runnable, Callable and use an ExecutorService to concurrently execute tasks
- Identify potential threading problems among deadlock, starvation, livelock, and race conditions
- Use synchronized keyword and java.util.concurrent.atomic package to control the order of thread execution

Building Database Applications with JDBC

- Describe the interfaces that make up the core of the JDBC API including the Driver, Connection, Statement, and ResultSet interfaces and their relationship to provider implementations
- Identify the components required to connect to a database using the DriverManager class including the JDBC URL
We map the topics and subtopics of the Java Certification Exam into KUs

- **Java Concurrency**
  - Topic
    - Create worker threads using Runnable, Callable and use an ExecutorService to concurrently execute tasks
    - Identify potential threading problems among deadlock, starvation, livelock, and race conditions
    - Use synchronized keyword and java.util.concurrent.atomic package to control the order of thread execution

**Building Database Applications with JDBC**

- Describe the interfaces that make up the core of the JDBC API including the Driver, Connection, Statement, and ResultSet interfaces and their relationship to provider implementations
- Identify the components required to connect to a database using the DriverManager class including the JDBC URL
We map the topics and subtopics of the Java Certification Exam into KUs.
We map the topics and subtopics of the Java Certification Exam into KUs.
We identify 28 KUs of the Java programming language
We identify 28 KUs of the Java programming language

(1) Data Type KU  (10) Stream API KU
(2) Operator and Decision KU  (11) Exception KU
(3) Array KU  (12) Date time API KU
(4) Loop KU  (13) IO KU
(5) Method and Encapsulation KU  (14) NIO KU
(6) Inheritance KU  (15) Concurrency KU
(7) Advanced Class Design KU  (16) Database KU
(8) Generics and Collection KU  (17) String Processing KU
(9) Functional Interface KU  (18) Localization KU

JAVA SE KUs
We identify 28 KUs of the Java programming language

1. Data Type KU
2. Operator and Decision KU
3. Array KU
4. Loop KU
5. Method and Encapsulation KU
6. Inheritance KU
7. Advanced Class Design KU
8. Generics and Collection KU
9. Functional Interface KU
10. Stream API KU
11. Exception KU
12. Date time API KU
13. IO KU
14. NIO KU
15. Concurrency KU
16. Database KU
17. String Processing KU
18. Localization KU
19. Java Persistence KU
20. Enterprise Java Bean KU
21. Java Message Service API KU
22. SOAP Web Service KU
23. Servlet KU
24. Java REST API KU
25. Websocket KU
26. Java Server Faces KU
27. Contexts and Dependency injection (CDI) KU
28. Batch Processing KU

JAVA SE KUs

JAVA EE KUs
Our objective

How we can leverage KUs to build expertise-profile for developers and construct a **recommender system (KUREC)** for GitHub pull requests (PRs)
We represent developers’ expertise with KUs that are associated with changed files in commits
We represent developers’ expertise with KUs that are associated with changed files in commits

(a) Developers' commit activity and the changed files
We represent developers’ expertise with KUs that are associated with changed files in commits

(a) Developers' commit activity and the changed files

(b) Representation of developer’s expertise with KUs
We represent developers’ expertise with KUs that are associated with changed files in commits.

(a) Developers' commit activity and the changed files

(b) Representation of developer’s expertise with KUs
We represent developers' expertise with KUs that are associated with changed files in commits.

(a) Developers' commit activity and the changed files

(b) Representation of developer’s expertise with KUs
We collected **290k** commit data and **65k** pull request data from 8 active Java projects in GitHub.
Preliminary Study: Do KUs provide a new lens to study developers’ expertise?
KUs offer a fine-grained lens to study developers’ expertise.
KUs offer a fine-grained lens to study developers’ expertise
KUs offer a fine-grained lens to study developers’ expertise.

KUs identify 71 different clusters of developers.
KUs offer a fine-grained lens to study developers’ expertise

KUs identify 71 different clusters of developers

The Gini index for clusters’ size is 0.96, indicating that there exists a very strong inequality.
KUs offer a fine-grained lens to study developers’ expertise

KUs identify 71 different clusters of developers

The Gini index for clusters’ size is 0.96, indicating that there exists a very strong inequality

A few clusters with a large number of developers

Developer clusters with KU-based expertise profile
KUs offer a fine-grained lens to study developers’ expertise.

KUs identify 71 different clusters of developers.

The Gini index for clusters’ size is 0.96, indicating that there exists a very strong inequality.

A few clusters with a large number of developers.

Many clusters with a few developers.

Developer clusters with KU-based expertise profile.
KUs offer a fine-grained lens to study developers’ expertise

KUs identify 71 different clusters of developers

The Gini index for clusters’ size is 0.96, indicating that there exists a very strong inequality

57% of the generated clusters are singleton (i.e., the size of these clusters is one)
KUs offer a fine-grained lens to study developers’ expertise

KUs identify 71 different clusters of developers

The Gini index for clusters’ size is 0.96, indicating that there exists a very strong inequality

57% of the generated clusters are singleton (i.e., the size of these clusters is one)

Developer clusters with KU-based expertise profile

Singleton cluster
KUs offer a fine-grained lens to study developers’ expertise. KUs identify 71 different clusters of developers.

The Gini index for clusters’ size is 0.96, indicating that there exists a very strong inequality.

57% of the generated clusters are singleton (i.e., the size of these clusters is one).

Each cluster hosts a set of developers with unique KU-based expertise profile.

Developer clusters with KU-based expertise profile
KUs offer a fine-grained lens to study developers’ expertise.

KUs identify 71 different clusters of developers.

The Gini index for clusters’ size is 0.96, indicating that there exists a very strong inequality.

57% of the generated clusters are singleton (i.e., the size of these clusters is one).

Each cluster hosts a set of developers with unique KU-based expertise profile.

Our encouraging results from the preliminary study motivate us to build a KU-based reviewer recommendation system (KUREC).
We address three research questions (RQs)

**RQ1:** How accurately can KUREC recommend code reviewers in pull requests?

**RQ2:** Can KUREC be made more accurate by combining it with existing recommenders?

**RQ3:** How reasonable are those recommendations of KUREC that are not matched with ground truth data?
We address three research questions (RQs)

**RQ1:** How accurately can KUREC recommend code reviewers in pull requests?

**RQ2:** Can KUREC be made more accurate by combining it with existing recommenders?

**RQ3:** How reasonable are those recommendations of KUREC that are not matched with ground truth data?
Our approach for building KUREC recommender
Our approach for building KUREC recommender

Commit Dataset

PR Dataset
Our approach for building KUREC recommender

1. Construct a KU-based development expertise profile for all developers using the Commit Dataset.
2. Construct a KU-based review expertise profile for all developers using the PR Dataset.

Calculate the expertise score of every developer and recommend top-k developers.

List of recommended developers of KUREC.
Our approach for building KUREC recommender

1. Construct a KU-based development expertise profile for all developers (Commit Dataset)
2. Construct a KU-based review expertise profile for all developers (PR Dataset)
3. Calculate the expertise score of every developer
4. Recommend top-\(k\) developers

List of recommended developers of KUREC
Our approach for building KUREC recommender

- Construct a KU-based development expertise profile for all developers (Commit Dataset)
- Construct a KU-based review expertise profile for all developers (PR Dataset)
- Calculate the expertise score of every developer
- Recommend top-k developers

List of recommended developers of KUREC
Our approach for building KUREC recommender

1. Construct a KU-based development expertise profile for all developers
2. Construct a KU-based review expertise profile for all developers
3. Calculate the expertise score of every developer
4. Recommend top-k developers
5. List of recommended developers of KUREC
We construct four baseline recommenders
We construct four baseline recommenders


The recommender sorts developers in decreasing order of commit counts and recommends the top-k ones.
We construct four baseline recommenders


The recommender sorts developers in decreasing order of commit counts and recommends the top-k ones


The recommender sorts developers in decreasing order of review counts and recommends the top-k ones
We construct four baseline recommenders


The recommender sorts developers in decreasing order of commit counts and recommends the top-k ones


The recommender sorts developers in decreasing order of review counts and recommends the top-k ones


The recommender sorts developers in reverse chronological order based on the date who last modified the changed file in a given PR. Finally, ER recommends the top-k ranked developers
We construct four baseline recommenders


The recommender sorts developers in decreasing order of commit counts and recommends the top-k ones.


The recommender sorts developers in decreasing order of review counts and recommends the top-k ones.


The recommender sorts developers in reverse chronological order based on the date who last modified the changed file in a given PR. Finally, ER recommends the top-k ranked developers.


CHREV distills review contribution into three measures:
1. total number of review comments
2. total number of workdays
3. recency of the review comments
CHREV generates a score for every developer based on these measures, sorts developers decreasing order of the score and recommends top-k ones.
We use two popular metrics to evaluate the performance of recommenders.
We use two popular metrics to evaluate the performance of recommenders.

Top-k Accuracy

\[ \text{Top-k accuracy} = \frac{\sum_{r \in R} \text{isCorrect}(r, \text{Top} - k)}{|R|} \]

Here, R denotes the set of PRs in the test dataset. The isCorrect(r, Top–k) returns 1 if at least one of top-k developers is the correct reviewer of the PR r and returns 0 otherwise.
We use two popular metrics to evaluate the performance of recommenders

**Top-k Accuracy**

\[
\text{Top-k accuracy} = \frac{\sum_{r \in R} \text{isCorrect}(r, \text{Top-k})}{|R|}
\]

Here, \( R \) denotes the set of PRs in the test dataset. The \( \text{isCorrect}(r, \text{Top-k}) \) returns 1 if at least one of top-k developers is the correct reviewer of the PR \( r \) and returns 0 otherwise.

**Mean Average Precision (MAP)**

\[
\text{AP} @ k = \frac{\sum_{i=1}^{k} \frac{s(i)}{i} \times \text{rel}(i)}{\sum_{i=1}^{k} \text{rel}(i)}
\]

Here, \( i \) is the position of each developer in the recommended list of developers, and \( s(i) \) is the sequence number of the correct developer at position \( i \). The \( \text{rel}(i) \) returns 1 if the \( i \)th developer in the list is correct and 0 otherwise.

MAP @\( k \) is the average of AP@\( k \) over all the PRs in the test dataset.
KUREC is more stable than RF and outperforms the remaining three baselines.
KUREC is more stable than RF and outperforms the remaining three baselines.
The top Scott-Knott ESD-ranked recommenders are KUREC and RF in both top-k accuracy and MAP.

KUREC is more stable than RF and outperforms the remaining three baselines.
The top Scott-Knott ESD-ranked recommenders are KUREC and RF in both top-k accuracy and MAP.

KUREC is more stable than RF and outperforms the remaining three baselines.
KUREC is more stable than RF and outperforms the remaining three baselines. The top Scott-Knott ESD-ranked recommenders are KUREC and RF in both top-k accuracy and MAP. The interquartile range of KUREC is smaller than RF in both metrics. For MAP, the interquartile range of KUREC is 0.25 whereas the interquartile range of RF is 0.56.
The top Scott-Knott ESD-ranked recommenders are **KUREC and RF** in both **top-k accuracy and MAP**

The interquartile range of KUREC is smaller than RF in both metrics. For MAP, the interquartile range of KUREC is 0.25 whereas the interquartile range of RF is 0.56.

KUREC is more stable than RF and outperforms the remaining three baselines.
KUREC is more stable than RF in Top-K accuracy

The top Scott-Knott ESD-ranked recommenders are KUREC and RF in both top-k accuracy and MAP

The interquartile range of KUREC is smaller than RF in both metrics

KUREC is more stable than RF and outperforms the remaining three baselines
KUREC outperforms the remaining three baselines and has a more stable performance compared to RF, which is a desired property in practice.
We address three research questions (RQs)

**RQ1:** How accurately can KUREC recommend code reviewers in pull requests?

**RQ2:** Can KUREC be made more accurate by combining it with existing recommenders?

**RQ3:** How reasonable are those recommendations of KUREC that are not matched with ground truth data?
We construct **three recommenders** by combining KUREC with the baseline recommenders
We construct three recommenders by combining KUREC with the baseline recommenders.

To construct a combined recommender by leveraging the recommendations of different recommenders, we are motivated by the work of Malik and Hassan [1].

We construct **three recommenders** by combining KUREC with the baseline recommenders.

To construct a combined recommender by leveraging the recommendations of different recommenders, we are **motivated by the work of Malik and Hassan** [1].

In this approach, all the recommenders uses a Best Recommender System Table (BRST) to track the best-performing recommender.

We construct **three recommenders** by combining KUREC with the baseline recommenders.

To construct a combined recommender by leveraging the recommendations of different recommenders, we are **motivated by the work of Malik and Hassan** [1].

In this approach, all the recommenders uses a Best Recommender System Table (BRST) to track the best-performing recommender.

We implement three techniques to update the BRST and these are our new recommenders based on heuristics.

We construct **three recommenders** by combining KUREC with the baseline recommenders.

(1) Adaptive Frequency Technique (AD_FREQ)

The BRST stores the frequency of each recommender that becomes the best performing recommender. The recommender with the highest count is selected for recommendation.
We construct **three recommenders** by combining KUREC with the baseline recommenders

(1) Adaptive Frequency Technique (AD_FREQ)

The BRST stores the frequency of each recommender that becomes the best performing recommender. The recommender with the highest count is selected for recommendation.

(2) Adaptive Recency Technique (AD_REC)

The BRST stores the best-performing recommender that is identified in the last PR.
We construct three recommenders by combining KUREC with the baseline recommenders

(1) Adaptive Frequency Technique (AD_FREQ)
The BRST stores the frequency of each recommender that becomes the best performing recommender. The recommender with the highest count is selected for recommendation.

(2) Adaptive Recency Technique (AD_REC)
The BRST stores the best-performing recommender that is identified in the last PR.

(3) Adaptive Hybrid Technique (AD_HYBRID)
We select the recommender that has the highest count in the BRST among the last 10 previous PRs.
All the combined recommenders outperform individual recommenders
All the combined recommenders outperform individual recommenders.
All the combined recommenders outperform individual recommenders.
Combining the KU-based recommender (KUREC) with the baselines in a straight-forward manner results in better-performing recommenders.
We address three research questions (RQs)

**RQ1:** How accurately can KUREC recommend code reviewers in pull requests?

**RQ2:** Can KUREC be made more accurate by combining it with existing recommenders?

**RQ3:** How reasonable are those recommendations of KUREC that are not matched with ground truth data?
We study the recommendations of a recommender that does not match with ground truth data.
We study the recommendations of a recommender that does not match with ground truth data.
We study the recommendations of a recommender that does not match with ground truth data.

**Pull Request**

**Recommender System (e.g., KUREC)**

**Actual Reviewers (Ground Truth data)**

1. X
2. Y
3. Z
We study the recommendations of a recommender that does not match with ground truth data.

Actual Reviewers (Ground Truth data):
1. X
2. Y
3. Z

Recommended Reviewers:
1. P
2. Q
3. R

Pull Request

Recommender System (e.g., KUREC)
We study the recommendations of a recommender that does not match with ground truth data.
We study the recommendations of a recommender that does not match with ground truth data.
We consider a recommendation to be **reasonable** if the recommended individual had recent (last six months) development experience with the majority (50%) of the files included in the PR in question.
AD_FREQ strikes the best balance between sticking to the ground truth and reasonable recommendations.

<table>
<thead>
<tr>
<th>Recommender</th>
<th>Percentage of Reasonable recommendations</th>
</tr>
</thead>
<tbody>
<tr>
<td>KUREC</td>
<td>63.4%</td>
</tr>
<tr>
<td>ER</td>
<td>60.9%</td>
</tr>
<tr>
<td>AD_FREQ</td>
<td>59.4%</td>
</tr>
<tr>
<td>AD_HYBRID</td>
<td>54.3%</td>
</tr>
<tr>
<td>AD_REC</td>
<td>54.2%</td>
</tr>
<tr>
<td>CHREV</td>
<td>32.7%</td>
</tr>
<tr>
<td>CF</td>
<td>25.4%</td>
</tr>
<tr>
<td>RF</td>
<td>15.2%</td>
</tr>
</tbody>
</table>
AD_FREQ strikes the best balance between sticking to the ground truth and reasonable recommendations.

<table>
<thead>
<tr>
<th>Recommender</th>
<th>Percentage of Reasonable recommendations</th>
</tr>
</thead>
<tbody>
<tr>
<td>KUREC</td>
<td>63.4%</td>
</tr>
<tr>
<td>ER</td>
<td>60.9%</td>
</tr>
<tr>
<td>AD_FREQ</td>
<td>59.4%</td>
</tr>
<tr>
<td>AD_HYBRID</td>
<td>54.3%</td>
</tr>
<tr>
<td>AD_REC</td>
<td>54.2%</td>
</tr>
<tr>
<td>CHREV</td>
<td>32.7%</td>
</tr>
<tr>
<td>CF</td>
<td>25.4%</td>
</tr>
<tr>
<td>RF</td>
<td>15.2%</td>
</tr>
</tbody>
</table>
**AD_FREQ** strikes the best balance between sticking to the ground truth and reasonable recommendations.

<table>
<thead>
<tr>
<th>Recommender</th>
<th>Percentage of Reasonable recommendations</th>
</tr>
</thead>
<tbody>
<tr>
<td>KUREC</td>
<td>63.4%</td>
</tr>
<tr>
<td>ER</td>
<td>60.9%</td>
</tr>
<tr>
<td>AD_FREQ</td>
<td>59.4%</td>
</tr>
<tr>
<td>AD_HYBRID</td>
<td>54.3%</td>
</tr>
<tr>
<td>AD_REC</td>
<td>54.2%</td>
</tr>
<tr>
<td>CHREV</td>
<td>32.7%</td>
</tr>
<tr>
<td>CF</td>
<td>25.4%</td>
</tr>
<tr>
<td>RF</td>
<td>15.2%</td>
</tr>
</tbody>
</table>

The best-performing baseline RF is the lowest in reasonable recommendation.
**AD_FREQ** strikes the best balance between sticking to the ground truth and reasonable recommendations.

<table>
<thead>
<tr>
<th>Recommender</th>
<th>Percentage of Reasonable recommendations</th>
</tr>
</thead>
<tbody>
<tr>
<td>KUREC</td>
<td>63.4%</td>
</tr>
<tr>
<td>ER</td>
<td>60.9%</td>
</tr>
<tr>
<td><strong>AD_FREQ</strong></td>
<td><strong>59.4%</strong></td>
</tr>
<tr>
<td>AD_HYBRID</td>
<td>54.3%</td>
</tr>
<tr>
<td>AD_REC</td>
<td>54.2%</td>
</tr>
<tr>
<td>CHREV</td>
<td>32.7%</td>
</tr>
<tr>
<td>CF</td>
<td>25.4%</td>
</tr>
<tr>
<td>RF</td>
<td>15.2%</td>
</tr>
</tbody>
</table>

The best-performing baseline RF is the lowest in reasonable recommendation.
KUREC is the recommender with the highest percentage of reasonable recommendations. Yet, AD_FREQ strikes the best balance between sticking to the ground truth and issuing reasonable recommendations when those deviate from that ground truth.
We address three research questions (RQs)

**RQ1:** How accurately can KUREC recommend code reviewers in pull requests?

**RQ2:** Can KUREC be made more accurate by combining it with existing recommenders?

**RQ3:** How reasonable are those recommendations of KUREC that are not matched with ground truth data?
We address three research questions (RQs)

**RQ1:** How accurately can KUREC recommend code reviewers in pull requests?

**RQ2:** Can KUREC be made more accurate by combining it with existing recommenders?

**RQ3:** How reasonable are those recommendations of KUREC that are not matched with ground truth data?

Summary of RQ1

KUREC outperforms the remaining three baselines and has a more stable performance compared to RF, which is a desired property in practice.
We address three research questions (RQs)

RQ1: How accurately can KUREC recommend code reviewers in pull requests?

RQ2: Can KUREC be made more accurate by combining it with existing recommenders?

RQ3: How reasonable are those recommendations of KUREC that are not matched with ground truth data?

Summary of RQ1

KUREC outperforms the remaining three baselines and has a more stable performance compared to RF, which is a desired property in practice.

Summary of RQ2

Combining the KU-based recommender (KUREC) with the baselines in a straightforward manner results in better-performing recommenders.
We address three research questions (RQs)

**RQ1**: How accurately can KUREC recommend code reviewers in pull requests?

**RQ2**: Can KUREC be made more accurate by combining it with existing recommenders?

**RQ3**: How reasonable are those recommendations of KUREC that are not matched with ground truth data?

Summary of RQ1:

KUREC outperforms the remaining three baselines and has a more stable performance compared to RF, which is a desired property in practice.

Summary of RQ2:

Combining the KU-based recommender (KUREC) with the baselines in a straight-forward manner results in better-performing recommenders.

Summary of RQ3:

KUREC is the recommender with the highest percentage of reasonable recommendations. Yet, AD_FREQ strikes the best balance between sticking to the ground truth and issuing reasonable recommendations when those deviate from that ground truth.
We address three research questions (RQs)

**RQ1:** How accurately can KUREC recommend code reviewers in pull requests?

**RQ2:** Can KUREC be made more accurate by combining it with existing recommenders?

**RQ3:** How reasonable are those recommendations that are not matched with ground truth?

**Summary of RQ1**

KUREC outperforms the remaining three baselines and has a more stable performance compared to RF, which is a desired property in practice.

**Summary of RQ2**

Combining the KU-based recommender (KUREC) with the baselines in a straightforward manner results in better-performing recommenders.

**Summary of RQ3**

While some recommenders with the highest percentage of reasonable recommendations. Yet, AD_FREQ strikes the best balance between sticking to the ground truth and issuing reasonable recommendations when those deviate from that ground truth.