

Ontology-Guided Knowledge Component Extraction for Programming Exercises

Rully Agus Hendrawan^{1,2}, Shravya Suresh¹, Rafaella Sampaio de Alencar¹, Alice Micheli⁴, Peter Brusilovsky¹ and Sergey Sosnovsky⁴

¹*School of Computing and Information, University of Pittsburgh, Pittsburgh, United States*

²*Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia*

⁴*Utrecht University, Utrecht, Netherlands*

Abstract

Knowledge component annotation is important for adaptive learning systems, such as programming practice recommenders, but manually annotating programming exercises with fine-grained concepts is costly and difficult to scale. Large language models (LLMs) offer a promising alternative, yet open-ended LLM annotation can produce unstable labels that are difficult to align with existing curriculum models. This paper studies ontology-guided KC extraction as a human-AI collaboration task: human experts define a domain's conceptual structure, such as the Python ontology explored in our work, while LLMs perform scalable exercise-level annotation within that fixed label space. Our work has been motivated by the success of the CLAWS framework, a pioneer approach to annotate programming exercises with ontology concepts through KC-windowing and aggregation. We reproduce and extend the CLAWS approach and evaluate whether its performance remains reliable on a much broader range of LLM deployment classes: Compact Generalist API, Mixture of experts Generalist API, Efficient Generalist API, Open-Weight Generalist API, Small Generalist Local, and Code Specialist Local. We compare annotation quality delivered by these models against expert labels using precision, recall, F1, Jaccard similarity, and average prediction density. The results show that the ontology-guided LLM-based KC extraction can achieve strong expert agreement, but its calibration varies substantially between model classes.

Keywords

Ontology, Knowledge component, Concept extraction, CS education, Personalized learning

1. Introduction

Annotating learning content with knowledge components (KC) is important for all kinds of personalized learning systems, but traditional manual annotation by domain experts is hard to scale, as this process requires substantial time and domain expertise [1, 2]. For example, modern repositories of interactive learning content for programming [3] offer hundreds of content items with dozens new items added every week. These content items require fine-grained KC indexes to be found by course developers and to be timely recommended to learners with different levels of knowledge. Prior work has explored automated approaches for extracting programming concepts from code and student data [4, 5, 6]. But, scalable annotation remains challenging when the goal is to produce labels that are interpretable and aligned with curricula. Large language models (LLMs) offer a new opportunity for KC annotation because they can interpret natural-language exercise descriptions, program code, and educational context. Recent work has begun to use LLMs for KC generation, concept tagging, and educational annotation [7, 8, 9, 10]. However, open-ended LLM annotation can produce labels that vary in wording, granularity, and coverage across exercises or model runs. For adaptive learning systems, this variability is problematic because student progress must be tracked against a stable and reusable concept vocabulary.

This problem is especially relevant for computing education, because programming repositories can grow faster than expert annotation workflows. To facilitate access to new exercises, their labeling with

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

✉ ruhendrawan@pitt.edu (R. A. Hendrawan); shs670@pitt.edu (S. Suresh); ras555@pitt.edu (R. S. d. Alencar);

alice.micheli12@gmail.com (A. Micheli); peterb@pitt.edu (P. Brusilovsky); s.a.sosnovsky@uu.nl (S. Sosnovsky)

🆔 0000-0001-8541-0305 (R. A. Hendrawan); 0009-0008-0099-5446 (R. S. d. Alencar); 0009-0009-3324-1509 (A. Micheli);

0000-0002-1902-1464 (P. Brusilovsky); 0000-0001-8023-1770 (S. Sosnovsky)



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

KCs needs to be expedited as much as possible so that they can be indexed, connected to learner models and recommended.

With the continuous proliferation of LLMs, a decision to choose the right model for the task becomes difficult. Hosted API models may offer stronger instruction following, but they can introduce prohibitively-high cost, privacy, and dependency concerns. Local and open-weight models may offer more institutional control, but they may require additional output control or calibration. Therefore, evaluating across model classes helps identify desirable LLM behaviors for reliable deployment.

This paper examines ontology-guided KC extraction for programming exercises. Instead of treating KC extraction as a generation task, prompting an LLM to invent concept labels, we frame it as an annotation task constrained by an expert-authored ontology. Ontologies have long been used in educational technology to represent domain knowledge and support personalization [11, 12], particularly because in many fields (including programming), learning objectives when expressed in terms of concepts have hierarchical and relational structures that are natural to represent with the help of ontologies [13, 14]. They also provide other benefits such as development of shared, reusable, controlled and semantically consistent vocabularies of important notions in a domain. Hence, the approach, advocated in this research is to use an ontology to define the concept label space, while the LLM performs exercise-level annotation within that space.

The paper builds on the success of the CLAWS approach for ontology-guided KC extraction from programming learning content [15]. It demonstrated the promise of the approach using a single LLM, leaving open the question of whether the same method works reliably across different LLM deployment classes. We address this gap by performing an extensive exploration of the CLAWS approach across a broader range of models and annotation harness parameters. We contribute an analysis of how model class and harness parameters interact in a programming KC extraction task; including output compliance, batch stability, expert agreement, threshold sensitivity, and prediction density.

2. Literature Review

2.1. Automated KC Extraction for Programming

Recent work has used student submissions and learning traces to discover or refine programming KC [16, 5, 6]. These data-driven approaches connect KC modeling to observed student behavior, but automatically discovered features may still require interpretation before they can be used as curriculum-facing concept labels. Hybrid approaches address this limitation by combining automated extraction with expert knowledge, preserving interpretability while reducing manual annotation effort [17]. The present work follows this hybrid direction by using automated annotation while keeping the label space grounded in a human-authored concept model.

2.2. Ontologies as Shared Vocabulary

Ontologies provide a structured way to represent educational domain knowledge. They have been used to organize learning resources, support semantic indexing, structure course content, and enable personalization in adaptive systems [11, 12, 18, 19]. In programming education, ontologies are especially appropriate because programming knowledge has hierarchical and relational structure. Earlier work developed ontology-based representations for programming domains [13], and more recent work has extended this direction to Python programming concepts [14].

For KC annotation, ontologies are useful because they define a shared vocabulary and stable label space. This stability is important for adaptive learning systems, where student progress must be tracked consistently across activities and over time. If an annotation system generates labels freely, it may produce inconsistent labeling and granularity, or concepts that do not align with the curriculum. A fixed ontology reduces this problem by constraining annotation to a shared domain model.

2.3. Model Selection and Reliability

LLM performance varies across model families, deployment settings, and task formats. General benchmarks provide useful signals, but they do not fully determine performance on specialized educational annotation tasks. Holistic and arena-style evaluation work emphasizes that model quality depends on the task, context, and evaluation criterion [20, 21, 22]. For KC extraction, a model must do more than understand code or language: it must also follow instructions, remain within the target label space, and produce annotations that can be processed by downstream systems.

Structured-output reliability is therefore central to LLM-based annotation pipelines. Work on schema-constrained and structured generation shows that producing valid machine-readable output remains a nontrivial challenge for LLMs [23, 24]. This concern is particularly important in educational data mining, where malformed or inconsistent outputs can affect later student modeling, recommendation, and analytics. Differences among model classes may also matter: mixture-of-experts, dense open-weight, code-specialized, and local models represent different tradeoffs in capability, cost, deployment control, and output reliability [25, 26, 27, 28].

2.4. Ontology-Guided KC Extraction

The literature suggests a need for KC annotation methods that are scalable, interpretable, and aligned with stable curriculum models. Manual annotation provides interpretability but is costly. Parser-based and data-driven approaches reduce manual effort but may not fully capture pedagogical intent or may require additional interpretation. LLMs can support semantic annotation, but open-ended generation can produce imbalanced, irregular, or curriculum-misaligned labels.

Ontology-guided KC extraction addresses these limitations by combining expert-defined concept structure with scalable LLM-based annotation. In this framing, the ontology defines the valid KC space, while the LLM assigns programming exercises to concepts within that space. This preserves the role of human expertise in defining the domain model while using LLMs to reduce the cost.

This study builds on the CLAWS framework for ontology-guided concept extraction from programming learning content [15]. Whereas prior work established the feasibility of ontology-guided LLM annotation, the present study examines whether this annotation harness generalizes across different LLM deployment classes. LLM harness refer to the full annotation setup around the model: the ontology-constrained prompt, KC-window size, repeated annotation batches, score aggregation, thresholding, and output parser. A model that performs well in general natural-language tasks may still be unsuitable for educational annotation if it cannot follow the schema, maintain stable scores, or produce KC sets with an appropriate level of detail. This positions ontology-guided KC extraction as a human-directed form of human-AI collaboration for scalable annotation.

3. Methods

This study extends the CLAWS ontology-guided extraction of KC [15]. We use the same Python programming exercise dataset, expert KC annotations, and Python educational ontology. The dataset contains 262 interactive learning activities from introductory Python courses taught across multiple institutions. The candidate label space is defined by the Python educational ontology, which contains 128 classes, including 98 leaf-level concepts used as candidate KC [14]. Expert-authored KC labels serve as the gold standard for expert-agreement evaluation.

The annotation harness follows CLAWS: each exercise is submitted to an LLM together with candidate ontology concepts, and the model generates a brief relevance rationale then assigns a score from 0 to 5 for each candidate concept. The expected output is a structured CSV-style table.

A KC-window is the subset of candidate ontology concepts shown to the model in one prompt. For a KC-window size w , the leaf-level candidate concepts are divided into groups of size w , and each exercise is annotated once for each group. We evaluate $w = 5$, $w = 10$, and $w = 20$, as well as a no-window configuration in which the full candidate label space is provided in a single prompt. KC-windowing is

intended to reduce the number of candidate labels the model must judge at once, but it may also remove useful global context about the ontology. For this reason, we treat KC-window size as a calibration parameter rather than as a fixed best practice.

Each model configuration is run for three repeated batches when feasible. For each exercise–concept pair, repeated scores are aggregated using the modal score, with ties resolved toward the higher score. The aggregated scores are converted into predicted KC sets using thresholds $t \in \{3, 4, 5\}$. A concept is predicted as present when its aggregated score is greater than or equal to t . The threshold $t = 4$ is used as the primary comparison point. Following common five-level rating conventions, a score of 4 is typically interpreted as a good or strong relevance rating. Because LLMs may generate ordinal score scales differently, we treat threshold choice as another calibration parameter.

We selected models to represent common deployment choices for annotation pipelines: compact hosted models, large mixture-of-experts hosted models, efficient low-latency hosted models, large open-weight generalist models, small local generalist models, and code-specialized local models. GPT-4.1-mini represents a compact generalist API model, capturing a lower-cost hosted option with strong instruction-following and coding-oriented capabilities. DeepSeek-V3 represents a large Mixture-of-Experts (MoE) generalist API model; it has 671B total parameters, 37B active parameters per token, and was trained on 14.8T tokens, allowing us to test whether a high-capacity sparse model improves ontology-guided annotation over smaller or denser alternatives [25]. Claude Haiku 4.5 represents an efficient generalist API model, included to examine whether a speed- and cost-oriented hosted model can still produce reliable structured KC annotations. Llama 3.3 70B represents a large open-weight generalist model, providing a comparison point for non-proprietary models that offer more deployment control while retaining broad reasoning and code-understanding capability [26]. Mistral 7B represents a small instruction-tuned local model, included to test whether lightweight local deployment is sufficient for a structured ontology-scoring task. Qwen2.5-Coder 7B represents a code-specialized local model; it is built for programming-related tasks and is included to examine whether code specialization can compensate for smaller model scale in KC extraction [28]. Qwen2.5-Coder 7B was evaluated on a half-sampled subset of 132 exercises with single batch.

We first evaluate output compliance and batch stability. These diagnostics indicate whether poor downstream performance is caused by failures in producing stable and parseable outputs. Expert agreement is then evaluated by comparing each predicted KC set with the corresponding expert-label.

4. KC-Extraction Diagnostics

4.1. Output Compliance

Output compliance measures whether model responses could be parsed into the expected structured annotation format. We report the percentage of prompt responses that were successfully parsed into usable annotation files. We ran three batches for each model configuration; the reported value is the mean parse success rate across batches.

Table 1
Output compliance.

Model	$w = 5$	$w = 10$	$w = 20$	No window ($w = 98$)
GPT-4.1-mini	99.6%	99.6%	99.6%	99.6%
DeepSeek-V3	100.0%	100.0%	95.7%	100.0%
Claude Haiku 4.5	99.0%	98.8%	99.2%	99.5%
Llama 3.3 70B	96.2%	98.6%	99.4%	99.3%
Mistral 7B	44.2%	56.6%	41.2%	63.6%
Qwen2.5-Coder 7B	56.4%	62.9%	65.8%	61.0%

The API-based model classes generally produced highly parseable outputs. GPT-4.1-mini, DeepSeek-V3, Claude Haiku 4.5, and Llama 3.3 70B all exceeded 95.0% parse success in every reported KC-window

configuration, with most configurations near 100.0%. In contrast, the local model classes had much lower parse success. Mistral 7B parsed only 44.2% of $w = 5$ responses, and Qwen2.5-Coder 7B parsed 56.4% of $w = 5$ responses on the sampled subset.

4.2. Batch Stability

Batch stability measures whether repeated annotation runs produce similar predicted KC sets. We report two stability metrics. The first is all-batch intersection-over-union (IoU), which measures the overlap shared across all available batches. The second is mean pairwise Jaccard similarity (Mean J), which averages the Jaccard similarities across all pairs of batches. Higher values indicate more stable repeated annotations.

Table 2
Batch stability

Model (Class)		$w = 5$	$w = 10$	$w = 20$	No window ($w = 98$)
GPT-4.1-mini (Compact Generalist API)	IoU	.84	.82	.85	.49
	Mean J	.92	.91	.90	.71
DeepSeek-V3 (MoE Generalist API)	IoU	.93	.92	.78	.93
	Mean J	.95	.95	.85	.96
Claude Haiku 4.5 (Efficient Generalist API)	IoU	.60	.69	.64	.15
	Mean J	.71	.79	.76	.28
Llama 3.3 70B (Open-Weight Generalist API)	IoU	.43	.38	.24	.14
	Mean J	.57	.53	.39	.26

High output compliance does not necessarily imply stable annotation behavior, as shown in Table 2. DeepSeek-V3 was highly stable across most configurations, with mean pairwise Jaccard values of .95 for KC-window size $w = 5$, .95 for $w = 10$, and .96 for *no-window*. GPT-4.1-mini was also stable under KC-windowed prompting, with mean pairwise Jaccard values at least .90 for $w = 5$, $w = 10$, and $w = 20$, but was less stable in the *no-window*. Claude Haiku 4.5 and Llama 3.3 70B showed lower stability, especially in the *no-window*, suggesting that these model classes were more sensitive to large candidate-label contexts.

5. Expert Evaluation

This evaluation compares CLAWS-generated knowledge components (KC) against expert annotations. For each exercise, LLM-generated scores were converted into a predicted KC set by applying a threshold $t \in \{3, 4, 5\}$: a concept was treated as present when its score was greater than or equal to t . We report macro-averaged precision (P), recall (R), F1, and Jaccard similarity (J) across exercises. The average number of predicted concepts per exercise is also reported because prediction density affects the interpretation of precision and recall. The expert annotations contain an average of 13.41 concepts per exercise. We interpret each model as a representative case of a broader model class.

5.1. Aggregate Expert-Agreement Results

Table 3 reports the expert-agreement results across thresholds and KC-window sizes using mode aggregation with high-score tie breaking. The table shows that annotation quality varies by model class. This means that performance is not only a property of the base model; it is also affected by the surrounding annotation harness. In several configurations, macro F1 is lower than both precision and recall, indicating uneven precision–recall behavior across exercises.

Table 3Expert agreement for three-batch aggregate predictions by model, KC-window size w , and threshold t .

Model (Class)	t	KC window, $w = 5$					KC window, $w = 10$					KC window, $w = 20$					No KC window				
		P	R	F1	J	$M (SD)$	P	R	F1	J	$M (SD)$	P	R	F1	J	$M (SD)$	P	R	F1	J	$M (SD)$
GPT-4.1-mini (Compact Generalist API)	3	.82	.96	.88	.80	15.8 (5.74)	.72	.96	.82	.71	17.7 (6.48)	.78	.96	.86	.76	16.3 (5.70)	.84	.92	.88	.80	14.3 (5.76)
	4	.86	.96	.90	.83	15.0 (5.73)	.76	.96	.84	.74	17.0 (6.42)	.81	.96	.88	.79	15.7 (5.59)	.86	.92	.88	.81	14.1 (5.70)
	5	.89	.90	.89	.81	13.5 (5.31)	.81	.87	.83	.73	14.5 (6.21)	.87	.91	.89	.81	14.0 (5.43)	.88	.89	.88	.81	13.3 (5.63)
DeepSeek-V3 671B total; 37B active (MoE Generalist API)	3	.78	.96	.86	.76	16.3 (5.94)	.81	.96	.87	.79	15.6 (5.61)	.80	.96	.87	.78	15.9 (5.73)	.91	.96	.93	.88	13.9 (4.98)
	4	.81	.95	.87	.79	15.6 (5.91)	.85	.95	.89	.81	14.9 (5.49)	.82	.95	.88	.80	15.4 (5.61)	.92	.95	.93	.88	13.7 (5.00)
	5	.85	.93	.88	.80	14.6 (5.43)	.87	.93	.89	.82	14.1 (5.02)	.85	.94	.89	.82	14.8 (5.40)	.92	.93	.92	.88	13.3 (5.01)
Claude Haiku 4.5 (Efficient Generalist API)	3	.82	.83	.80	.70	13.9 (7.11)	.83	.79	.78	.70	12.6 (6.85)	.79	.80	.78	.70	12.6 (6.99)	.75	.77	.76	.69	12.1 (6.99)
	4	.83	.80	.79	.69	13.0 (6.66)	.85	.75	.77	.68	11.5 (6.18)	.81	.75	.76	.68	11.4 (6.37)	.76	.71	.73	.65	10.9 (6.41)
	5	.88	.57	.67	.54	8.60 (4.72)	.89	.55	.66	.53	7.99 (4.40)	.84	.55	.65	.52	8.00 (4.56)	.78	.52	.62	.50	7.73 (4.85)
Llama 3.3 70B (Open-Weight Generalist API)	3	.58	.88	.69	.54	20.7 (8.12)	.59	.90	.71	.56	20.5 (7.54)	.60	.84	.69	.54	18.9 (7.62)	.70	.67	.67	.52	12.8 (5.52)
	4	.62	.87	.71	.56	19.0 (7.24)	.63	.89	.73	.58	18.9 (6.70)	.63	.82	.70	.55	17.5 (6.92)	.71	.65	.67	.52	12.1 (5.11)
	5	.71	.76	.72	.58	14.4 (5.24)	.71	.78	.73	.60	14.6 (5.21)	.70	.70	.69	.54	13.3 (5.19)	.73	.55	.61	.46	9.79 (4.17)
Mistral 7B (Small Generalist Local)	3	.095	.064	.069	.039	6.17 (4.06)	.079	.032	.040	.023	2.71 (2.80)	.063	.022	.031	.018	1.78 (2.39)	.011	.004	.004	.003	1.86 (2.98)
	4	.10	.062	.071	.040	5.48 (3.62)	.078	.029	.038	.021	2.39 (2.47)	.059	.020	.028	.016	1.60 (2.12)	.004	.003	.003	.002	1.18 (2.12)
	5	.12	.053	.067	.038	3.86 (2.83)	.075	.020	.028	.016	1.61 (1.63)	.057	.015	.023	.013	1.18 (1.67)	.006	.003	.003	.002	0.74 (1.59)
Qwen2.5-Coder 7B [†] (Code Specialist Local)	3	.37	.38	.36	.23	14.8 (7.71)	.37	.33	.33	.21	12.8 (7.10)	.37	.21	.25	.15	8.89 (6.01)	.20	.068	.095	.060	2.18 (2.60)
	4	.42	.25	.29	.18	–	.39	.20	.25	.16	–	.37	.13	.18	.11	–	.19	.026	.044	.025	–
	5	.15	.022	.036	.021	–	.13	.028	.043	.026	–	.092	.019	.029	.017	–	0	0	0	0	–

P = precision; R = recall; F1 = F1 score; J = Jaccard similarity.

The number of KC predicted per exercise reported as the mean and standard deviation, $M (SD)$.

At threshold $t = 4$, DeepSeek-V3, the MoE Generalist API class, achieved the strongest expert-agreement result, with macro F1 = .93 and J = .88 in the no-window configuration. This configuration predicted 13.7 concepts per exercise on average, which is close to the expert average of 13.41.

The GPT-4.1-mini, Compact Generalist API class, also performed strongly with KC-window size $w = 5$, reaching macro F1 = .90 and J = .83. However, it predicted slightly more concepts per exercise on average, suggesting a more recall-oriented behavior. Claude Haiku 4.5 followed the small KC-window pattern, performing best with $w = 5$, while Llama 3.3 70B performed best with $w = 10$ at this threshold. In contrast, DeepSeek-V3 performed best without KC-windowing. These results suggest that KC-windowing is a model-class-dependent mechanism rather than a universally optimal configuration.

We therefore avoid interpreting the best-performing model as a permanent recommendation. Instead, the result shows that model-class behavior interacts with the KC-window and threshold settings, and that future models should be calibrated rather than assumed to inherit the same optimal configuration.

5.2. Prediction Density and Threshold Effects

All models were evaluated against the same expert annotations within a given evaluation set and were prompted with the same candidate KC labels within a given KC-window size. What differs is how many labels each model assigns scores at or above the threshold. Any label below the threshold is treated as not predicted. Thus, recall is directly affected by the number of labels that pass the threshold. Models that predict more labels tend to recover more gold-standard concepts, but may get more false positives.

The average number of predicted labels per exercise varied across models for predictions aggregated across three batches at KC-window size $w = 5$ and threshold $t = 3$. The gold-standard annotations contain 13.4 KC per exercise on average. GPT-4.1-mini and DeepSeek-V3 predict more labels than the gold standard, with 15.8 and 16.3 mean predicted KC per exercise, respectively, which helps explain their high recall. Claude Haiku 4.5 predicts 13.9 labels per exercise, close to the gold-standard density. Llama 3.3 70B is much more expansive, predicting 20.7 labels per exercise, which supports high recall but can reduce precision. Mistral 7B is much sparser, predicting only 6.17 labels per exercise, and therefore lowering recall.

A model with high recall may be recovering many correct KC because it predicts more labels overall. A model with high precision may be predicting fewer labels, but this can cost missing many gold-standard KC. For ontology-guided KC extraction, the ideal behavior is to produce a prediction density close to the expert annotation pattern while maintaining high precision and recall.

Threshold sensitivity further supports the class-dependent interpretation. DeepSeek-V3 was relatively stable across thresholds in the no-window setting, with aggregate macro F1 changing only slightly from .93 at $t = 3$ to .93 at $t = 4$ and .92 at $t = 5$. Claude Haiku 4.5 was more threshold-sensitive: its best

aggregate result occurred at $t = 3$ with macro F1 = .80, while increasing the threshold to $t = 5$ reduced its best macro F1 to .67. Llama 3.3 70B showed a different pattern, with its strongest aggregate result at $w = 10$ and $t = 5$ (macro F1 = .73). The numerical score scale is not used identically across LLMs. Consequently, the threshold should be viewed as a calibration parameter rather than a fixed property.

In practice, this means that a small validation set with expert labels is needed when changing model classes. The threshold should be selected not only by maximizing F1, but by checking whether prediction density is close to the expert annotation pattern and appropriate for the intended downstream use.

5.3. Local LLMs and Pipeline Readiness

The lower expert-agreement scores of Mistral 7B and Qwen2.5-Coder 7B are partly driven by output non-compliance rather than only by semantic disagreement with the expert annotations. The evaluation pipeline expects each model to generate structured CSV output with valid exercise identifiers, ontology labels, and numeric scores. When a model fails to generate a valid row, omits required labels, produces malformed CSV, or hallucinates labels outside the ontology, those missing or invalid predictions cannot be counted as positive KC. The current CLAWS pipeline is more suitable for models that can reliably follow the structured-output format without additional orchestration. Consequently, the local-model results are best understood as readiness under the current pipeline. Local models require stronger output-control mechanisms, such as constrained decoding, schema validation, automatic output repair, few-shot formatting examples, or fine-tuning.

6. Discussion and Conclusion

This study evaluated whether the CLAWS ontology-guided KC extraction generalizes across different classes of LLMs. The results show that the approach can produce high-quality KC annotations, but performance is uneven across model classes. Hosted generalist models were the most reliable overall, while smaller local models struggled mainly with structured-output compliance and sparse or incomplete KC predictions. The main implication is that ontology-guided KC extraction should be treated as a calibrated annotation pipeline. KC-window size, score threshold, prediction density, aggregation, and output compliance all affect performance. These factors are likely to remain relevant even as specific LLMs change, because future models will still need to interpret the ontology, score candidate concepts, follow a structured schema, and produce KC sets at a useful level of detail.

The results also show that KC-windowing should not be treated as a fixed best practice. In the earlier CLAWS study, smaller ontology KC-windows improved performance by reducing the number of candidate concepts shown at once. In this broader model-class evaluation, the best KC-window configuration varied by model. Some models benefited from small KC-windows, while others performed better when given the full label space. This suggests that both KC-window size and threshold should be treated as calibration parameters that depend on the model and the downstream use case.

This study has several limitations. The evaluation used one Python ontology and one collection of Python programming activities, so the results may not transfer directly to other programming languages, domains, or ontology structures and granularity. Each model represents a broader deployment class, but one model cannot fully characterize that class. Finally, this study evaluates annotation quality against expert labels, but not the effect of the resulting annotations on student learning, knowledge tracing, or recommendations in a deployed adaptive system.

Ontology-guided LLM annotation offers a scalable and human-directed approach to human-AI collaboration for KC extraction in programming education. A useful LLM harness should follow the ontology, produce stable structured output, assign concepts consistently, and generate KC sets with an appropriate level of detail. Future work should examine hierarchical annotation, model-specific output-control strategies, and the use in deployed adaptive programming systems.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant HCC-2426839.

Declaration on Generative AI

During the preparation of this work, the author(s) used GenAI for: Paraphrase and reword. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take full responsibility for the publication's content.

References

- [1] H. Cen, K. R. Koedinger, B. Junker, Learning factors analysis: A general method for cognitive model evaluation and improvement, in: *Intelligent Tutoring Systems*, volume 4053 of *Lecture Notes in Computer Science*, Springer, 2006, pp. 164–175. doi:10.1007/11774303_17.
- [2] R. E. Clark, D. F. Feldon, J. J. G. van Merriënboer, K. A. Yates, S. Early, Cognitive task analysis, in: J. M. Spector, M. D. Merrill, J. J. G. van Merriënboer, M. P. Driscoll (Eds.), *Handbook of Research on Educational Communications and Technology*, 3 ed., Macmillan/Gale, New York, 2008, pp. 577–593.
- [3] A. Hicks, K. Akhuseyinoglu, C. Shaffer, P. Brusilovsky, Live catalog of smart learning objects for computer science education, in: *Sixth SPLICE Workshop "Building an Infrastructure for Computer Science Education Research and Practice at Scale"* at ACM Learning at Scale 2020, 2020. URL: https://csssplice.github.io/LAS20/proc/SPLICE_2020_LS_paper_7.pdf.
- [4] R. Hosseini, P. Brusilovsky, Javaparser: A fine-grain concept indexing tool for java problems, in: *Proceedings of the First Workshop on AI-Supported Education for Computer Science*, volume 1009 of *CEUR Workshop Proceedings*, University of Pittsburgh, 2013, pp. 60–63.
- [5] Y. Shi, R. Schmucker, M. Chi, T. Barnes, T. W. Price, KC-Finder: Automated knowledge component discovery for programming problems, in: *Proceedings of the 16th International Conference on Educational Data Mining*, International Educational Data Mining Society, 2023, pp. 28–39. doi:10.5281/zenodo.8115671.
- [6] Y. Shi, R. Schmucker, K. Tran, J. Bacher, K. R. Koedinger, T. W. Price, M. Chi, T. Barnes, The knowledge component attribution problem for programming: Methods and tradeoffs with limited labeled data, *Journal of Educational Data Mining* 16 (2024) 1–33. doi:10.5281/zenodo.10844782.
- [7] S. Moore, R. Schmucker, T. Mitchell, J. Stamper, Automated generation and tagging of knowledge components from multiple-choice questions, in: *Proceedings of the Eleventh ACM Conference on Learning @ Scale, L@S '24*, Association for Computing Machinery, 2024, pp. 122–133. doi:10.1145/3657604.3662030.
- [8] H. Moon, R. L. Davis, S. P. Neshaei, P. Dillenbourg, Using large multimodal models to extract knowledge components for knowledge tracing from multimedia question information, in: *Proceedings of the 18th International Conference on Educational Data Mining*, International Educational Data Mining Society, 2025, pp. 342–353. doi:10.5281/zenodo.15870171.
- [9] Y. Özyurt, S. Feuerriegel, M. Sachan, Automated knowledge concept annotation and question representation learning for knowledge tracing, 2024. doi:10.48550/arXiv.2410.01727. arXiv:2410.01727.
- [10] Y. Wei, P. Carvalho, J. Stamper, Kcluster: An LLM-based clustering approach to knowledge component discovery, in: *Proceedings of the 18th International Conference on Educational Data Mining*, International Educational Data Mining Society, 2025, pp. 228–240. doi:10.5281/zenodo.15870197.
- [11] S. Sosnovsky, I.-H. Hsiao, P. Brusilovsky, Adaptation “in the wild”: Ontology-based personalization of open-corpus learning material, in: *21st Century Learning for 21st Century Skills*,

- volume 7563 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 425–431. doi:10.1007/978-3-642-33263-0_38.
- [12] A. Nguyen, L. Gardner, D. Sheridan, Building an ontology of learning analytics, in: *Proceedings of the Pacific Asia Conference on Information Systems*, 2018. URL: <https://aisel.aisnet.org/pacis2018/158>, paper 158.
 - [13] S. Sosnovsky, T. Gavrilova, Development of educational ontology for c-programming, in: *Proceedings of the International Conference Knowledge-Dialogue-Solution (KDS-2005)*, Varna, Bulgaria, 2005, pp. 127–131.
 - [14] A. Micheli, Development and Evaluation of the Python Educational Ontology: A Data-Mining Experiment, Master’s thesis, Utrecht University, 2025. URL: <https://studenttheses.uu.nl/handle/20.500.12932/49904>.
 - [15] R. A. Hendrawan, R. Sampaio de Alencar, A. Micheli, P. Brusilovsky, J. Barria-Pineda, S. Sosnovsky, Data-driven evaluation of LLM-based ontology concept extraction from programming learning content, in: *Proceedings of the 16th International Conference on Learning Analytics and Knowledge, LAK ’26*, Association for Computing Machinery, 2026, pp. 526–535. doi:10.1145/3785022.3785106.
 - [16] Y. Shi, M. Chi, T. Barnes, T. W. Price, Code-DKT: A code-based knowledge tracing model for programming tasks, in: *Proceedings of the 15th International Conference on Educational Data Mining*, International Educational Data Mining Society, 2022, pp. 50–61. doi:10.5281/zenodo.6853105.
 - [17] R. Sampaio de Alencar, M. A. Demirtas, A. Soukarjya Saha, Y. Shi, P. Brusilovsky, Integrating expert knowledge with automated knowledge component extraction for student modeling, in: *Proceedings of the 33rd ACM Conference on User Modeling, Adaptation and Personalization, UMAP ’25*, Association for Computing Machinery, 2025, pp. 307–312. doi:10.1145/3699682.3728348.
 - [18] D. B. Guruge, R. Kadel, S. J. Halder, The state of the art in methodologies of course recommender systems: A review of recent research, *Data* 6 (2021) 18. doi:10.3390/data6020018.
 - [19] W. Villegas-Ch, J. García-Ortiz, Enhancing learning personalization in educational environments through ontology-based knowledge representation, *Computers* 12 (2023) 199. doi:10.3390/computers12100199.
 - [20] P. Liang, R. Bommasani, T. Lee, D. Tsipras, D. Soylu, M. Yasunaga, Y. Zhang, D. Narayanan, Y. Wu, A. Kumar, B. Newman, B. Yuan, B. Yan, C. Zhang, C. Cosgrove, C. D. Manning, C. Ré, D. Acosta-Navas, D. A. Hudson, E. Zelikman, et al., Holistic evaluation of language models, *Transactions on Machine Learning Research* (2023). URL: <https://openreview.net/forum?id=iO4LZibEqW>. doi:10.48550/arXiv.2211.09110. arXiv:2211.09110.
 - [21] W.-L. Chiang, L. Zheng, Y. Sheng, A. N. Angelopoulos, T. Li, D. Li, H. Zhang, B. Zhu, M. I. Jordan, J. E. Gonzalez, I. Stoica, Chatbot arena: An open platform for evaluating llms by human preference, in: *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, PMLR, 2024, pp. 8359–8388. URL: <https://proceedings.mlr.press/v235/chiang24b.html>.
 - [22] T. Li, W.-L. Chiang, E. Frick, L. Dunlap, T. Wu, B. Zhu, J. E. Gonzalez, I. Stoica, From crowdsourced data to high-quality benchmarks: Arena-hard and benchbuilder pipeline, *arXiv preprint arXiv:2406.11939* (2024). URL: <https://arxiv.org/abs/2406.11939>. doi:10.48550/arXiv.2406.11939. arXiv:2406.11939.
 - [23] S. Geng, H. Cooper, M. Moskal, S. Jenkins, J. Berman, N. Ranchin, R. West, E. Horvitz, H. Nori, Generating structured outputs from language models: Benchmark and studies, 2025. doi:10.48550/arXiv.2501.10868. arXiv:2501.10868.
 - [24] D. Y.-B. Wang, Z. Shen, S. S. Mishra, Z. Xu, Y. Teng, H. Ding, Slot: Structuring the output of large language models, 2025. doi:10.48550/arXiv.2505.04016. arXiv:2505.04016.
 - [25] DeepSeek-AI, A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, D. Dai, D. Guo, D. Yang, D. Chen, D. Ji, E. Li, F. Lin, F. Dai, F. Luo, et al., Deepseek-v3 technical report, *arXiv preprint arXiv:2412.19437* (2024). URL: <https://arxiv.org/abs/2412.19437>. doi:10.48550/arXiv.2412.19437. arXiv:2412.19437.

- [26] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan, A. Yang, et al., The llama 3 herd of models, arXiv preprint arXiv:2407.21783 (2024). URL: <https://arxiv.org/abs/2407.21783>. doi:10.48550/arXiv.2407.21783. arXiv:2407.21783.
- [27] B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, R. Sauvestre, T. Remez, J. Rapin, A. Kozhevnikov, I. Evtimov, J. Bitton, M. Bhatt, C. C. Ferrer, A. Grattafiori, W. Xiong, A. Défossez, J. Copet, F. Azhar, H. Touvron, L. Martin, N. Usunier, T. Scialom, G. Synnaeve, Code llama: Open foundation models for code, arXiv preprint arXiv:2308.12950 (2023). URL: <https://arxiv.org/abs/2308.12950>. doi:10.48550/arXiv.2308.12950. arXiv:2308.12950.
- [28] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Lu, K. Dang, Y. Fan, Y. Zhang, A. Yang, R. Men, F. Huang, B. Zheng, Y. Miao, S. Quan, Y. Feng, X. Ren, X. Ren, J. Zhou, J. Lin, Qwen2.5-coder technical report, arXiv preprint arXiv:2409.12186 (2024). URL: <https://arxiv.org/abs/2409.12186>. doi:10.48550/arXiv.2409.12186. arXiv:2409.12186.