Adaptive Learning Curve Analytics with LLM-KC Identifiers for Knowledge Component Refinement

Jing Fan
Aalto University
Espoo, Finland
jing.fan@aalto.fi

Tsvetomila Mihaylova
Aalto University
Espoo, Finland
tsvetomila.mihaylova@aalto.fi

Bita Akram
North Carolina State University
Raleigh, NC, USA
bakram@ncsu.edu

Narges Norouzi
University of California, Berkeley
Berkeley, CA, USA
norouzi@berkeley.edu

Peter Brusilovsky
University of Pittsburgh
Pittsburgh, PA, USA
peterb@pitt.edu

Arto Hellas
Aalto University
Espoo, Finland
arto.hellas@aalto.fi

Juho Leinonen
Aalto University
Espoo, Finland
juho.2.leinonen@aalto.fi

Abstract

Accurate modeling of student knowledge is essential for delivering timely, targeted feedback in Intelligent Tutoring Systems (ITS). Knowledge Components (KCs)—discrete units of domain knowledge—have traditionally been handcrafted by experts, a process that is both time-consuming and difficult to scale. In this work, we replicate a recent Large Language Model (LLM)-based approach to automate KC extraction called LLM-KC Identifier (LLM-KCI) and extend on it by evaluating the extracted KCs using learning curve analysis. By comparing LLM-generated KCs against expert-annotated counterparts in an introductory programming course, we demonstrate that LLMs not only match experts in capturing core concepts but also bring unique advantages: consistent identification across diverse assignments and scalability. Through static overlap metrics (Jaccard similarity, overlap coefficient) and learning curve analyses, we show that certain LLMs (e.g., GPT-4o, DeepSeekR1) produce error-rate trajectories as smooth or smoother than expert-annotated KC models, effectively isolating student learning trends. Our findings suggest that automated KC extraction can become a mainstream tool for personalized learning analytics, enabling educators to rapidly adapt curriculum and interventions at scale.

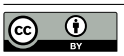
CCS Concepts

• **Social and professional topics** → **Computing education**.

Keywords

Learning Analytics, LLMs, Knowledge Component

ACM Reference Format:
Jing Fan, Tsvetomila Mihaylova, Bita Akram, Narges Norouzi, Peter Brusilovsky, Arto Hellas, and Juho Leinonen. 2025. Adaptive Learning Curve Analytics with LLM-KC Identifiers for Knowledge Component Refinement. In *UK and Ireland Computing Education Research Conference (UKICER 2025), September 04–05, 2025, Edinburgh, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3754508.3754514



UKICER 2025, Edinburgh, United Kingdom

ACM ISBN 979-8-4007-2078-9/25/09
https://doi.org/10.1145/3754508.3754514

1 Introduction

Knowledge Components (KCs)—the core cognitive skills or concepts underpinning students' problem solving—in the Knowledge-Learning-Instruction (KLI) framework are viewed as learned cognitive units inferred from related task performance, guiding course design and robust student modeling [12, 13, 17, 19]. Although KCs occupy a central role in Intelligent Tutoring Systems (ITS), their exploration in Computing Education Research (CER) has been relatively limited [6, 10, 16]. To address this, automated tools such as KC-Finder, based on deep neural networks, map expert-defined KCs to programming exercises by analyzing student submissions [21, 24, 28, 30], while recent advances in Large Language Models (LLMs) enable further detection and analysis of KCs in programming exercises [18, 19, 23]. Despite the potential of LLMs to reduce manpower and accelerate KC generation, systematic testing and quality evaluation of automatically generated KCs—especially comprehensive cross-field and cross-model validation—remains underexplored [5, 7, 15, 19, 26]. Meanwhile, learning curve models in educational data mining link practice opportunities with performance to reveal how mastery of KCs evolves over time [22, 25, 31].

Despite these advances, significant gaps remain: most LLM-based KC frameworks lack systematic evaluation and rigorous tests of generalizability across diverse programming languages and prompt complexities. Moreover, few studies analyze the learning curves of extracted KCs, limiting insights into student learning trajectories.

This paper presents an end-to-end pipeline that extends the LLM-KCI framework [19] by integrating LLMs for automated KC mapping and learning curve analysis. We address the following research questions:

RQ1 What are the characteristics of KCs identified by LLMs, and how do they differ from KCs annotated by experts?

RQ2 What is the quality of extracted KCs in capturing units of knowledge learned by students?

To address these questions, we extract KCs from Dart code commits with low- and high-complexity prompts. We then examine the quality of the KCs by demonstrating the students' learning curves obtained from a set of KCs.

2 Background

KCs are widely used for Knowledge Tracing (KT) [13], an important component of most ITS frameworks [1]. A KC is an individual unit of learning that students are supposed to learn. For example, in the context of introductory programming, one KC could be "printing" and another "for-loops". KCs can be at different levels of granularity: for example, a higher-level KC could be "loops" while a lower-level KC could split this into "for-loops" and "while-loops" separately. Most modern knowledge tracing algorithms utilize KCs to predict the student's knowledge of course concepts [1].

However, one downside of using KCs is that, traditionally, an expert had to develop the knowledge model—a group of KCs—that describes the content that students are supposed to learn [3]. Furthermore, to be able to perform KT, each exercise must be tagged with the KCs that are related to the exercise, which takes a lot of manual effort [3]. Some prior work before LLMs utilized automated methods to come up with knowledge models [8, 27]. Shi et al. [27] used DNNs to automatically identify KCs and evaluated them using learning curves. Goutte et al. [8] used probabilistic keyword extraction and found that they could accurately describe latent KCs.

LLMs provide new opportunities to automatically construct knowledge models and tag exercises with relevant KCs. Niousha et al. [19] built LLM-KCI, an LLM-based pipeline to automatically extract KCs from Python programming exercises based on the problem description, the program solution, or both. They found that GPT-4 [2] was able to successfully identify relevant KCs.

In our work, we evaluate the generalizability of LLM-KCI by applying the existing pipeline in a new context, using Dart programming exercises. We also extend the LLM-KCI work by evaluating the representation power of extracted KCs using learning curves, which is a commonly used method to evaluate the quality of KCs in representing units of knowledge acquired by students [25]. For good quality KCs, it is expected that students' error rates for subsequent attempts on that KC will be lower, representing a smooth downward slope when the learning curve is depicted [4]. To the best of our knowledge, no prior work has evaluated the quality of LLM-extracted KCs using learning curves.

3 Methodology

3.1 Context and Data

Our study uses data from an open online introductory programming course offered by Aalto University. Students in the course are both degree students and lifelong learners. The course uses the Dart programming language and has interactive online learning materials with intertwined theory and practice. Programming assignments are completed in an embedded online editor that supports syntax highlighting and feedback, allows program execution, and enables submission for automated assessment.

In this study, we focus on the first 45 programming assignments of the course. We created a dataset consisting of the problem statement and a sample solution for each assignment. The assignments were manually annotated by the course instructor, who annotated each assignment with (1) high-level KCs that we call key learnables, and (2) low-level KCs. These form the ground truth for the data.

We also extracted student submission data for analysis. The submission data contains students' programs, the problem description, submission time, and results from running the submission through automated assessment. Only data from students who provided research consent are used. Our dataset is described in Table 1.

Table 1: High-Level Descriptive Statistics.

Statistic	Count
Number of exercises	45
Number of key learnables	23
Number of low-level KCs	96
Number of students	6,955
Total number of submissions	289,450

3.2 Research Process Design

Figure 1 shows our extended LLM-KCI-based learning curve analysis method. The main extension to Niousha's LLM-KCI process [19] is the learning curve analysis. We benchmarked LLM-generated KCs using our dataset, including problem descriptions, solutions, and instructor-provided ground-truth KCs. For LLM extraction, we used GPT-4 [2], GPT-4o [11], o3-mini [20], Llama-2-70b [29], Llama4-Scout-17B-16E-Instruct [14] and DeepseekR1 [9]. Below, we outline our research process in seven stages.

Stage 1: Dataset Preparation. Stage 1 involves dataset preparation, with an expert annotation of ground-truth KCs, and developing two prompts with different levels of instruction details. Our dataset includes Dart programming problems, including problem descriptions and standard solutions. The two prompts that we try are: 1) *a simple version **(S)***, which contains only a brief set of instructions, and 2) *a complex version **(C)***, which contains a more detailed instruction with one-shot example. The set of ground-truth KCs is our baseline for evaluating the KCs generated by LLMs. We evaluate two prompts to explore whether the complexity of the instructions affects the generated KCs.

Stage 2: Context Consideration. In stage 2, LLMs generate KCs for three context scenarios: problem-only, program-only, and all (problem and program) based on the context set up as proposed by [19], with both simple (S) and complex (C) prompts. We aim to identify a combination of context and prompting technique that leads to the highest accuracy and precision in extracting KCs. Examples of simple and complex prompts as below:

Prompt(S): Extract the knowledge components from the following (problems/programs/Programming tasks). Generate an English version of them in numbered bullet format. Prompt (C): We are building an intelligent tutoring system for novice programmers. For the intelligent tutoring system, we need knowledge components. Extract the course-level knowledge components from the following Problems/Programs/Programming tasks. One example is shown

Figure 1: LLM-KCI-based learning curve analysis pipeline

below... Generate an English version of them in numbered bullet format.

Stage 3: KC List aggregation. In stage 3, we aggregate all the generated KC lists based on all combinations of context and prompting techniques for various LLMs. **Stage 4: Semantic Duplicate Removal.** We remove KCs that are semantic duplicates from the lists generated in stage 3.

Stage 5: Final KC Extraction using a Reference KC List. We feed the final aggregated KC lists that are generated in stage 4 to different LLMs and have them generate the final KCs corresponding to each description/program/both in three different scenarios (problem-only, program-only, or both).

Stage 6: Final Determination of KC Set. We determine the refined KC set based on the answers given by LLMs.

Stage 7: Learning curve Analysis. We compare quality of ground-truth KCs and the refined set of KCs generated by LLMs in stage 6 by using learning curves.

In summary, our pipeline examines how prompt complexity (simple vs. complex) and context scenarios (problem-only, program-only, or all) influence the quality of LLM-generated KCs compared to ground-truth annotations. By analyzing the resulting learning curves, we identify the most effective prompt-context combinations for generating KCs that accurately capture student knowledge.

3.3 Comparison of LLM-Generated KCs and Ground-Truth KCs

To evaluate the accuracy and coverage of LLM-generated KCs compared to ground-truth KCs, we use the following metrics based on Niousha's study [19]:

TP True Positive (TP) is the number of LLM-generated KCs present in the ground-truth KC set.

FN False Negative (FN) is the number of KCs not identified through LLM generation but present in the ground-truth.

FP False Positive (FP) is the number of KCs identified by LLM but not present in the ground-truth.

Let G be the set of ground-truth KCs and L be one of the KC sets generated by LLM using each of the 6 context-prompt combination scenarios ([problem-only, program-only, or both] × [simple or complex prompt]). We then define:

$$\text{TP} = |G \cap L|, \quad \text{FP} = |L \setminus G|, \quad \text{FN} = |G \setminus L|.$$

The Jaccard index (J) is given by:

$$J = \frac{\text{TP}}{\text{TP} + \text{FP} + \text{FN}}$$

and the overlap coefficient (O) is:

$$O = \frac{\text{TP}}{\min(|G|, |L|)}.$$

3.4 Learning Curve Analysis

To ensure clean and meaningful learning curve analysis, we preprocess the data to remove noise and preserve temporal integrity. Specifically, we eliminate "pseudo attempts" (repeated submissions of identical code) to avoid inflating effort counts, and we maintain the original timestamps to reflect the true progression of student activity across problems and KCs. The example below provides an illustrative example of how pre-processing is performed.

Illustrative Example. Suppose student "Jane" works on **Problem 1** and submits solutions at timestamps T_1, T_2, T_3, T_5. If submissions at T_2 and T_3 are identical, T_3 is removed, leaving $\{T_1, T_2, T_5\}$. Jane also submits a solution for **Problem 2** at timestamp T_4, which involves the same KC (KC1) and is non-duplicate, so it is included.

Combining these across problems, the deduplicated submission sequence for KC1 is:

$$\{T_1,\ T_2,\ T_4,\ T_5\},$$

corresponding to four meaningful attempts. These timestamps are then chronologically reordered as:

$$\{t_1,\ t_2,\ t_3,\ t_4\},$$

representing Jane's 1st through 4th attempts on KC1 and forming the x-axis of her KC1 learning curve. The following subsections formally outline our pre-processing steps.

Sequential Timestamp Sorting and Pairwise Deduplication Removal. Submission records are grouped by problem ID, then by student ID, and each student's submissions are sorted in ascending order of timestamp (T).

$$\mathrm{T}_1 < \mathrm{T}_2 < \cdots < \mathrm{T}_i.$$

We then remove duplicates:

$$(\mathrm{T}_1 \leftrightarrow \mathrm{T}_2),\ (\mathrm{T}_2 \leftrightarrow \mathrm{T}_3),\ \ldots,\ (\mathrm{T}_{i-1} \leftrightarrow \mathrm{T}_i).$$

If two adjacent submissions are identical, we delete the latter one (the one with the larger timestamp). We repeat the above pairwise comparisons until no two consecutive submissions are duplicates. Deletions remove only redundant records; original timestamps remain unchanged to preserve the true chronological order.

Global KC-Level Attempt Sequencing. A given KC may appear in multiple problems for the same student. After removing all duplicate submissions for each student–KC pair for all relevant problems, we construct a timeline of unique attempts. In this context, k denotes the total number of deduplicated attempts a student has made on a given KC across all relevant problems, and let $t_k^{(\mathrm{KC})}$ represent the timestamp of the k^{th} attempt, i.e., final value in the ordered sequence

$$t_1^{(\mathrm{KC})} < t_2^{(\mathrm{KC})} < \cdots < t_k^{(\mathrm{KC})}.$$

We use the attempt index (1^{st}, 2^{nd}, …, k^{th}) as the x-axis in the student's learning curve for that KC.

KC Correctness Assessment and Error Rate Calculation. In this step, we analyze the correctness of code submissions. In the dataset we collected, we see the final score for each submission to be able to identify "correct" and "incorrect" submissions.

Here each $P_{s,j}^{(k)}$ is the result correct of k^{th} code submission on KC_j at time stamp $t_k^{(KC_j)}$. We introduce a binary correctness indicator for the student s, at time stamp k, and on the j^{th} KC:

$$P_{s,j}^{(k)} = \begin{cases} 1, & \text{If the } k^{\text{th}} \text{ attempt with } KC_j \text{ is correct,} \\ 0, & \text{otherwise.} \end{cases}$$

For each KC_j, we plot the global correctness-rate curve over the attempt indices k. Recall that each student s has a binary indicator $P_{s,j}^{(k)}$. Define the aggregated counts in the ordinal attempt k as

$$N_{KC_j}(k) = \sum_s P_{s,KC_j}^{(k)}, \quad D_{KC_j}(k) = \sum_s \mathbf{1}\big(n_{s,KC_j} \geq k\big),$$

where n_{s,KC_j} is the total number of attempts by student s on KC_j.

$$n_{s,KC_j} \in \mathbb{N}, \quad \mathbf{1}(\cdot) \text{ is the indicator function.}$$

Here,

- $N_{KC_j}(k)$ is the total number of correct k^{th} attempts related to KC_j among all students;
- $D_{KC_j}(k)$ is the total number of k^{th} attempts on KC_j among all students.

The correctness-rate at attempt k is then

$$\text{CR}_{KC_j}(k) \;=\; \frac{N_{KC_j}(k)}{D_{KC_j}(k)} \;\times 100.$$

The corresponding error rate is

$$\text{ER}_{KC_j}(k) \;=\; \frac{D_{KC_j}(k) - N_{KC_j}(k)}{D_{KC_j}(k)} \;\times 100 \;=\; 100 - \text{CR}_{KC_j}(k).$$

In Jane's example, the deduplication process results in $\{T_1, T_2, T_4, T_5\}$ showing 4 attempts at $k \in \{1, 2, 3, 4\}$. Aggregating across all students gives $\{N_1(k)\}$ and $\{D_1(k)\}$ for $k \in \{1, \ldots, 4\}$, which produce the four points on the global KC1 correctness rate curve.

Learning-Curve Construction. In each figure of the KC learning curve, the horizontal axis is the attempt index k (that is, the student's k^{th} attempt for KC_j), and the vertical axis is $\text{ER}_{KC_j}(k)$ - the percentage of students who had incorrect submission for the implementation of KC_j on their k^{th} attempt. We plot points $\big(k, \text{CR}_{KC_j}(k)\big)$ for $k \in \{1, \ldots, M_{KC_j}\}$. M_{KC_j} is defined as:

$$M_{KC_j} = \max\{\, k \mid D_{KC_j}(k) \geq 10\}.$$

We exclude data points that have fewer than 10 submissions.

4 Results

4.1 Replication Results

In our study, GPT-4 achieves 85% J on complex prompts in content (Problem)—40 percentage points above the original experiment by Niousha et al. [19]. Llama-2-70b reaches 100% on both complex and simple prompts, 10 percentage points above the prior best, GPT-3.5. GPT-4o leads on complex prompts (J = 75%, O = 96%), while DeepSeekR tops simple prompts in content (All) (J = 67%, O = 100%) versus key-learnable GT KCs. GPT-4 posts 53% J and 56% O in content (All) in the previous study. Our study exceeds the original experiment by 14–22 and 40–44 percentage points, respectively. Our improvement over previous work lies in the use of more complex prompts for a better LLM-generated context. Furthermore, when GT contains fewer KCs, LLM's broader coverage increases Jaccard and Overlap scores, as both metrics are more sensitive to intersection under small cardinality, even with some redundancy.

Table 2: Jaccard index (J) and Overlap coefficient (O) of different LLMs using prompts (S/C) of different complexity for identifying Low-level KCs and Key-learnables. Top performers in each column are bolded.

	Low-level GT KCs (Low)		Key-learnables GT KCs (Key)		KC Count (| L|)
LLMs	J (S/C)	O (S/C)	J (S/C)	O (S/C)	(S/C)
All					
GPT-4	8%/**25%**	70%/92%	28%/38%	70%/59%	10/**25**
GPT-4o	13%/23%	80%/84%	58%/**75%**	93%/**96%**	**15**/**25**
o3-mini	**14%**/17%	**100%**/**100%**	48%/54%	92%/93%	12/15
DeepSeekR1	**14%**/11%	93%/**100%**	**67%**/56%	**100%**/90%	14/10
Llama-2-70b	11%/17%	83%/88%	61%/44%	92%/71%	12/17
Llama-4-Scout	8%/9%	**100%**/**100%**	30%/24%	86%/75%	7/8
Program (Prog)					
GPT-4	11%/15%	**100%**/82%	**63%**/54%	**100%**/**88%**	10/17
GPT-4o	10%/**25%**	75%/92%	42%/**76%**	92%/86%	12/**25**
o3-mini	12%/20%	**100%**/**100%**	38%/52%	82%/78%	11/18
DeepSeekR1	**17%**/17%	73%/**100%**	50%/62%	59%/87%	**22**/15
Llama-2-70b	11%/21%	91%/95%	53%/57%	82%/80%	11/20
Llama-4-Scout	7%/18%	86%/**100%**	46%/57%	86%/75%	7/16
Problem (Prob)					
GPT-4	**15%**/14%	70%/**100%**	**54%**/**85%**	75%/92%	**20**/12
GPT-4o	8%/**20%**	88%/90%	35%/76%	88%/95%	8/**20**
o3-mini	12%/16%	92%/**100%**	48%/60%	92%/86%	12/14
DeepSeekR1	**15%**/18%	78%/**100%**	50%/74%	75%/78%	18/16
Llama-2-70b	4%/10%	**100%**/**100%**	18%/33%	**100%**/**100%**	4/9
Llama-4-Scout	5%/14%	38%/**100%**	3%/67%	8%/**100%**	13/12

4.2 Comparison Between GT and LLM KCs

Across all three contexts—All, Program, and Problem—as shown in Table 2. GPT-4o with a complex prompt delivers the strongest key-learnables performance ($J = 75\%$, $O = 96\%$, $|L| = 25$), o3-mini under a simple prompt achieves perfect low-level overlap ($J = 14\%$, $O = 100\%$), and DeepSeekR1 with a simple prompt leads on key-learnables among simple-prompt runs ($J = 67\%$, $O = 100\%$) at *All*. Under a simple prompt, GPT-4 achieves a Jaccard index of $J = 63\%$ and a perfect overlap coefficient of $O = 100\%$ on key-learnables at *Program*. Llama-2-70b (simple & complex) attains perfect overlap ($O = 100\%$) on both low-level KCs and key-learnables at *Problem*, indicating zero false positives. In contrast, GPT-4 under a simple prompt recovers only $J = 15\%$ of low-level KCs and $J = 54\%$ of key-learnables, reflecting more limited coverage despite high precision.

As shown in Figure 2, DeepSeekR1 stands out under simple prompts with the highest average Jaccard index at **35.5%**, while GPT-4o leads under complex prompts with **49.2%** recall. In terms of precision, o3-mini is unrivaled—**93.0%** overlap on simple prompts and **92.8%** on complex—whereas Llama-4-Scout trails in recall under simple prompts at just **16.5%**.

As shown in Figure 3, complex prompts yield consistent gains in recall (J) across all scenarios—most importantly +31.2 percentage points (pp) for *Prob-Key* and +11.0 pp for *Prog-Key*, with smaller improvements for *All-Low* (+5.7 pp), *Prog-Low* (+8.0 pp) and *Prob-Low* (+5.5 pp), and a marginal decline for *All-Key* (−0.2 pp). Precision (O) is high under simple prompts (73.0 %–88.8 %) and is only moderately affected by added complexity: it increases by up to +20.7 pp for *Prob-Low*, but decreases by 8.2 pp for *All-Key* and by 1.2 pp for *Prog-Key*. Together, these findings indicate that complex prompts substantially improve recall, especially for key learnables, while leaving an already high precision largely unchanged.

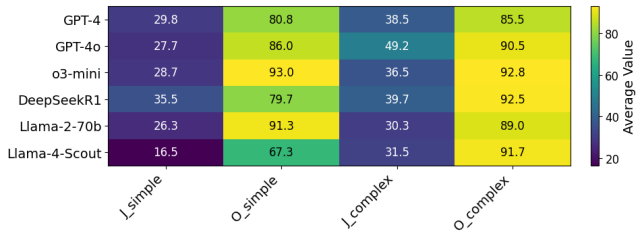


Figure 2: Average Jaccard and Overlap Scores of LLMs under Simple vs. Complex Prompts

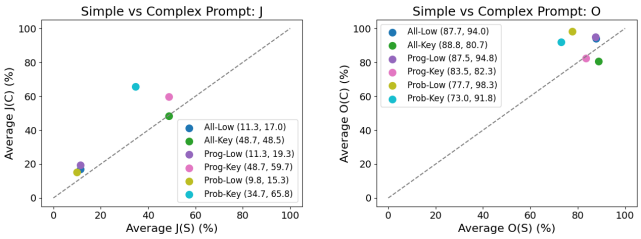


Figure 3: Scatter plots comparing average J and O for simple (S) versus complex (C) prompts across all task types (All, Program, Problem) and GT KC levels (Low, Key).

4.3 KC Correctness and Learning Curves

We evaluate the ground-truth and generated KCs by analyzing their learning curves. For each KC, we show the average error from all students for their corresponding attempt with each KC.

For each model, we evaluate the learning curve *trend* by fitting a linear model to each curve. To describe the trend, we consider the slope of the curve, and the average distance of all the points in the curve to this line. We fit two lines: one line over the full sequence of attempts for the KC (*full trend*), and one line over the first 5 attempts (*early trend*). The intuition behind early trends is that usually the first few attempts for each KC can show learning, and for further attempts, the curve can include noise from interactions with other KCs. It is expected that a "good KC" should have a negative slope, i.e., the error decreases with the number of attempts.

In Table 3, we show the full and early trends for ground-truth and LLM-generated KCs, grouped by prompt type. For each type, we show the averages of all input types – problem-only, program-only, and both. The data for all KCs are very noisy – the deviation from the line is high for all KCs. The ground-truth KCs have higher slopes than the LLM-generated ones, and the models with the lowest average slopes are GPT-4o, GPT-4, and o3-mini.

On Figure 4, we show examples of "good" and "bad" learning curves. In the first few attempts of the KC, the "good" learning curve shows a decreasing error rate. Since we make the learning curves based on the KCs per assignment, the presence of multiple KCs per assignment adds noise to the performance of this KC. The examples are from ground-truth KCs, but they are also representative of the LLM-generated KCs.

Table 3: Averaged error metrics for ground-truth and LLM-generated KCs (slope ± average distance from the line). *Full* shows the slope over the whole learning curve, and *Early* shows the slope over the first 5 attempts for each KC. We show the results per prompt type, averaged for program-only, problem-only, and both.

Model	S		C	
	Full	Early	Full	Early
GPT-4	0.004 ± 0.079	0.043 ± 0.043	0.006 ± 0.077	0.036 ± 0.034
GPT-4o	0.006 ± 0.079	0.044 ± 0.044	0.008 ± 0.078	0.042 ± 0.040
GPT-o3-mini	0.006 ± 0.081	0.041 ± 0.041	0.005 ± 0.078	0.033 ± 0.035
DeepSeekR1	0.011 ± 0.076	0.036 ± 0.038	0.006 ± 0.076	0.029 ± 0.041
Llama-2-70b	0.022 ± 0.074	0.064 ± 0.040	0.020 ± 0.081	0.063 ± 0.039
Llama-4	0.025 ± 0.079	0.061 ± 0.040	0.016 ± 0.077	0.043 ± 0.042
	Ground-truth			
Key Learnables	0.025 ± 0.068	0.055 ± 0.035		
Low-Level	0.020 ± 0.071	0.060 ± 0.034		

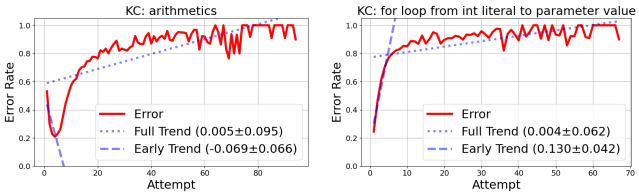


Figure 4: Examples for good (left) and bad (right) learning curves.

5 Discussion

5.1 Context-Type Comparison

The striking disparity in GPT-4's key-learnable extraction between code and text reveals a clear content sensitivity: under a simple prompt, the model attains a Jaccard index of 63% with perfect precision ($O = 100\%$) on programmatic inputs, yet on purely textual problems its recall falls to 54% and precision to 75%. This suggests that GPT-4's pattern-matching abilities align naturally with the rigid syntactic structures of code—minimizing hallucinations—but that free-form prose, with its richer semantics and variable phrasing, overwhelms a minimalist prompt. Therefore, even when using the same model and prompting strategy, structured code tasks may require only light prompting to achieve high accuracy, whereas text-centric tasks benefit from more elaborate prompts or retrieval-augmentation to shore up both coverage and correctness.

5.2 Model Comparative Advantages

As shown in the Figure 2, we can classify the models into three prototypes: precision-focused prototypes represented by `o3-mini`, which achieve zero false positives under simple prompts; coverage-focused prototypes represented by `GPT-4o` and `GPT-4`, which have the highest recall for key learnable concepts under complex prompts rich in examples; and weak candidate models `Llama-2-70b` and `Llama-4-Scout`, which show low recall and moderate precision under both prompt modes. This shows that while example injection can significantly improve recall (GPT-4o is most robust for program logic, while GPT-4 shows the largest improvement for text-based

reasoning), it also sacrifices precision, emphasizing the need for balance between increasing coverage and the risk of false positives.

5.3 Prompt Complexity and Automation.

Under simple prompts (S), which serve as a baseline, models receive only minimal instructions with no ground-truth KC example and thus demonstrate their out-of-the-box performance. In contrast, complex prompts (C) include a one-shot example of ground-truth KC and operate in a semi-automated human–model collaborative workflow: the one-shot example directly conveys potential target labels, resulting in substantial gains in true-positive rates and Jaccard index (an average uplift of 5.5–31.2 percentage points across Problen, Program, and All, most pronounced for Problem to identify key learnable), while inevitably introducing some annotation bias. Crucially, precision remains stable: The overlap coefficients already lie between 73.0 % and 100 % under S and shift by only −8.2 to +20.7 percentage points under C, indicating that recall improvements do not come at the expense of precision. This trade-off between "zero manual-cost" automation and "exemplar-guided" accuracy highlights a spectrum along which practitioners can balance throughput and human oversight based on application requirements.

5.4 Error-Rate Slopes for GT and LLM KCs

From Table 3, we observe that GPT-4, GPT-4o, and o3-mini generate KCs with full-curve slopes of only 0.004–0.006, substantially lower than the 0.025 observed for ground-truth KCs. In the early phase, their slopes of 0.041–0.044 also fall below the ground-truth value of 0.055. This indicates that, although LLM-extracted KCs have not yet reached ideal negative slopes, they more closely capture the "error rate decreasing with practice" trend than human-annotated GT KCs—at least, on average, they exhibit weaker positive growth or a greater tendency toward downward trends.

The larger models Llama-4 and Llama-2-70b exhibit higher full-trend slopes (0.022–0.025) and early-trend slopes (0.061–0.064) than the GPT series, suggesting that the KCs they extract are less likely to demonstrate sustained declines in error rate. DeepSeekR1 achieves a lower early-trend slope (0.036), showing its relative advantage in capturing negative or smaller positive trends during initial practice opportunities. This implies that model capacity alone does not determine KC extraction quality; instead, internal architecture and pre-training/fine-tuning strategies play a more significant role.

Under the S (Simple) prompt context, the average slopes are the lowest. Under the C (Complex) context, slopes do not improve substantially, and noise levels (average deviation) even increase slightly. This suggests that complex prompts do not significantly enhance the models' ability to extract KCs that effectively represent students' learning through a "steady error decrease with practice" trend, and that prompt engineering offers limited benefit for extracting effective KCs as evaluated by learning curves, even though the complex prompt led to better performance as evaluated by the Jaccard index and Overlap coefficient.

As illustrated in Figure 4, even when individual KC learning curves exhibit downward trends, interference from multiple KCs within the same assignment often induces "rebounds" or "plateaus," resulting in high overall noise. The average deviation for ground-truth KCs (≈ 0.068) is nontrivial, and LLM-generated KCs show even higher deviations (≈ 0.074–0.081), indicating that multi-KC interference remains a challenge for automated KC identification.

In summary, lower slopes more accurately reflect the expected "error rate decreasing" trend. From this perspective, GPT-4, GPT-4o, and o3-mini are closer to pedagogical expectations; although they have not yet achieved ideal negative slopes, they outperform human-annotated ground-truth KCs in several respects.

5.5 Limitations and Future Work

This study is constrained by a limited set of tasks and models, which warrants a broader evaluation across diverse domains (e.g., different subjects in addition to programming). In addition, our learning curve analysis only considers correctness at the assignment level and does not isolate individual KCs within each assignment although an assignment with multiple KCs could have some correct even if the overall result is that the submission is incorrect. The predominantly flat or noisy error-rate trends suggest that current KC extraction methods cannot reliably disentangle pure concept learning. Additionally, the GT KC quality was poor as measured by learning curves, which suggests that learning curves might not be the ideal way to measure the quality of KCs in our context.

In the future, we will (1) extend our evaluation to diverse domains, (2) formalize KCs with concise human-readable definitions and deploy a hybrid LLM–expert two-phase validation pipeline to yield precise per-KC success rate, and (3) improve learning curve modeling via piecewise regression, multimodal signals, and clustering-based de-noising to isolate true learning trends.

6 Conclusion

In this study, we replicate and extend the LLM-KCI framework originally proposed by Niousha et al. ([19]), introducing a more comprehensive evaluation of knowledge components (KCs) generated by human annotations and LLMs under different prompts strategies. Unlike previous studies, we benchmark static overlap metrics and dynamic learning curve trends to evaluate the quality of KCs. Results show that more complex prompts enhance the ability of LLMs to generate high-level KCs that are consistent with expert annotations. Importantly, our learning curve analysis finds that there is a lot of noise in both the ground truth (GT) and LLM-generated KCs, revealing the limitations of current annotation practices. However, we find that the error rate trends of some LLMs are as stable as or even more stable than those of human-annotated KCs. These findings highlight our key contribution: showing that under a well-calibrated prompting strategy, LLMs can serve as a viable and scalable alternative to manual KC authoring, potentially enabling reliable KC tagging with less human effort.

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 was used to polish the text.

References

[1] Ghodai Abdelrahman, Qing Wang, and Bernardo Nunes. 2023. Knowledge tracing: A survey. *Comput. Surveys* 55, 11 (2023), 1–37.

[2] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[3] Vincent Aleven and Kenneth R Koedinger. 2013. Knowledge component (KC) approaches to learner modeling. *Design recommendations for intelligent tutoring systems* 1 (2013), 165–182.

[4] Hao Cen, Kenneth Koedinger, and Brian Junker. 2006. Learning factors analysis–a general method for cognitive model evaluation and improvement. In *International conference on intelligent tutoring systems*. Springer, 164–175.

[5] Yongwan Cho, Rabia Emhamed AlMamlook, and Tasnim Gharaibeh. 2024. A Systematic Review of Knowledge Tracing and Large Language Models in Education: Opportunities, Issues, and Future Research. *arXiv preprint arXiv:2412.09248* (2024).

[6] László Csépányi-Fürjes and László Kovács. 2024. Evolving graph based knowledge space model for tutoring systems. *Pollack Periodica* 19, 3 (2024), 40–45.

[7] Zhangqi Duan, Nigel Fernandez, Sri Kanakadandi, Bita Akram, and Andrew Lan. 2025. Automated Knowledge Component Generation and Knowledge Tracing for Coding Problems. *arXiv preprint arXiv:2502.18632* (2025).

[8] Cyril Goutte, Guillaume Durand, and Serge Léger. 2015. Towards automatic description of knowledge components. In *Proceedings of the tenth workshop on innovative use of nlp for building educational applications*. 75–80.

[9] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948* (2025).

[10] Fatima-Zohra Hibbi. 2023. A Literature Review of Tutoring Systems: Pedagogical Approach and Tools. In *The International Conference on Artificial Intelligence and Smart Environment*. Springer, 99–104.

[11] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276* (2024).

[12] Steffi Knorn and Damiano Varagnolo. 2020. Automatic control: The natural approach for a quantitative-based personalized education. *IFAC-PapersOnLine* 53, 2 (2020), 17326–17331.

[13] 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 (2012), 757–798.

[14] Meta. 2025. *The Llama 4 herd: The beginning of a new era of natively multimodal AI innovation*. Technical Report. Meta. https://ai.meta.com/blog/llama-4-multimodal-intelligence/

[15] Steven Moore, Robin Schmucker, Tom Mitchell, and John Stamper. 2024. Automated generation and tagging of knowledge components from multiple-choice questions. In *Proceedings of the eleventh ACM conference on learning@ scale*. 122–133.

[16] Brent Morgan, Marie Hogan, Andrew J Hampton, Anne Lippert, and Arthur C Graesser. 2020. The need for personalized learning and the potential of intelligent tutoring systems. In *Handbook of learning from multiple representations and perspectives*. Routledge, 495–512.

[17] Huy Nguyen, Yeyu Wang, John Stamper, and Bruce M McLaren. 2019. Using Knowledge Component Modeling to Increase Domain Understanding in a Digital Learning Game. *International Educational Data Mining Society* (2019).

[18] Rose Niousha, Muntasir Hoq, Bita 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*. 1762–1763.

[19] Rose Niousha, Abigail O'Neill, Ethan Chen, Vedansh Malhotra, Bita Akram, and Narges Norouzi. 2025. 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*. 1557–1558.

[20] OpenAI. 2024. *OpenAI o3 mini*. Technical Report. OepnAI. https://openai.com/index/openai-o3-mini/

[21] Poorvaja Penmetsa. 2021. *Investigate effectiveness of code features in knowledge tracing task on novice programming course*. North Carolina State University.

[22] MI Ponomarenko, DE Peshkov, A Yu Vishnyakova, NV Papulovskaya, and NN Dubinin. 2023. Effectiveness of the learning tool based on the learning curve analysis. In *AIP Conference Proceedings*, Vol. 2812. AIP Publishing.

[23] James Prather, Juho Leinonen, Natalie Kiesler, Jamie Gorson Benario, Sam Lau, Stephen MacNeil, Narges Norouzi, Simone Opel, Vee Pettit, Leo Porter, et al. 2025. Beyond the Hype: A Comprehensive Review of Current Trends in Generative AI Research, Teaching Practices, and Tools. *2024 Working Group Reports on Innovation and Technology in Computer Science Education* (2025), 300–338.

[24] Yanjun Pu, Wenjun Wu, Tianhao Peng, Fang Liu, Yu Liang, Xin Yu, Ruibo Chen, and Pu Feng. 2022. Embedding cognitive framework with self-attention for interpretable knowledge tracing. *Scientific Reports* 12, 1 (2022), 17536.

[25] Kelly Rivers, Erik Harpstead, and Kenneth R Koedinger. 2016. Learning curve analysis for programming: Which concepts do students struggle with?. In *ICER*, Vol. 16. ACM, 143–151.

[26] Alexander Scarlatos, Ryan S Baker, and Andrew Lan. 2025. Exploring knowledge tracing in tutor-student dialogues using llms. In *Proceedings of the 15th International Learning Analytics and Knowledge Conference*. 249–259.

[27] Yang Shi, Robin Schmucker, Min Chi, Tiffany Barnes, and Thomas Price. 2023. KC-Finder: Automated Knowledge Component Discovery for Programming Problems. *International Educational Data Mining Society* (2023).

[28] 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 (2024), 1–33.

[29] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[30] Wentao Wang, Huifang Ma, Yan Zhao, Zhixin Li, and Xiangchun He. 2022. Tracking knowledge proficiency of students with calibrated Q-matrix. *Expert Systems with Applications* 192 (2022), 116454.

[31] Daniel Weitekamp III, Erik Harpstead, Christopher J MacLellan, Napol Rachatasumrit, and Kenneth R Koedinger. 2019. Toward Near Zero-Parameter Prediction Using a Computational Model of Student Learning. *International Educational Data Mining Society* (2019).