

Automated Recommendation of Programming Learning Content Using Pattern-based Knowledge Components

Muntasir Hoq¹, Griffin Pitts¹, Zhangqi Duan², Arun Balajiee Lekshmi Narayanan³,
Mohammad Hassany³, Andrew Lan², Peter Brusilovsky³ and Bitu Akram^{1,*}

¹North Carolina State University, Raleigh, NC, USA

²University of Massachusetts, Amherst, MA, USA

³University of Pittsburgh, Pittsburgh, PA, USA

Abstract

Introductory programming instruction relies on hands-on practice and short learning activities to support mastery of foundational concepts. Although many such learning resources exist, organizing and linking these items in instructionally meaningful ways is challenging without time-intensive expert curation. This study investigates the use of pattern-based Knowledge Components (KCs) to automatically identify code-based learning resources targeting similar concepts. In our approach, pattern-based KCs are extracted from each code sample, and related activities are identified by measuring similarity between the KC sets associated with each activity. By leveraging alignment at the level of semantically important programming patterns, this method supports contextually appropriate and pedagogically useful recommendations. We evaluate our approach on an expert-organized corpus of introductory Python materials in which instructors grouped items into bundles based on conceptual similarity. Results show that our pattern-based KC approach retrieves resources that align with this expert organization, and outperformed representative KC- and embedding-based baselines across standard ranking evaluations. Overall, the framework supports targeted, concept-oriented guidance for programming learners and can help instructors organize, bundle, and recommend instructional content at scale.

Keywords

Programming education, knowledge components, explainable recommendations, educational recommendations

1. Introduction

Example-based problem solving has shown promise for improving students' learning across educational domains [1]. In computer science (CS) education, one common activity type is a "worked example". In a worked example, a correct solution of a programming problem (i.e., code) is presented to students along with instructional explanations about each coding step that can be revealed as needed [2, 3]. Timely exposure to a relevant example can help students deepen their understanding and progress when they encounter difficulty [2, 3, 4]. Complementing worked examples, introductory programming environments often provide practice-oriented activities that challenge students to apply what they know, for instance, by completing incomplete code [2], or arranging code lines in a correct order [5].

Prior work suggests that alternating between types of activities focusing on similar concepts can be beneficial for students, especially when semantic similarity between activities is curated by experts [6]. For example, a study using the Program Construction Examples (PCEX) system, bundling worked examples with closely related completion problems, increased engagement and improved students' performance [2]. Subsequent studies confirmed that semantic similarity between a practice-oriented activity and its paired worked example is a driver of problem-solving success and persistence [6].

Though an abundance of programming activities are available across repositories, it remains difficult to maintain links among activities that target similar programming concepts and provide learners with relevant items at the moment of need [7]. Instructors have traditionally addressed this by manually linking items, such as assigning practice problems with examples [8]. However, human curation does not scale well. In large repositories, maintaining these links becomes slow, difficult to update, and prone to inconsistency. Early automated approaches mostly used surface signals, such as keywords

CSEDN'26: 10th Educational Data Mining in Computer Science Education (CSEDN) Workshop, June 28, 2026, Seoul, South Korea

*Corresponding author: Bitu Akram, bakram@ncsu.edu



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

or n-grams [9, 10]. These features are computationally efficient, but they often match code that appears similar while targeting different underlying concepts, or fail to connect conceptually similar solutions expressed with different syntax. Knowledge-based approaches, such as ontology-based linking, improved relevance by indexing items with domain concepts and accounting for relationships among those concepts [11]. However, these methods typically require hand-engineered domain concept models and extensive expert annotation [12, 13]. This motivates learning activity representations that are grounded in program structure and do not require extensive expert annotation.

Motivated by this gap, this paper presents a knowledge component (KC)-based recommendation framework that derives similarity from program representations in terms of the KCs they represent. We hypothesize that KCs, defined as the skills or concepts a learner must apply to solve a task, provide an appropriate level of abstraction for recommending instructional content across worked examples and practice activities [13]. Matching at the KC level prioritizes items that reflect similar conceptual understanding required by a target activity. We adopt a pattern-based KC formulation [14] in which KCs are recurring, semantically meaningful subtrees drawn from program Abstract Syntax Trees (ASTs).

In our implementation, we discover candidate pattern-based KCs from a repository of programming activities using an explainable extraction method proposed in prior work [14]. Each program is represented as an IDF-weighted knowledge vector over the resulting KC set, and candidate items are ranked using cosine similarity. The system can also project the highest-contributing KCs back into source code, yielding line-level rationales that allow instructors to audit why a recommendation was made. To evaluate recommendation quality, we use instructional content from PCEX [2], an online tool for interactive program learning. In PCEX, instructors organized worked examples and practice activities into topics and finer-grained bundles, which we treat as ground truth for assessing retrieval quality. We report Top-5 accuracy, mean reciprocal rank (MRR), and mean average precision (mAP) across two retrieval directions: retrieving worked examples for a given problem and retrieving problems for a given worked example. We also perform neighborhood tightness and clustering analyses to examine whether items from the same expert-defined bundle are placed closer together in the learned representation space. Across analyses, the pattern-based KC representation retrieves instructionally aligned items and compares favorably with code-similarity methods and alternative KC discovery baselines, including code2vec [15], SANN [16], ontology-based approaches [11], and recent LLM-identified KC approaches [17, 18, 19]. We further demonstrate the explainability of the proposed framework through a proof-of-concept example.

Our contributions are threefold: (i) a recommendation approach based on knowledge vectors that uses KC similarity across programming activities, (ii) an explainable pattern-based KC model that discovers and identifies programming patterns for recommendation, and (iii) an evaluation of how different KC models improve recommendation accuracy and alignment with expert-defined bundles.

2. Related Work

Early efforts to connect related learning materials emerged in educational hypertext under similarity-based navigation, where items were often matched by keyword overlap [9, 10]. Later work introduced richer semantic approaches, often termed intelligent linking, including methods that compute similarity over metadata [20]. Related work on ontology-based linking indexed resources with ontology terms and computed similarity from their positions and relations within an ontology graph [12, 21, 22].

In CS education, similarity estimation has often focused on program content. Gross et al. [23], for example, linked Java materials using similarity between full-program abstract syntax trees (ASTs), though whole-program AST comparison can miss fine-grained correspondences in smaller structural units [24]. Concept-based approaches address this limitation by representing problems and examples as vectors over domain concepts and computing similarity between these profiles [25]. Subsequent work extended this direction through ontology-informed similarity metrics for programming items [11] and recommendation methods for supplementing introductory programming textbooks with reusable learning content [26]. However, automatically allocating and sequencing such resources remains

```
#Step 1: Iterate through the numbers in the sequence
for num in range(1, 10):
    #Step 1.1: Print each number in the sequence
    print(num)
```

(a) Same bundle code (Ex. 1)

```
#Step 1: Iterate through the numbers in the sequence
for num in range(8, 15):
    #Step 1.1: Print each number in the sequence
    print(num)
```

(b) Same bundle code (Ex. 2)

```
#Step 1: Iterate through the numbers in the sequence
for num in range(1, 10, 2):
    #Step 1.1: Square and print each number in the
    #sequence
    print(num, "squared =", num * num)
```

(c) Different bundle code (Ex. 3)

Figure 1: Examples of recommendations with shared and different bundle-code patterns.

challenging, especially when course ordering varies across instructors [27].

Recent advances in learned code representations have opened a new avenue for automatically extracting syntactic and semantic structure from programs using deep learning models [28]. These representations reduce reliance on manual expert design of isomorphic problem-example pairs by enabling similarity search directly in the learned embedding space. Prior work shows that such methods can support a range of code-analysis tasks, including recommending similar learning content [29, 16]. However, these approaches primarily capture surface- or structure-level similarity between programs. In this work, we propose a KC-based recommendation framework grounded in the programming patterns required to solve a task. We evaluate this pattern-based KC framework against both code-similarity approaches and alternative KC representations.

3. Dataset

This work uses a Python programming dataset collected from the PCEX system [2]. PCEX provides online access to worked code examples and short code completion tasks (“challenges”). Domain experts have organized these items into “**bundles**”: compact groups of challenges and examples that target the same programming constructs and recurring patterns. Prior studies have validated expert-defined bundles [6, 2], noting the value of aligning practice and assessment around shared, recurring programming patterns. Although constructing such bundles is time- and expertise-intensive, this limitation motivates the need for approaches that can support or scale this alignment.

In PCEX, the dataset contains 123 programs (examples and challenges) spanning 13 topics, including Variables and Operations, If-Else, Boolean Expressions, For Loops, and Nested Loops. These items are partitioned into 49 bundles (approximately four bundles per topic). A typical bundle consists of one worked example and one to three closely related challenges for the students; on average, each bundle includes 1.35 challenges. We treat the existing topic and bundle assignments in PCEX, produced by expert instructors, as gold-standard labels for evaluating recommendation and clustering quality [27]. Figures 1a and 1b illustrate two examples from the same bundle within the “For Loops” topic, while Figure 1c shows an example from a different bundle in the same topic, emphasizing the structural distinctions that motivate bundle-level grouping.

4. Methodology

In this section, we describe our pipeline for recommending programming learning activities by matching items based on the patterns expressed in their associated code. We first extract *pattern-based* KCs from each activity’s code sample. Each activity is then mapped to an IDF-weighted knowledge vector, and recommendations are produced by cosine similarity, supporting retrieval, as illustrated in Figure 2. We

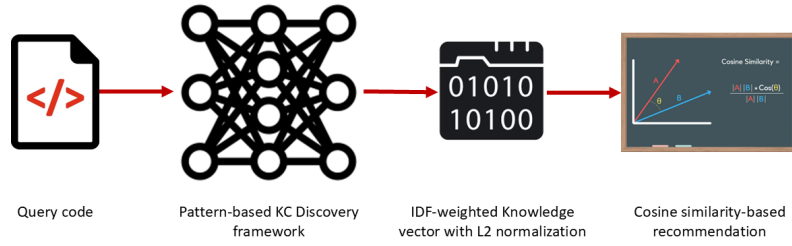


Figure 2: Learning activity recommendation pipeline.

apply the KC discovery framework of [14] to code artifacts in the PCEX dataset, producing (i) a set of structural KCs and (ii) a mapping from each learning activity to the KCs it expresses.

4.1. Pattern-based Knowledge Component Discovery

Automatically discovering KCs from code is not a trivial task. In programming, small syntactic edits can obscure semantic equivalence, the same conceptual skill can be realized through many structurally distinct implementations, and the instructional relevance of a code fragment often depends on its surrounding program context rather than the fragment in isolation [30].

To address this, we adopt pattern-based KCs [14], which define KCs as recurring, semantically meaningful structural patterns based on important subtrees from program Abstract Syntax Trees (ASTs). These subtree patterns capture programming constructs that recur across solutions, and their correct implementation is crucial for completing the assignments [14]. Compared with KC models based on named concepts or curricular tags, pattern-based KCs provide finer-grained, implementation-level signals (e.g., guarded loop traversals, accumulator updates, boundary checks). This granularity is well matched to recommendations, since it allows the system to retrieve items that share structural patterns while also supporting line-level explanations of why an item was retrieved. Four steps of pattern-based KC discovery are specified below.

First, *subtree pattern extraction*. Each program is converted into an AST using the Python AST parser¹. Subtrees are extracted at multiple granularity levels, and a modified Subtree-based Attention Neural Network (SANN) [31] assigns an attention weight to each subtree. These weights identify structural fragments that most strongly represent topic-relevant logic, and the most important subtrees per program are retained as candidate patterns. We train SANN using only coarse-grained topic labels for challenges and examples, deliberately withholding expert-defined bundle information. Although bundles encode finer-grained structure, our aim was to test whether the KC discovery framework can recover bundle-level regularities from coarser supervision through learned pattern similarity. This ensures that the extracted patterns do not trivially encode bundle membership, but reflect recurring structural regularities that generalize across problems within a topic. Data is split into 80% training, 10% validation, and 10% test sets, ensuring that every expert-defined bundle appears in the training split to preserve structural variation. Subtree embeddings use 64 dimensions (selected from 64, 128, 256). Models are trained with batch size 32, up to 200 epochs, and early stopping with patience 20. Remaining hyperparameters follow prior work [14].

Second, *normalization*. We abstract selected subtrees by anonymizing identifiers, literal values, and other incidental tokens. We also canonicalize simple syntactic alternatives, such as commutative operator order, so that surface-different but conceptually equivalent code maps to the same abstract pattern. This step helps ensure that pattern-based KCs reflect conceptual similarity instead of lexical similarity. Introductory programming problems often admit multiple correct implementations that differ in variable naming, literal values, or syntactic choices, none of which should affect KC attribution.

Third, *context-aware encoding*. While individual subtree patterns capture local structural information, their instructional meaning often depends on how they are composed within a complete program. For

¹<https://docs.python.org/3/library/ast.html>

example, the same conditional structure may represent a boundary check, a classification decision, or part of an accumulator update, depending on its surrounding logic. To model these dependencies, we encode sequences of normalized subtrees using a context-aware latent representation. A β -variational autoencoder (β -VAE), using an LSTM, is used to encode the normalized subtrees to preserve how patterns co-occur within complete programs and across submissions, and learn a latent space in which structurally and semantically similar patterns lie close together. The β -VAE is trained to encode representative patterns for the important subtrees, guided by attention weights extracted from SANN. Following [14], the β -VAE model employed subtree embeddings of dimension 64 and a latent space of dimension 128, with intermediate dense layers matching the embedding size. To mitigate overfitting, a dropout rate of 0.2 was applied. Training was performed for up to 50 epochs using the Adamax optimizer with a learning rate of 0.001, and early stopping with a patience of 10 epochs. We adopted a β -VAE objective with a KL-divergence weighting factor of $\beta = 1 \times 10^{-2}$ to balance reconstruction accuracy and latent space regularization.

Fourth, *clustering into KCs*. We cluster the latent vectors of the retained subtrees (with an attention threshold of 20%, following [14]) from programs using K-means (with the number of clusters selected by the silhouette heuristic). The resulting clusters form our pattern-based KCs, with centroids serving as canonical representatives and links preserved back to the original AST spans for explainability. Next, we use these KC assignments to compute similarity between learning activities for recommendation.

4.2. Knowledge Vectors and Recommendation

After discovering pattern-based KCs, we create a *knowledge vector* that captures the presence of each KC for each code sample. For a corpus with K KCs, each program d is mapped to a binary incidence vector $\mathbf{x}_d \in \{0, 1\}^K$ indicating KC presence. We then apply an inverse-document-frequency (IDF) reweighting to emphasize discriminative patterns and downweight ubiquitous ones. Let N be the number of programs and n_i the number of programs that contain KC i . The weight for KC i is

$$w_i = \log\left(\frac{N}{n_i}\right).$$

The resulting weighted representation is $\mathbf{v}_d = \mathbf{x}_d \odot \mathbf{w}$, which we L2-normalize, so that dot products correspond to cosine similarity:

$$\hat{\mathbf{v}}_d = \frac{\mathbf{v}_d}{\|\mathbf{v}_d\|_2}.$$

Given a target item q (challenge or worked example), we rank candidate items e by cosine similarity

$$s(q, e) = \hat{\mathbf{v}}_q^\top \hat{\mathbf{v}}_e,$$

and return the top-ranked items as recommendations. Because similarity is computed in the pattern-based KC latent space, retrieval favors candidates that instantiate similar underlying program patterns expressed in q , rather than items that are merely lexically or syntactically similar.

We evaluate this approach on two retrieval tasks using the held-out test set: (1) retrieving worked examples for a target challenge and (2) retrieving challenges for a target worked example. A retrieved item is considered relevant if it belongs to the *same expert-defined bundle* as the target item (our gold standard), with self-matches prevented. We report Top-5 accuracy, mean reciprocal rank (MRR), and mean average precision (mAP) to assess whether pattern-based KC-level similarity recovers bundle-aligned items relative to baselines.

4.3. Baseline Approaches

As baselines, we compare against (1) code2vec-based code vectorization [15], (2) ontology concept-based code vectorization [11], (3) SANN-based code vectorization [16], (4) KCGEN-KT KCs [17], and (5) LLM-KCI KCs [18, 19].

code2vec. code2vec learns a vector representation of a code snippet by decomposing its abstract syntax tree into many “path-contexts.” Each path-context is formed by two terminal tokens together with the AST path connecting them, written as (token_s, path, token_t). The model embeds the tokens and the path and then uses an attention mechanism to weight and aggregate the set of path-context embeddings into a single fixed-length vector for the snippet.

Ontology-KC. Ontology-based concept vectorization represents programming items using concept annotations derived from their source code. Following prior work [11], a set of high-level programming concepts from each code submission was extracted using an open source static code parser² using an expert-defined programming ontology³. Each submission is then encoded as a concept-based vector, and similarity between submissions is computed based on cosine similarity.

SANN. We treat SANN as a standalone code-embedding model and use only its learned source-code vectors derived from AST structure (no topic, bundle, or knowledge signals beyond what is implicit in the encoder). Each challenge and worked example is embedded once with SANN, and recommendations are produced by cosine similarity in this SANN embedding space, following [16].

KCGen-KT. KCGen-KT [17] represents each item with a set of LLM-generated KCs. It prompts GPT-4o to propose candidate KCs from the problem statement and canonical solution, clusters semantically similar candidates into KC groups, and summarizes each cluster into a consolidated KC description, yielding 30 unique KCs. These KCs are assigned as tags to items to form a KC-based representation.

LLM-KCI. LLM-KCI [18] indexes programming items by inducing an expert-curated KC inventory with an LLM and then using that inventory to assign KCs to items. It prompts GPT-4o to generate candidate KCs from the available item context, aggregates and removes semantic duplicates to form a KC list, and then prompts the LLM with this list to select the most relevant KCs for each item, yielding a sparse KC assignment vector used for similarity-based retrieval.

5. Experiments and Results

To evaluate whether our pattern-based KC representation supports accurate recommendations of programming learning activities, we first examine the alignment of the knowledge vector using pattern-based KCs with expert-defined topics and bundles using tightness analyses. As described in Section 3, we evaluate our recommendation approach on the PCEX instructional repository [2]. We test two retrieval directions that reflect common instructional needs, recommending worked examples for a target challenge activity and recommending challenge activities for a target worked example. We treat expert-defined bundle membership as ground truth and report Top-5 accuracy, mean reciprocal rank (MRR), and mean average precision (mAP). We also illustrate explainability with visualizations that project the highest-contributing pattern-based KCs back onto code to explain specific recommendations, and include an ablation on IDF weighting to isolate its contribution.

5.1. Knowledge Vector Alignment with Expert-Defined Topics and Bundles

We use expert-defined bundles as our ground truth and hypothesize that these bundles capture fine-grained similarity among challenges and worked examples. Representations derived from the knowledge vectors should therefore place items from the same bundle closer together than items from different bundles. The same intuition extends to topics at a coarser granularity: items within a topic should be nearer to one another than to items from other topics, but not as tightly grouped as items within a bundle. To test this hypothesis, we quantify tightness as the mean pairwise cosine distance among vectors within a group, with lower values indicating tighter, more coherent groups. Using expert labels as our standard, we computed tightness separately for each bundle and each topic.

Bundles consistently exhibit smaller average distances than topics, indicating more compact clusters at the bundle level. The average bundle tightness was 0.51 (cosine-distance units), compared to 0.81 for

²<https://acos.cs.aalto.fi/python-parser>

³<https://acos.cs.aalto.fi/static/python-parser/ontology.png>

Table 1

Performance of different KC modeling approaches for recommending worked examples and challenges. Higher is better for all metrics.

Model	Worked-example retrieval			Challenge retrieval		
	Top-5	MRR	mAP	Top-5	MRR	mAP
LLM-KCI	0.87	0.79	0.79	0.88	0.78	0.78
KCGen-KT KC	0.84	0.62	0.62	0.85	0.64	0.65
Ontology-based KC	0.88	0.67	0.65	0.89	0.66	0.65
code2vec embeddings	0.83	0.71	0.71	0.82	0.70	0.71
SANN embeddings	0.87	0.78	0.78	0.87	0.77	0.78
Pattern-based KC	0.89	0.81	0.82	0.90	0.80	0.79

topics, showing that bundle members are considerably closer to one another than items that merely share a topic. The dataset-wide average distance was 1.7, reflecting the broader diversity across programs and reinforcing that bundle membership captures highly similar code patterns, whereas topic membership reflects a looser but still meaningful similarity.

5.2. Clustering Bundles without Expert Effort

To assess whether the knowledge vectors recover the dataset’s latent bundle organization without manual expert curation, we performed agglomerative hierarchical clustering with cosine distance and complete linkage on the knowledge vectors. The dendrogram preserved the original pairwise geometry well (cophenetic correlation, $r=0.819$. Values closer to 1 mean the tree faithfully reflects the distances). We then evaluated alignment with expert bundlings *post hoc* by sweeping distance cut thresholds and computing Adjusted Rand Index (ARI) and V-measure (the harmonic mean of homogeneity and completeness) for the induced flat partitions. At the *bundle* level, the best ARI was 0.605 at 0.6 threshold with 63 clusters, indicating substantial agreement beyond chance. The accompanying V-measure was 0.933 (homogeneity 0.949 and completeness 0.916), suggesting clusters were both internally pure and capture most items of each expert bundle.

Additionally, we quantified local neighborhood structure by identifying, for every program, its single nearest neighbor in the clustering (excluding self-matches) and categorizing the match as: (1) *same bundle*, (2) *same topic but different bundle*, or (3) *different topic*. The distribution of nearest-neighbor types was: 76.4% same bundle, 11.9% same topic/different bundle, and 11.7% different topic. The high same-bundle rate indicates that knowledge vectors place structurally similar problems and examples in close proximity, while same-topic matches capture looser but meaningful commonalities across bundles within a topic. Further inspection of mismatches indicates that they often reflect broadly used programming patterns, such as common control-flow in Loops, Nested Loops, and Strings, that legitimately span topic boundaries. This analysis shows that knowledge vectors can automatically recover bundle-level groupings within topics and expose finer substructure, reducing the need for manual pre-linking of learning content.

5.3. KC-based Recommendation of Learning Content

We first report retrieval performance under the KC-based similarity model. For each target item, we rank candidate items by cosine similarity of their knowledge vectors. A recommendation is counted correct when at least one of the top five retrieved items belongs to the same expert-defined bundle as the target item. Table 1 reports Top-5 accuracy, mean reciprocal rank (MRR), and mean average precision (mAP) for both retrieval directions across all baselines.

The pattern-based KC approach achieves the strongest retrieval performance in both directions, indicating that our KC-based retrieval method, which combines pattern-based KCs with IDF-weighted

knowledge vectors, more reliably ranks bundle-aligned items ahead of learned code embeddings and other KC-based baselines. Improvements in MRR and mAP further suggest that relevant items tend to appear earlier in the ranking, which is important in practice because instructional systems typically surface only the top one or two recommendations to learners.

For worked-example retrieval, the pattern-based KC approach achieves Top-5 accuracy of 0.89%, MRR= 0.81%, and mAP= 0.82%, outperforming code2vec embeddings (Top-5= 0.83%, MRR= 0.71%, mAP= 0.71%), ontology-based KCs (Top-5= 0.88%, MRR= 0.67%, mAP= 0.65%), and the remaining baselines. For challenge retrieval, the pattern-based KC approach again ranks best with Top-5= 0.90%, MRR= 0.80%, and mAP= 0.79% (Table 1).

We also conducted an ablation that removes IDF reweighting when constructing knowledge vectors for our pattern-based KC approach. In this setting, each program is represented as a binary KC-incidence vector indicating which KCs are present. Without IDF, worked-example retrieval achieves Top-5 accuracy of 88%, mAP of 72%, and MRR of 76%, and the challenge retrieval achieves Top-5 accuracy of 85%, mAP of 68%, and MRR of 75%. Compared with the IDF-weighted knowledge vectors, IDF yields a +1% percentage point (pp) gain in Top-5 accuracy for worked-example retrieval (89% vs. 88%), a +5% pp gain in MRR (81% vs. 76%), and a +10% pp gain in mAP (82% vs. 72%). For challenge retrieval, IDF yields a +5 pp gain in Top-5 accuracy (90% vs. 85%), a +5% pp gain in MRR (80% vs. 75%), and a +11 pp gain in mAP (79% vs. 68%), supporting the value of down-weighting ubiquitous KCs.

5.4. Explainability of KC-based Recommendations

A benefit of recommending at the level of programming patterns is that the rationale for a match is explicit, with pattern-based KCs emphasizing instructionally important patterns. For example, any targeted *challenge* q and candidate *example* e , their knowledge vectors $\mathbf{v}_q$ and $\mathbf{v}_e$ decompose the cosine score into per-KC contribution, $s(q, e)$, as:

$$s(q, e) = \hat{\mathbf{v}}_q^\top \hat{\mathbf{v}}_e = \sum_{i=1}^K \hat{v}_{q,i} \hat{v}_{e,i},$$

The KCs with the largest products $\hat{v}_{q,i} \hat{v}_{e,i}$ are the primary drivers of the recommendation. To visualize this rationale, we trace each high-contributing KC back through the extraction pipeline to the high-attention subtree instances associated with that KC, then map those instances to their code spans. We highlight these spans with color intensity proportional to the underlying attention weight. This provides a line-level view of *which patterns* influenced the match, without implying that raw AST subtrees are directly clustered or compared during retrieval. Prior work has validated that these high-attention subtrees capture instructionally meaningful evidence related to program correctness and logical errors, supporting their use as explanatory units [31].

Figures 3a and 3b illustrate an explanation from the same bundle within the *If-Else* topic. In the challenge (left), attention concentrates on the binary conditional structure used to determine whether an input integer is even or odd, highlighting the modulus-based predicate and the associated if-else branching logic. In the recommended worked example (right), attention emphasizes the conditional decision structure used to classify an integer as positive, negative, or zero, which involves a multi-branch if-elif-else construct. Although the specific predicates and the number of branches differ, both instantiate the same underlying control-flow schema—sequential input processing followed by mutually exclusive conditional evaluation and branching—explaining why the worked example is retrieved. In contrast, scaffolding elements such as input prompts and output statements receive little or no highlight.

These explanations are designed to support *pattern-focused learning*. For students, the highlighted regions direct attention to the abstract decision-making schema underlying conditional logic, such as evaluating a predicate and selecting among mutually exclusive branches. This encourages learners to internalize the general structure of conditional reasoning and transfer it across problems. For instructors, KC-level contribution breakdowns provide an auditable account of why a particular example was retrieved, help diagnose failure modes, and support inspection and maintenance of the KC inventory and content repository.

```

#Step 1: Read the integer

text = input("Enter an integer: ")

num = int(text)

#Step 2: Determine whether the integer is even or odd

if num % 2 == 0 :

    print("The integer is even.")

else :

    print("The integer is odd.")

```

(a) An example challenge code.

```

#Step 1: Read the integer

text = input("Enter an integer: ")

num = int(text)

#Step 2: Determine whether the integer is positive, negative, or zero

if num > 0 :

    print("The integer is positive.")

elif num < 0 :

    print("The integer is negative.")

else :

    print("The integer is zero.")

```

(b) Recommended worked example.

Figure 3: KC-level similarity within the *If-Else* topic from the same bundle. Left: challenge code that determines whether an input integer is *even or odd*; right: recommended worked example that determines whether an input integer is *positive, negative, or zero*. Colored rails in the left margin mark the top-contributing pattern-based KCs; color intensity is proportional to the attention weight.

6. Discussion

This work addresses a challenge in programming education: recommending learning resources at scale that are contextually relevant to a student’s current learning focus [32]. Our approach tags learning resources with pattern-based KCs that capture instructionally important programming patterns in students’ code, enabling more targeted recommendations. By aligning retrieval with pedagogical units such as KCs, the recommended resources are likely to address the skills students need to practice.

To assess whether the pattern-based KC vectors capture educationally meaningful structure, we conducted a tightness and clustering analysis. The tightness results show clear alignment with the expert-bundled content: learning activities within the same instructor-defined bundle form substantially tighter neighborhoods than those grouped only at the topic level, while topic-level groupings remain more compact than the dataset as a whole. This ordering is consistent with instructional intent, since bundles are designed to capture near-isomorphic problem families, suggesting that the representation encodes fine-grained, instructionally relevant similarity.

We then used hierarchical clustering as an unsupervised validation of the representation space. This analysis examined whether learning activities naturally organize into coherent groups based only on their knowledge vectors, without access to expert-defined bundle labels. In contrast to retrieval-based evaluation, which depends on ranked neighbor lists and top-N cutoffs, clustering tests whether programs that require similar KCs are placed near one another. The strong alignment between induced clusters and expert bundles indicates that pattern-based KC vectors capture conceptual similarity within the programs. This recovery of bundle-level structure suggests that the approach may generalize to new or unlabeled repositories. Such clustering could support instructors by proposing candidate bundles, auditing existing groupings, or surfacing related activities that share underlying patterns across topics.

Experimental results further showed that the proposed framework achieves strong retrieval performance on the PCEX dataset when evaluated against expert-annotated bundles as ground truth. Using KC-level similarity, the method performs well under rank-based metrics, including Top-5, MRR, and mAP, and it outperforms concept vectors [11], code2vec [15], and SANN [16] embedding, and LLM-based KC vectors [17, 18]. Compared with other embedding approaches, pattern-based KCs better capture semantic similarity by reducing reliance on surface-level information through normalization and improving generalization through VAE training. LLM-based KC identification approaches can map similar implementations of programming concepts, but may suffer from sparsity and limited explainability. LLM-KCI showed competitive performance when supplied with an instructor-defined KC set, but constructing such inventories requires substantial expert effort and may reflect an instructor’s

subjective view of the domain. Pattern-based KCs are learned directly from student code, allowing the representation to adapt its granularity based on where students struggle and which patterns require mastery. Ontology-based KC approaches exhibit lower MRR and mAP, suggesting that predefined conceptual structures are less effective at capturing fine-grained patterns in student solutions.

We also demonstrated the explainability of our model, an important characteristic to increase the trust of instructors and educators alike [33]. As each pattern-based KC is anchored to AST spans, we can project top-contributing KCs back onto programs for both the targeted item and the recommendation, producing line-level rationales. These explanations help students focus on the specific constructs they need to review. They also let instructors audit the basis for a match, diagnose false positives, and refine the resource inventory over time. More broadly, this supports actionable analytics and interfaces, such as open learner model views [34] that communicate progress and gaps at the KC level.

Overall, the proposed pipeline offers a scalable alternative to manually curated links, which become costly as repositories grow [2]. By discovering and reusing structural KCs across items and topics, the system can generalize to new materials with limited expert effort, supporting maintenance and expansion of content libraries. This helps bridge the gap between large repositories and personalized support, complementing prior work that showed the value of bundling learning content [2].

7. Limitations and Future Work

Although the proposed KC-based recommendation method shows strong performance, there are multiple avenues for improving the work in the future. First, the dataset is small (123 programs), which may constrain both pattern-based KC discovery and evaluation sensitivity. Future work should expand the dataset across additional course offerings, textbooks, and activity types, then measure how recommendation quality changes as the number and diversity of learning activities increase. Second, explanation quality depends on the attention weights assigned by the framework and the KC clustering resolution. We highlight KC patterns that are the primary drivers for a recommendation; however, highly overlapping KCs can produce adjacent or redundant highlights. Future work will (i) calibrate attention thresholds and clustering granularity, (ii) merge near-duplicate KCs, and (iii) validate explanation fidelity through instructor judgments. Third, the first step of the KC extraction pipeline uses topic labels for training, limiting direct application to unlabeled corpora. Although many instructional resources are organized by topic, future work should explore weakly supervised or self-supervised alternatives and examine performance when labels are incomplete or noisy. Finally, future classroom studies should measure impacts on problem-solving performance, time-to-solution, and transfer when recommendations and explanations are available.

8. Conclusion

This paper presented a recommendation framework for suggesting relevant programming learning resources by matching code at the level of pattern-based knowledge components. Similarity analysis showed that the resulting knowledge vectors align with expert-defined topics and bundles, indicating that they capture fine-grained programming patterns reflected in instructional content organization. Retrieval results showed that the pattern-based KC approach produces bundle-aligned recommendations and outperforms representative KC- and embedding-based baselines. The framework also provides line-level rationales by highlighting the code regions responsible for a match, supporting transparency for students and instructors. Overall, the method provides a scalable alternative to manual curation of relevant programming activities by aligning recommendations with the underlying skills a program requires and, thus, supporting targeted remediation.

Acknowledgments

This research was supported by the National Science Foundation (NSF) under Grants #2418658, #2418655, and #2418657. Any opinions, findings, and conclusions expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT in order to: Paraphrase and reword. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

- [1] A. Renkl, Toward an instructionally oriented theory of example-based learning, *Cognitive Science* 38 (2014) 1–37.
- [2] R. Hosseini, K. Akhuseyinoglu, P. Brusilovsky, L. Malmi, K. Pollari-Malmi, C. Schunn, T. Sirkiä, Improving engagement in program construction examples for learning python programming, *International Journal of Artificial Intelligence in Education* 30 (2020) 299–336.
- [3] G. Pitts, M. Hoq, P. Brusilovsky, N. Norouzi, A. Hellas, J. Leinonen, B. Akram, Personalized worked example generation from student code submissions using pattern-based knowledge components, *arXiv preprint arXiv:2604.24758* (2026).
- [4] A. S. Najar, A. Mitrovic, B. M. McLaren, Learning with intelligent tutors and worked examples: selecting learning activities adaptively leads to better learning outcomes than a fixed curriculum, *User Modeling and User-Adapted Interaction* 26 (2016) 459–491. URL: <https://doi.org/10.1007/s11257-016-9181-y>.
- [5] D. Parsons, P. Haden, Parson's programming puzzles: a fun and effective learning tool for first programming courses, in: *Proceedings of the 8th Australasian Conference on Computing Education*, volume 52, Australian Computer Society, 2006, pp. 157–163.
- [6] K. Akhuseyinoglu, A. Klačnja-Milicevic, P. Brusilovsky, The impact of connecting worked examples and completion problems for introductory programming practice, in: *Proceedings of the European Conference on Technology Enhanced Learning (EC-TEL 2024)*, Springer International Publishing, 2024, pp. 3–18.
- [7] J. Barria-Pineda, K. Akhuseyinoglu, P. Brusilovsky, Explaining need-based educational recommendations using interactive open learner models, in: *Proceedings of the International Workshop on Transparent Personalization Methods based on Heterogeneous Personal Data, ExHUM at the 27th ACM Conference On User Modelling, Adaptation And Personalization, UMAP '19*, 2019, p. 273–277. URL: <https://dl.acm.org/citation.cfm?id=3323463>. doi:10.1145/3314183.3323463.
- [8] A. S. Najar, T. Mitrovic, B. M. McLaren, Adaptive support versus alternating worked examples and tutored problems: Which leads to better learning?, in: V. Dimitrova, T. Kuflik, D. Chin, F. Ricci, P. Dolog, G.-J. Houben (Eds.), *Proceedings of the 22nd Conference on User Modeling, Adaptation and Personalization (UMAP 2014)*, Lecture Notes in Computer Science, Springer Verlag, 2014, pp. 171–182.
- [9] M. Kibby, J. Mayes, Towards intelligent hypertext, *Hypertext: Theory into Practice* (1989) 164–172.
- [10] J. T. Mayes, M. R. Kibby, H. Watson, Strathtutor®: The development and evaluation of a learning-by-browsing system on the macintosh, *Computers & Education* 12 (1988) 221–229.
- [11] R. Hosseini, P. Brusilovsky, A study of concept-based similarity approaches for recommending program examples, *New Review of Hypermedia and Multimedia* 23 (2017) 161–188.
- [12] L. Carr, W. Hall, S. Bechhofer, C. Goble, Conceptual linking: ontology-based open hypermedia, in: *Proceedings of the 10th International Conference on World Wide Web*, 2001, pp. 334–342.
- [13] K. R. Koedinger, A. T. Corbett, C. Perfetti, The knowledge-learning-instruction framework:

Bridging the science-practice chasm to enhance robust student learning, *Cognitive Science* 36 (2012) 757–798.

- [14] M. Hoq, G. Pitts, T. Bhatt, A. Pandya, A. Lan, P. Brusilovsky, B. Akram, Pattern-based knowledge component extraction from student code using representation learning, *arXiv preprint arXiv:2508.09281* (2025).
- [15] U. Alon, M. Zilberstein, O. Levy, E. Yahav, code2vec: Learning distributed representations of code, *Proceedings of the ACM on Programming Languages* 3 (2019) 1–29.
- [16] M. Hoq, A. Patil, K. Akhuseyinoglu, P. Brusilovsky, B. Akram, An automated approach to recommending relevant worked examples for programming problems, in: *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, 2025, pp. 527–533.
- [17] Z. Duan, N. Fernandez, A. B. L. Narayanan, M. Hassany, R. S. de Alencar, P. Brusilovsky, B. Akram, A. Lan, Automated knowledge component generation for interpretable knowledge tracing in coding problems, 2025. URL: <https://arxiv.org/abs/2502.18632>. *arXiv:2502.18632*.
- [18] R. Niousha, A. O’Neill, E. Chen, V. Malhotra, B. Akram, N. Norouzi, Llm-kci: Leveraging large language models to identify programming knowledge components, in: *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 2*, 2025, pp. 1557–1558.
- [19] J. Fan, T. Mihaylova, B. Akram, N. Norouzi, P. Brusilovsky, A. Hellas, J. Leinonen, Adaptive learning curve analytics with llm-kc identifiers for knowledge component refinement, in: *Proceedings of the 2025 Conference on UK and Ireland Computing Education Research*, 2025, pp. 1–7.
- [20] D. Tudhope, C. Taylor, Navigation via similarity: automatic linking based on semantic closeness, *Information Processing & Management* 33 (1997) 233–242.
- [21] M. Crampes, S. Ranwez, Ontology-supported and ontology-driven conceptual navigation on the world wide web, in: *Proceedings of the 11th ACM on Hypertext and Hypermedia*, 2000, pp. 191–199.
- [22] P. Dolog, N. Henze, W. Nejdl, Logic-based open hypermedia for the semantic web, in: *Proceedings of the International Workshop on Hypermedia and the Semantic Web, Hypertext 2003 Conference*, 2003.
- [23] S. Gross, B. Mokbel, B. Hammer, N. Pinkwart, How to select an example? a comparison of selection strategies in example-based learning, in: *Proceedings of the Intelligent Tutoring Systems: 12th International Conference, ITS 2014*, Springer, 2014, pp. 340–347.
- [24] G. Weber, A. Mollenberg, Elm-pe: A knowledge-based programming environment for learning lisp, in: *Proceedings of ED-MEDIA*, 1994, p. 557–562.
- [25] R. Hosseini, P. Brusilovsky, Example-based problem solving support using concept analysis of programming content, in: *Proceedings of the Intelligent Tutoring Systems: 12th International Conference, ITS 2014*, Springer, 2014, pp. 683–685.
- [26] J. Barria-Pineda, A. B. L. Narayanan, P. Brusilovsky, Augmenting digital textbooks with reusable smart learning content: solutions and challenges., in: *iTextbooks 2022: Proceedings of the Fourth International Workshop on Intelligent Textbooks 2022 co-located with 23d International Conference on Artificial Intelligence in Education (AIED 2022)*, 2022, pp. 77–91.
- [27] A. J. Sabet, I. Alpizar-Chacon, J. Barria-Pineda, P. Brusilovsky, S. Sosnovsky, Enriching intelligent textbooks with interactivity: When smart content allocation goes wrong, in: *iTextbooks 2022: Proceedings of the Fourth International Workshop on Intelligent Textbooks 2022 co-located with 23d International Conference on Artificial Intelligence in Education (AIED 2022)*, 2022, pp. 1–13.
- [28] M. Hoq, S. R. Chilla, M. Ahmadi Ranjbar, P. Brusilovsky, B. Akram, SANN: Programming code representation using attention neural network with optimized subtree extraction, in: *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, 2023, pp. 783–792.
- [29] M. Hoq, Y. Shi, J. Leinonen, D. Babalola, C. Lynch, T. Price, B. Akram, Detecting chatgpt-generated code submissions in a cs1 course using machine learning models, in: *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, 2024, pp. 526–532.
- [30] Y. Shi, R. Schmucker, M. Chi, T. Barnes, T. Price, Kc-finder: automated knowledge component discovery for programming problems., in: *Proceedings of the International Conference on*

Educational Data Mining (EDM), EDM, 2023, pp. 28–39.

- [31] M. Hoq, A. Rao, R. Jaishankar, K. Piryani, N. Janapati, J. Vandenberg, B. Mott, N. Norouzi, J. Lester, B. Akram, Automated identification of logical errors in programs: Advancing scalable analysis of student misconceptions, in: Proceedings of the 18th International Conference on Educational Data Mining (EDM), 2025, pp. 90–103.
- [32] P. Brusilovsky, C. Peylo, Adaptive and intelligent web-based educational systems, International Journal of Artificial Intelligence in Education 13 (2003) 156–169.
- [33] G. Pitts, S. Motamedi, Understanding human-ai trust in education, Telematics and Informatics Reports (2025) 100270.
- [34] J. Barria-Pineda, J. Guerra, Y. Huang, P. Brusilovsky, Concept-level knowledge visualization for supporting self-regulated learning, in: Companion Proceedings of the 22nd International Conference on Intelligent User Interfaces, 2017, pp. 141–144.