

Speed–Accuracy Trade-offs in Knowledge Component Identification from Student Code

Jing Fan¹, Tsvetomila Mihaylova¹, Peter Brusilovsky², Juho Leinonen¹,
Charles Koutchme¹, Narges Norouzi³, Bitu Akram⁴, Arto Hellas¹

¹Aalto University, Finland

²University of Pittsburgh, USA

³University of California, Berkeley, USA

⁴North Carolina State University, USA

jing.fan@aalto.fi

Abstract

Identifying knowledge components (KCs) from student-written code supports fine-grained feedback and learner modeling in programming education, but the comparative effectiveness of traditional machine learning (ML) and large language models (LLMs) remains unclear. We compare feature-based ML models with instruction-tuned Gemma 3 models (1B–27B) on 1,800 GPT-4o-annotated Dart submissions under a unified protocol. We also evaluate per-KC and joint multi-KC prompting to examine accuracy–efficiency trade-offs. On a fixed 450-submission held-out set, token- and character-based ML models are more accurate and several orders of magnitude faster than the strongest per-KC LLM. Joint prompting suffers from output-format failures and does not provide a favorable accuracy–efficiency trade-off. These findings position traditional ML as a strong deployment-oriented baseline within the evaluated exercise distribution, while suggesting complementary roles for LLMs in offline or semantic-analysis workflows.

of how different code representations support fine-grained, multi-label KC identification from student-written code.

Recent advances in large language models (LLMs) further expand the design space for code analysis. Although LLMs perform well on many programming-related tasks, their effectiveness for fine-grained KC identification remains underexplored, particularly with respect to model scale and the trade-offs between predictive performance and inference efficiency relative to traditional feature-based approaches.

In this paper, we present a systematic empirical study of KC identification from student-submitted code in an introductory Dart programming course. Using a dataset of 1,800 annotated submissions, we investigate (1) the effects of code feature representations and traditional machine learning models on KC identification performance, and (2) the speed–accuracy trade-offs between traditional ML and LLM-based approaches, including per-KC and joint multi-KC prompting.

1 Introduction

Knowledge components (KCs) are fine-grained skills or concepts that learners are expected to acquire and form a foundation for intelligent tutoring systems and learning analytics. Accurate KC identification supports applications such as student modeling, adaptive feedback, and curriculum design, yet KCs are traditionally associated with tasks or problems rather than with learners’ actual solutions, limiting their ability to reflect correct knowledge application.

In programming education, student-submitted code provides direct evidence of knowledge use. Prior work has examined automated code analysis using test cases, static analysis, and program representations such as abstract syntax trees (ASTs), but there has been limited systematic comparison

2 Related Work

Knowledge Components (KCs) are discrete concepts or skills that learners are expected to acquire in a course (Koedinger et al., 2012). They are central to intelligent tutoring systems and knowledge tracing, where KCs are used to model and monitor students’ evolving understanding over time (Rivers et al., 2016; Ai et al., 2019; Abdelrahman et al., 2023). In programming education, KCs may correspond to concrete language features (e.g., *for loop*) or higher-level actions that coordinate multiple concepts (e.g., *implementing a loop with a sentinel condition*). However, manually assigning KCs to programming assignments is burdensome and requires substantial instructor effort (Alevan and Koedinger, 2013).

Prior work has explored automated KC extrac-

tion from code submissions using primarily code-derived representations. Early work proposed ontology-driven KC indexing from abstract syntax trees (ASTs) (Hosseini and Brusilovsky, 2013). More recent studies include data-driven methods that discover KCs from student submissions (Shi et al., 2023), AST-based analyses that treat language constructs as candidate KCs and study their learning behavior (Demirtas et al., 2024), construct-based representations derived from code structure to characterize skills required by exercises (Pan et al., 2025), and representation-learning approaches that identify recurring code patterns and group them into latent KCs (Hoq et al., 2025). These approaches typically rely on the submission code as the primary input.

Recent studies further show that large language models (LLMs) can be applied to KC identification from programming tasks (Shi et al., 2024; Niousha et al., 2024; Fan et al., 2025; O’Neill et al., 2025), and LLM-derived KCs may be comparable to instructor-defined KCs in quality (Fan et al., 2025). Across the broader KC extraction literature, methods also differ in how guidance and domain knowledge are incorporated. Some approaches are unguided, relying directly on AST nodes without model training or external supervision (Demirtas et al., 2024; Pan et al., 2025). Others are ML-guided, inducing KCs from data via learned code representations (Shi et al., 2023; Hoq et al., 2025). A third line uses ontology-guided extraction to improve indexing consistency (Hosseini and Brusilovsky, 2013; Hendrawan et al., 2026; Sampaio de Alencar et al., 2025). Building on these lines of work, we extract KCs at the submission level and empirically compare traditional ML models and open-source LLMs using GPT-4o-based reference annotations.

3 Methods

3.1 Dataset Construction and Preprocessing

The dataset used in this study was collected from an introductory Dart programming course comprising 45 programming exercises. Each exercise was accompanied by an instructor-defined list of knowledge components (KCs) describing the intended programming concepts and skills. We deduplicated the instructor-provided KC lists across all exercises, resulting in a unified instructor-defined taxonomy of 22 KCs. Table 1 illustrates the dataset structure, including solution code for an example exer-

cise, instructor-defined KCs, and the corresponding GPT-4o reference annotations.

Student code submissions associated with these exercises were collected and processed using a standardized pipeline. First, duplicate submissions were removed to avoid bias from repeated solutions. Second, all remaining submissions were parsed into abstract syntax trees (ASTs), and only submissions that could be successfully parsed were retained. Third, submissions were labeled as correct or incorrect based on the course grading system. After duplicate removal and AST parsing, we randomly sampled 20 correct and 20 incorrect submissions from each of the 45 exercises. This yielded 900 correct and 900 incorrect submissions, for a total of 1,800 unique student solutions. The study received ethics approval from the Aalto university.

3.2 Knowledge Component Annotation

We annotated each student submission with respect to the instructor-defined KC taxonomy to determine whether each knowledge component was correctly applied in the code. For annotation, each submission was evaluated independently for each KC using a fixed prompt template and a binary decision format.

As illustrated in Figure 1, the model was provided with the full student code and a single target knowledge component. For each KC, the model produced a binary label, where 1 indicates that the code clearly and correctly applies the KC, and 0 indicates that the KC is not applied or is applied incorrectly or ineffectively. This process yielded a multi-label binary annotation vector for each submission spanning 22 distinct KCs. A consistent KC-by-KC task definition was used across GPT-4o and the Gemma 3 models, while model-specific output formats were applied. GPT-4o annotations were obtained through the Aalto university OpenAI gateway using a chat-completion interface with a fixed system prompt and task-specific user prompt, temperature set to 0, and JSON-only output instructions.

The resulting annotated dataset serves two roles in our experimental design: (1) as training and test data for the repeated-holdout evaluation of traditional machine learning models in Experiment 1, and (2) as training and fixed held-out test data for the controlled comparison in Experiment 2, with the LLM configurations evaluated on the same test split.

Table 1: Example dataset format.

Ex.	Solution code	Instructor-defined KCs	Reference KC annotations
1	<pre>print('Write something!'); var something = stdin.readLineSync(); print('You wrote \$something!');</pre>	bug detection	bug detection; standard input; string interpolation; ...
2	<pre>print('What is your name?'); var name = stdin.readLineSync(); print('Hi \$name!');</pre>	standard input; string interpolation	
...	...	...	

System prompt

You are a strict Dart code evaluator.
You MUST output ONLY one character: 0 or 1.
No extra text, no spaces, no newline.
If unsure, output 0.

User prompt

Scoring rule:
1 = the code clearly and correctly applies the KC
0 = the KC is not applied, or is applied incorrectly/ineffectively

KC:
{kc}

<DART_CODE>
{code}
</DART_CODE>

Output (single char 0 or 1):

Figure 1: Example per-KC prompt and output format used for the Gemma 3 models.

3.3 Annotation Validation

To assess the reliability of GPT-4o-based annotations, we compared them with manual human re-annotations on a subset of 450 submissions stratified by exercise and submission correctness. Each submission was re-labeled by a member of the research team familiar with the course context under a strict protocol in which KCs were judged solely from the code, with uncertain cases assigned 0. We then compared the original and re-annotated labels at the KC level. The overall agreement was **87.3%**, indicating strong consistency. Agreement was higher for syntactic KCs (e.g., loops, input/output) and lower for more semantic ones (e.g., arithmetic or conditional logic). This esti-

Table 2: Evaluation metrics used for multi-label KC identification.

Metric	Definition	Notes
Micro Precision	$\frac{\sum_k TP_k}{\sum_k (TP_k + FP_k)}$	Aggregated over all KCs.
Micro Recall	$\frac{\sum_k TP_k}{\sum_k (TP_k + FN_k)}$	Sensitive to missed KC detections.
Micro-F1	$2 \cdot \frac{P_{\text{micro}} R_{\text{micro}}}{P_{\text{micro}} + R_{\text{micro}}}$	Primary overall performance metric.
Macro-F1	$\frac{1}{K} \sum_{k=1}^K F1_k$	Gives equal weight to each KC.
Inference Time	Per submission (seconds)	Wall-clock prediction time.

mate is conservative due to the strict “uncertainty → 0” rule, but supports the reliability of GPT-4o annotations as a proxy for human labeling.

3.4 Evaluation Metrics

We used micro-F1 as the primary performance metric and report macro-F1 and per-submission inference time as secondary metrics. Table 2 summarizes the metrics.

3.5 Experiment 1: Traditional Machine Learning with Feature Engineering

We evaluated three families of code representations for multi-label KC classification: lexical, structural, and hybrid features. Lexical features were represented as sparse TF-IDF vectors over token unigrams, token n -grams ($n = 1-3$), and character n -grams ($n = 3-5$). Structural features were derived from Tree-sitter-parsed ASTs, including AST node sequences, SVD-reduced node-type frequency embeddings, selected construct counts, and basic tree statistics such as node count, depth, and branching factor. Hybrid features concatenated lex-

ical TF-IDF features with AST-derived statistics and are referred to as *tok+stats* in the results.

We compared several standard multi-label classifiers, including One-vs-Rest Logistic Regression, Linear SVM, SGD classifiers, and multi-output Random Forests and Extra Trees. For each feature-model combination, we used a repeated holdout protocol with a 75%/25% train-test split repeated across five random seeds (42, 43, 44, 45, 46); reported metrics were averaged across repetitions. Class imbalance was handled with class-weighted losses where supported. All ML models were implemented in `scikit-learn` using default hyperparameters, without systematic tuning or cross-validated model selection.

3.6 Experiment 2: Controlled Comparison Between ML and LLM Approaches

We conducted a controlled comparison between traditional ML and LLM-based approaches using a fixed submission-level 75%/25% split (random seed 42), yielding 1,350 training submissions and 450 test submissions. Traditional ML models were trained on the training set, while the ML and LLM configurations were evaluated on the same 450-submission test set against the same GPT-4o reference labels. We evaluated instruction-tuned Gemma 3 models with 1B, 4B, 12B, and 27B parameters in a zero-shot setting, without fine-tuning or in-context examples. All LLM inference experiments were conducted on a single NVIDIA A100 GPU with 80 GB of memory using offline inference with Hugging Face Transformers. For per-KC prompting, decoding was constrained to the tokens 0, 1. In addition to predictive performance, we measured per-submission inference time to analyze speed-accuracy trade-offs. For LLMs, we compared two prompting strategies: per-KC prompting, which aggregates 22 independent binary predictions, and joint prompting, which predicts all KCs as a fixed-length binary string in a single generation. Valid bit strings were mapped back to KC vectors. Joint-prompting F1 scores were computed only on successfully parsed outputs, while parsing failures were reported separately. For ML models, inference time was measured by applying the fitted feature-classifier pipeline to the complete test set over five runs and dividing the mean wall-clock time by the number of submissions. This measurement includes feature transformation and classifier prediction, but excludes model fitting and data loading. For LLMs, inference time was measured

from prompt tokenization to generation completion, excluding model loading, prompt construction, and downstream parsing. In the per-KC setting, submission-level inference time corresponds to the cumulative time over all KC-specific generations; in the joint setting, it corresponds to a single generation per submission.

4 Results

4.1 RQ1 (Experiment 1): Traditional ML for KC Prediction

We investigate the effects of feature representations and traditional machine learning (ML) models on multi-label KC prediction. Figure 2 summarizes the results using a unified feature-model heatmap, reporting micro-F1 and macro-F1 scores averaged over repeated train-test splits.

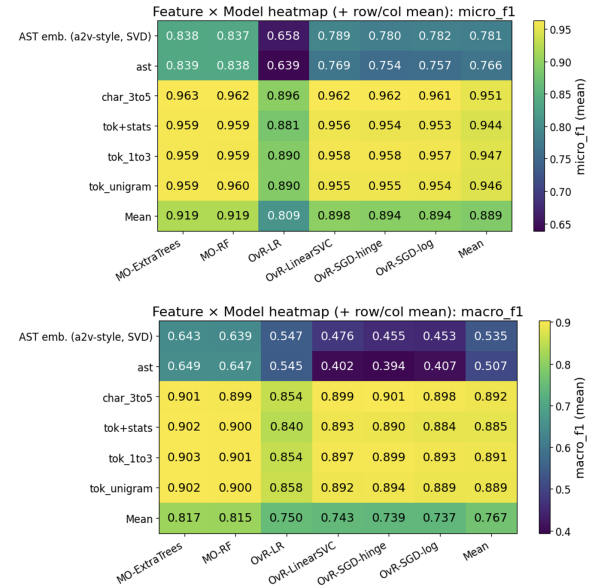


Figure 2: Feature-model heatmap of micro-F1 and macro-F1 for multi-label KC prediction.

Impact of feature representations. Across all classifiers, feature representation emerges as the dominant factor influencing performance. Lexical features consistently outperform structure-based representations. Character n -grams (3–5) and token-based features (token unigrams and token n -grams 1–3) achieve the strongest results, with micro-F1 scores above 0.94 and macro-F1 scores around 0.88–0.90. Augmenting token representations with lightweight statistical features (tok+stats), including code length, number of lines, and basic AST tree-shape statistics (total node count and maximum depth), yields comparable

performance but does not provide consistent improvements. In contrast, AST-based representations perform substantially worse. Both the raw AST node sequence and the reduced-dimensional AST embedding (ast2vec-style structural embedding with SVD) exhibit markedly lower micro-F1 (approximately 0.77–0.78) and macro-F1 (approximately 0.51–0.54). The performance degradation is more pronounced for macro-F1, indicating that AST features struggle to capture less frequent or fine-grained knowledge components.

Impact of classifier choice. Classifier choice has a secondary but observable effect on performance. Multi-output ensemble models, including MO-ExtraTrees and MO-RF, achieve the highest overall scores, particularly when paired with token- and character-based lexical representations. Linear one-vs-rest (OvR) classifiers, such as LinearSVC and SGD-based variants, perform competitively, whereas OvR logistic regression consistently underperforms across feature types. The performance variation attributable to classifier choice is smaller than that induced by feature representation, reinforcing the central role of feature engineering in KC prediction.

Feature–model interaction. The feature–model heatmap further reveals clear interaction effects. Lexical representations demonstrate robust performance across all classifiers, maintaining high scores regardless of model choice. In contrast, AST-based features are highly sensitive to the classifier: ensemble models partially mitigate performance loss, whereas linear OvR classifiers perform poorly when combined with AST representations.

The strongest feature–model combinations involve token- and character-based representations paired with ensemble models or linear SVM/SGD classifiers, achieving micro-F1 values above 0.95 and macro-F1 values around 0.90.

4.2 RQ2 (Experiment 2): Speed–Accuracy Trade-offs Between ML and LLM Approaches

To address RQ2, we analyze the trade-off between predictive performance and inference efficiency across all evaluated methods, including both traditional machine learning (ML) models and large language models (LLMs). Figure 3 visualizes this trade-off by plotting micro-F1 scores on the fixed 450-submission held-out test set against per-submission inference time on a logarithmic scale.

The results show a clear separation between ML-

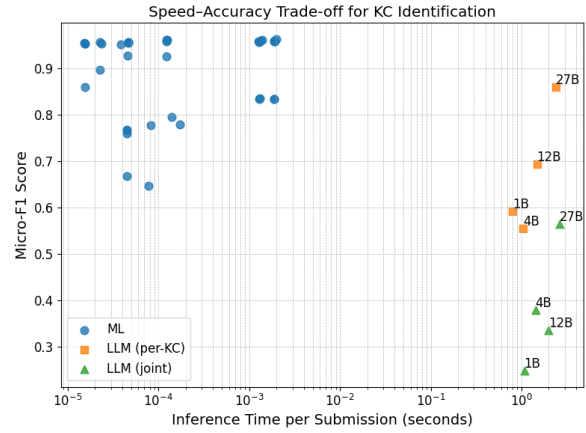


Figure 3: Speed–accuracy trade-off on the fixed 450-submission test set. ML and per-KC LLM scores are computed over the complete test set. Joint-prompting scores are conditional micro-F1 values computed on successfully parsed outputs; parsing failure rates are reported in the text.

based and LLM-based approaches. Traditional ML models consistently occupy the upper-left region of the plot, indicating that they achieve stronger predictive performance at substantially lower computational cost. In particular, the best-performing ML configurations reach micro-F1 scores around 0.96 while maintaining inference times below 0.002 seconds per submission. For example, the char_3to5 + MO-ExtraTrees configuration achieves the highest micro-F1 (0.962) with an inference time of approximately 1.98×10^{-3} seconds, while faster linear models such as tok_unigram + OvR-LinearSVC still achieve competitive performance (0.955) at only 1.5×10^{-5} seconds per submission. These results indicate that traditional ML methods provide highly favorable speed–accuracy trade-offs for fine-grained KC identification.

For LLM-based approaches, we compare two prompting strategies: (1) a per-KC prompting setup, where each knowledge component is evaluated independently, and (2) a joint multi-KC prompting setup, where all KCs are predicted simultaneously in a single prompt. Under the per-KC prompting setup, performance improves with model scale, but inference cost also increases. Specifically, micro-F1 rises from 0.591 for Gemma 3 1B to 0.859 for Gemma 3 27B, while per-submission inference time increases from approximately 0.80 to 2.39 seconds. On this controlled test split, the strongest per-KC LLM remains substantially slower and less accurate than the strongest ML baseline.

Notably, the joint multi-KC prompting strategy

does not provide the expected efficiency advantage. Among successfully parsed outputs, joint prompting achieves conditional micro-F1 scores of 0.249, 0.379, 0.335, and 0.564 for the 1B, 4B, 12B, and 27B models, respectively. At the same time, inference time is not reduced; instead, joint prompting requires approximately 1.07 to 2.63 seconds per submission, which is comparable to or slightly slower than the per-KC setting. This suggests that generating the full KC vector in a single step introduces additional difficulties, such as structured output control and simultaneous multi-label reasoning, which offset the theoretical efficiency benefits of joint prediction. This interpretation is further supported by the parsing failure rates observed in the joint setting. The proportion of unparseable outputs was 86.9% for Gemma 3 1B, 32.0% for Gemma 3 4B, 26.9% for Gemma 3 12B, and 1.6% for Gemma 3 27B. Because the reported joint-prompting F1 scores exclude unparseable outputs, they represent conditional performance given successful parsing rather than end-to-end performance over the full test set. The high failure rates nevertheless demonstrate limited output-format robustness, particularly for smaller models, with robustness improving markedly at larger scales.

5 Discussion

This study systematically compares traditional feature-based machine learning and instruction-tuned large language models for knowledge component (KC) identification from student-written code within a unified study design. Across the two research questions, the results highlight the importance of representation design and the practical relevance of speed–accuracy trade-offs for deployment in educational systems.

RQ1: Representation design outweighs model choice in traditional ML. For feature-based KC identification, representation choice is more important than classifier choice. Token- and character-based lexical features are better aligned with fine-grained multi-label KC prediction than the AST-based representations considered here, while differences among classifiers become secondary once suitable representations are used.

RQ2: Speed–accuracy trade-offs favor traditional ML. In the fixed 450-submission held-out comparison, traditional ML models provide a clearly better balance between predictive per-

formance and inference efficiency. They achieve higher micro-F1 with microsecond- to millisecond-level latency, whereas the LLM-based approaches require seconds per submission. This gap is especially important for high-throughput or near-real-time educational settings, where latency directly constrains the feasibility of iterative, per-submission feedback. Joint multi-KC prompting also exhibits substantial structured-output failures. Because its reported F1 scores are computed only on successfully parsed outputs, they should be interpreted together with the reported parsing failure rates.

Implications for scalable educational systems.

Taken together, these findings position feature-based ML models as a practical and scalable solution for fine-grained KC identification within the evaluated exercise distribution. LLMs are better viewed as complementary tools for offline analysis, richer semantic interpretation, or hybrid pipelines in which they support KC refinement while ML models handle routine inference. More broadly, KC identification methods should be evaluated not only by predictive accuracy, but also by inference efficiency, computational cost, and deployability.

6 Limitations and Future Work

This study has several limitations that also point to promising directions for future research. First, the dataset is drawn from a single introductory Dart programming course and a single instructor-provided KC taxonomy. KC structure and granularity may differ across languages, course levels, and pedagogical contexts. Evaluating additional courses, languages, and KC definitions would help assess the generalizability of the observed trends. Second, traditional machine learning models were evaluated using default hyperparameter settings without systematic tuning, which may underestimate their best achievable performance. Third, the train/test split was performed at the submission level rather than by exercise, so some exercise-specific patterns may have been shared across training and test data; the reported results therefore reflect generalization to held-out submissions within the same exercise distribution rather than to entirely unseen exercises. Fourth, the structural code representations considered here rely on relatively simple AST linearizations, such as node sequences and low-dimensional embeddings derived from linear co-occurrence features and SVD. More expres-

sive structural representations, such as path-based code representations or graph neural networks, may yield different accuracy–efficiency trade-offs and warrant further investigation.

7 Conclusions

This study compares traditional ML and LLM-based approaches for KC identification from student-written code. In the fixed held-out comparison, token- and character-based ML models provide the most favorable speed–accuracy trade-off, achieving higher predictive performance with substantially lower inference time than the evaluated LLMs. Larger Gemma 3 models perform better than smaller models under per-KC prompting, but remain slower and less accurate than the strongest ML configurations. Output-format failures under joint prompting further reduce the practicality of LLM-based approaches for high-throughput settings. Overall, traditional ML remains a strong deployment-oriented baseline within the evaluated exercise distribution, while LLMs may play complementary roles in offline or semantic-analysis workflows.

Acknowledgments

This work was supported by Research Council of Finland grants #367787 and #356114, and by the National Science Foundation grants #2426837, #2426838, and #2426839. Generative AI tools were used for language polishing.

References

- Ghodai Abdelrahman, Qing Wang, and Bernardo Nunes. 2023. Knowledge tracing: A survey. *ACM Computing Surveys*, 55(11):1–37.
- Fangzhe Ai, Yishuai Chen, Yuchun Guo, Yongxiang Zhao, Zhenzhu Wang, Guowei Fu, and Guangyan Wang. 2019. Concept-aware deep knowledge tracing and exercise recommendation in an online learning system. *International Educational Data Mining Society*.
- Vincent Aleven and Kenneth R Koedinger. 2013. Knowledge component (kc) approaches to learner modeling. *Design recommendations for intelligent tutoring systems*, 1:165–182.
- Mehmet Arif Demirtas, Max Fowler, and Kathryn Cunningham. 2024. Reexamining learning curve analysis in programming education: The value of many small problems. In *17th International Conference on Educational Data Mining (EDM 2024)*.
- Jing Fan, Tsvetomila Mihaylova, Bitu Akram, Narges Norouzi, Peter Brusilovsky, Arto Hellas, and Juho Leinonen. 2025. 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*, New York, NY, USA. ACM.
- Rully Agus Hendrawan, Rafaella Sampaio de Alencar, Alice Micheli, Peter Brusilovsky, Jordan Barria-Pineda, and Sergey Sosnovsky. 2026. Data-driven evaluation of llm-based ontology concept extraction from programming learning content. In *the 16th International Learning Analytics and Knowledge Conference*. ACM.
- Muntasir Hoq, Griffin Pitts, Andrew Lan, Peter Brusilovsky, and Bitu Akram. 2025. Pattern-based knowledge component extraction from student code using representation learning. *arXiv preprint arXiv:2508.09281*.
- Roya Hosseini and Peter Brusilovsky. 2013. Javaparser: A fine-grain concept indexing tool for java problems. In *The First Workshop on AI-supported Education for Computer Science (AIEDCS) at the 16th Annual Conference on Artificial Intelligence in Education, AIED 2013*, pages 60–63.
- Kenneth R Koedinger, Albert T Corbett, and Charles Perfetti. 2012. The knowledge-learning-instruction framework: Bridging the science-practice chasm to enhance robust student learning. *Cognitive science*, 36(5):757–798.
- Rose Niousha, Muntasir Hoq, Bitu Akram, and Narges Norouzi. 2024. Use of large language models for extracting knowledge components in cs1 programming exercises. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2, SIGCSE 2024*, page 1762–1763.
- Abby O’Neill, Samantha Smith, Aneesh Durai, John DeNero, J.D. Zamfirescu-Pereira, and Narges Norouzi. 2025. From code to concepts: Textbook-driven knowledge tracing with llms in cs1. In *the 56th ACM Technical Symposium on Computer Science Education V. 2*, pages 1565–1566. ACM.
- Ka Weng Pan, Bryn Jeffries, and Irena Koprinska. 2025. Grouping and differentiating programming exercises using minimal programming constructs. In *9th Educational Data Mining in Computer Science Education (CSEDM) Workshop at EDM2025*, volume 4019 of *CEUR Workshop Proceedings*.
- Kelly Rivers, Erik Harpstead, and Kenneth R Koedinger. 2016. Learning curve analysis for programming: Which concepts do students struggle with? In *ICER*, volume 16, pages 143–151. ACM.
- Rafaella Sampaio de Alencar, Mehmet Arif Demirtas, Aditya Soukarjya Saha, Yang Shi, and Peter Brusilovsky. 2025. Integrating expert knowledge with automated knowledge component extraction for student modeling. In *33rd ACM Conference on User Modeling, Adaptation and Personalization, UMAP 2025*, pages 307–312. ACM.
- Yang Shi, Robin Schmucker, Min Chi, Tiffany Barnes, and Thomas Price. 2023. Kc-finder: Automated knowledge component discovery for programming problems. In *the 16th International Conference on Educational Data Mining (EDM 2023)*, pages 28–39.
- Yang Shi, Robin Schmucker, Keith Tran, John Bacher, Kenneth Koedinger, Thomas Price, Min Chi, and Tiffany Barnes. 2024. The knowledge component attribution problem for programming: Methods and tradeoffs with limited labeled data. *Journal of Educational Data Mining*, 16(1).