CodeSensei: Turning Passive Behavioral Sensing into Adaptive Programming Education with Agentic AI

Zeyu Luo*
zeyuluo@ust.hk
The Hong Kong University of Science and Technology
Hong Kong SAR, China

Yuchao Wang*
ywangtc@connect.ust.hk
Hong Kong University of Science and Technology
Hong Kong SAR, China

Qijia Shao
qijiashao@ust.hk
The Hong Kong University of Science and Technology
Hong Kong SAR, China

Abstract

Programming learners differ widely in prior knowledge and the effort required to solve coding tasks, yet most practice tools rely on fixed sequences or adapt only based on correctness. We present CodeSensei, an IDE-integrated system that turns passive behavioral sensing into adaptive programming education with agentic AI. CodeSensei uses keyboard and mouse traces as unobtrusive sensing streams and organizes adaptation as an interpret–plan–act–adapt loop. A Behavior-Semantic Interpretation Agent learns temporal representations from interaction traces, infers cognitive load, and uses LLM-based semantic anchoring to map effort to programming knowledge components. An Adaptive Practice Agent then updates a load-aware learner profile, plans practice priorities, and uses retrieval-augmented LLM generation with automated validation and repair to produce personalized exercises. In preliminary studies, CodeSensei achieved 88.15% F1 for binary cognitive-load classification, received positive usability feedback, and showed exploratory evidence of short-term learning benefits. CodeSensei demonstrates a potential toward embedded learning companions that use agentic AI to interpret interactive learning traces and deliver timely, personalized support across digital learning environments.

CCS Concepts

• **Applied computing** → **Computer-assisted instruction**; • **Human-centered computing** → **Interactive systems and tools**; • **Social and professional topics** → **Computing education**.

Keywords

Cognitive Load, Adaptive Learning, Programming Exercise, LLMs

ACM Reference Format:
Zeyu Luo, Yuchao Wang, and Qijia Shao. 2026. CodeSensei: Turning Passive Behavioral Sensing into Adaptive Programming Education with Agentic AI. In *Companion of the 2026 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp Companion '26), October 11–15, 2026, Shanghai, China.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3798063.3839170

*Both authors contributed equally to this research.

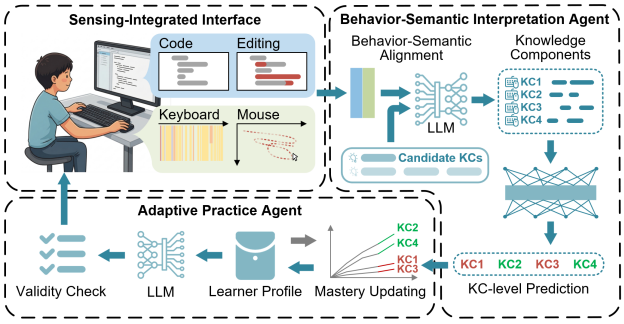

Figure 1: Overall framework of CodeSensei.

1 Introduction

Ubiquitous sensing is increasingly moving beyond passive measurement toward agentic systems that can perceive context, interpret uncertain signals, reason over goals, and act through feedback-driven intervention. Education is a natural domain for this shift: learners continuously generate rich behavioral traces while working in digital tools, yet most learning systems still respond primarily to explicit inputs or final outcomes. Programming practice is a representative example. Learners type, pause, delete, navigate, revise, and test code inside integrated development environments (IDEs), but most practice systems rely on fixed exercise sequences, adapt only from correctness, or recommend only at the course level [3, 10, 25]. Two learners may both pass a test while experiencing very different levels of effort; conversely, two learners may fail for different underlying knowledge components (KCs). Effective adaptive programming education therefore requires moving beyond outcome-only adaptation toward sensing and acting on process-level effort during authentic coding.

Recent LLM-based programming education systems can generate explanations, feedback, and practice tasks, but they are often driven by prompts, submitted code, or predefined curricula rather than continuous behavioral sensing [20, 22, 39]. Meanwhile, behavioral sensing research has shown that keyboard, mouse, and interaction dynamics can reveal cognitive effort, but such signals are often used for offline prediction or visualization rather than closed-loop educational intervention [17, 36, 50]. This creates a gap between sensing and action: behavioral traces can indicate that a learner is struggling, but adaptive systems still need to interpret what the learner is struggling with, decide what should be practiced next, and deliver an appropriate intervention. We argue that agentic AI provides a useful design frame for this gap, not by replacing teachers with fully autonomous tutors, but by embedding bounded agents into learning workflows that transform passive sensing into timely adaptive support.

This work is licensed under a Creative Commons Attribution 4.0 International License.
UbiComp Companion '26, Shanghai, China
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2533-3/2026/10
https://doi.org/10.1145/3798063.3839170

We present *CodeSensei*, an IDE-integrated agentic AI system for adaptive programming education (Figure 1). CodeSensei turns passive behavioral sensing into educational intervention through an interpret–plan–act–adapt loop. A sensing-integrated IDE interface unobtrusively captures keyboard and mouse traces, pauses, code snapshots, and submissions during routine programming practice. A *Behavior-Semantic Interpretation Agent* then interprets these low-level traces by learning temporal behavioral representations, inferring cognitive load, and anchoring effort to programming KCs through LLM-based semantic alignment. An *Adaptive Practice Agent* uses interpreted learner-state signals to update a load-aware learner profile, plan practice priorities, and act by generating validated programming exercises through a retrieval-augmented LLM pipeline. The learner's subsequent behavior and performance are fed back into the profile, enabling continuous adaptation across exercises.

We conducted a mixed-methods evaluation to examine the robustness of our in-situ sensing pipeline and the feasibility of the closed-loop adaptation process, combining a long-term field deployment (N=11) and a short-term controlled study (N=24). In the field deployment, learners integrated CodeSensei into their daily programming routines for over a week, where qualitative feedback highlighted perceived adaptivity, knowledge coverage, pacing, exercise quality, and usability. The controlled study further provides exploratory evidence that the adaptive condition is associated with greater perceived improvement and higher learning gains compared to a fixed baseline. Together, these results suggest the feasibility of using agentic AI to connect passive behavioral sensing with adaptive educational intervention in programming workflows.
In summary, this paper makes the following primary contributions:

- **Agentic sensing-to-intervention loop.** We present CodeSensei, an IDE-integrated system that turns passive keyboard/mouse sensing into adaptive programming intervention through a closed interpret–plan–act–adapt loop.
- **Behavior-Semantic Interpretation Agent.** We introduce an interpretation pipeline that learns temporal representations from interaction traces, infers cognitive load, and uses LLM-based semantic anchoring to map effort to programming KCs.
- **Adaptive Practice Agent.** We design a profile-driven intervention agent that updates KC-level mastery using both correctness and cognitive load, plans practice priorities, and uses retrieval-augmented LLM generation with validation and repair to deliver personalized programming exercises.

2 Related Work

LLMs have demonstrated strong capabilities in programming problem solving, exercise generation, and code explanation [1, 15, 39]; prior work has also reframed programming tasks as prompt engineering exercises [13]. At the same time, directly programming with LLMs may negatively affect learning outcomes [27, 34], motivating systems that use LLMs to provide more structured educational support. Recent LLM-powered programming education tools have explored personalized programming tasks and feedback [24], adaptive Parsons puzzles [20], autonomous programming learning for young children [7], guarded conceptual feedback and annotations without revealing full code [22], learner–LLM co-decomposition for tailored scaffolding [29], teachable-agent systems [21, 28], and LLM-supported project-based learning [54, 55]. More broadly, adaptive learning systems have used course-level recommendations based on machine learning and knowledge graphs [33, 42], dialog-based tutors and domain-specific adaptation [19, 38, 44, 45, 47], and process mining to support personalization by uncovering detailed learning patterns [4, 37, 48]. Cognitive load provides another basis for adaptation and has been measured through subjective surveys [9, 55], physiological signals [5, 6, 12, 40, 49, 51], and behavioral measurements [8, 11, 16, 36]. Physiological methods provide rich signals but often require additional instrumentation, while behavioral approaches can be integrated more naturally into authentic task settings. In programming education, CodeStream visualizes keystroke histories as a proxy for student effort [17], showing the potential relevance of programming-process traces for understanding learners' work. Building on these lines of work, CodeSensei combines LLM-powered exercise generation with behavioral cognitive-load sensing: it estimates cognitive load from unobtrusive IDE keyboard and mouse interactions during programming practice, updates a persistent learner profile across exercises, and generates subsequent programming exercises based on learners' knowledge progression and their cognitive load in previous exercises.

3 System Overview and Challenges

As illustrated in Figure 1, the system is realized as a closed-loop architecture consisting of a sensing-integrated interface and two specialized agents that together enable continuous, effort-aware adaptation inside authentic coding workflows:

Sensing-Integrated Interface. CodeSensei is implemented as a PyCharm plugin that unobtrusively captures keyboard and mouse interactions during programming. It also serves as the intervention interface, delivering exercises and collecting feedback, allowing users to interact with the system within normal coding workflow.

Behavior-Semantic Interpretation Agent. Serving as the system's perception engine, this agent maps raw behavioral interaction traces to high-level code semantics. By analyzing temporal representations from the interaction data and aligning them with KCs via an LLM, it infers not only the intensity of cognitive load but also which programming concepts are associated with that effort.

Adaptive Practice Agent. Acting as the decision making and execution engine, this agent closes the learning loop. It maintains a learner profile by integrating inferred cognitive load with historical performance at the KC-level, utilizes retrieval-augmented LLM generation, paired with deterministic validity checks, to synthesize and deploy personalized, executable exercises.

We adopt a bounded agentic workflow rather than open-ended autonomy because such boundedness is particularly important in learner-facing systems, as it improves decision traceability and couples generation with deterministic quality assurance. Together, they form an adaptive practicing loop where interactions inform cognitive load estimation and profile updates, guiding the personalized exercises generation for subsequent practicing.

Challenges. Developing CodeSensei as an agentic programming-practice system introduces three key technical challenges. First, **inferring cognitive load from indirect behavioral signals** is

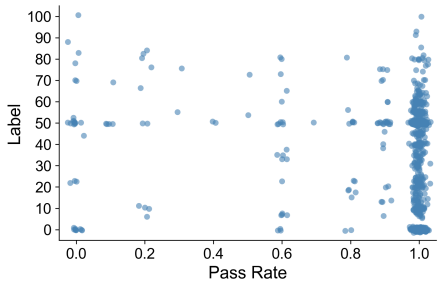

Figure 2: Relation between pass rates and self-reported cognitive load labels.

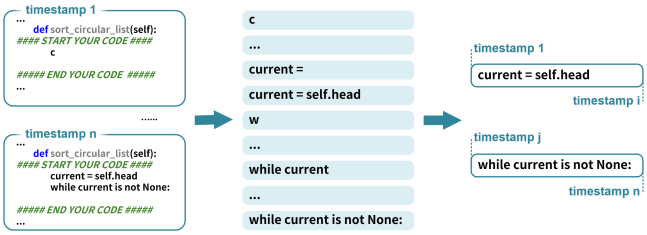

Figure 3: An example mapping between code lines and timestamps. From left to right: (1) Snapshots at different timestamps. (2) Tracked modifications. (3) Meaningful edited lines with mapped timestamps for subsequent processing.

difficult because keyboard and mouse interactions provide noisy, semantically unannotated traces; the system must therefore learn meaningful temporal patterns and align them with code semantics and KCs. Second, **constructing an interpretable learner profile** is non-trivial because programming lacks a universally accepted fine-grained KC taxonomy, and transient cognitive-effort signals must be integrated into a stable, long-term model of mastery. Third, **closing the loop with adaptive content generation** requires translating a dynamic learner profile into exercises that are relevant, appropriately difficult, diverse, and executable, while remaining non-redundant and pedagogically coherent in an automated pipeline. These challenges motivate the two-agent architecture detailed in Sec. 4 and Sec. 5.

4 Behavior-Semantic Interpretation Agent

This section presents the Behavior-Semantic Interpretation Agent, which transforms keyboard–mouse traces and code changes into KC-level cognitive-effort evidence through temporal behavior encoding and semantic anchoring.

4.1 Cognitive Load Modeling

Grounded in Cognitive Load Theory [2, 23, 43], we model learners' overall perceived cognitive load during code practice rather than distinguishing fine-grained cognitive-load subtypes [14, 16]. We formulate cognitive-effort interpretation as a mapping from temporal keyboard–mouse traces to minute-level load estimates. This process-oriented formulation treats both active interaction patterns and pauses between interactions as potentially meaningful evidence of learners' cognitive effort.

For model development, ground-truth cognitive load was collected through minute-level self-reports, which provide supervision during data collection but are not required during deployment, when the agent infers load directly from interaction traces. The resulting minute-level estimates are subsequently attributed and aggregated at the KC level.

4.2 Trace and Ground-Truth Alignment

Trace Construction. CodeSensei passively records timestamped keyboard and mouse events within the IDE, including event type, cursor position, optional content, and timestamp. From these logs, the agent computes nine features capturing editing, navigation, mouse activity, and temporal dynamics. Event timing is preserved to model active typing, navigation, mouse movement, and pauses as part of the same behavioral stream while maintaining ecological validity for downstream modeling.

Ground-Truth Alignment. At the data collection stage, participants additionally reported their perceived cognitive load using a slider in $[0, 100]$ with midpoint 50 every minute. Reports below 50 are labeled as *low* load and those at or above 50 as *high* load, and are aligned with behavioral traces by timestamp. Figure 2 shows that similar task outcomes can correspond to substantially different perceived load, motivating behavioral sensing as a complementary diagnostic signal.

4.3 Behavioral State Encoding

Temporal State Construction. Programming behavior is irregular: active keyboard and mouse events are interleaved with pauses that may reflect reading, planning, or debugging. The agent therefore preserves pauses as part of the behavioral sequence rather than treating them as missing data. To construct a temporal state representation, raw events and derived features are aggregated into 5-second feature vectors, and sequences are constructed using a 60-second sliding window with a 5-second step. This produces inputs $X \in \mathbb{R}^{T \times D}$, where $D = 9$ denotes the nine behavioral features. Each feature dimension is standardized using z-score normalization based on training-set statistics, which are reused during inference.

Self-Supervised State Encoder. The agent uses TS2Vec [53] as a self-supervised encoder to learn general-purpose behavioral states from unlabeled keyboard–mouse traces. Unlike supervised sequence models that require dense labels, TS2Vec leverages a hierarchical contrastive learning objective over *two augmented views* of the same time series. It produces contextual representations along the temporal axis and applies contrastive learning at multiple granularities: representations of corresponding temporal regions across the two views are treated as positives, while representations from mismatched regions and other sequences are treated as negatives. This hierarchical formulation encourages the agent's internal state to capture both local interaction motifs, such as cursor jumps and backspacing, and longer-range transitions among typing, mouse movement, and pauses.

4.4 Semantic Anchoring Toolchain

Behavioral states reveal *when* effort occurs, but adaptation also requires identifying *what* programming concept the effort pertains

to. We therefore anchor behavioral states to timestamped code edits and then to pedagogically meaningful KCs.

Temporal Anchoring of Code Edits. To contextualize behavioral windows, the plugin records timestamped code snapshots. After each session, the agent computes sequential diffs and identifies *meaningful edited lines*. An edit is considered meaningful when the Levenshtein distance between adjacent versions exceeds a threshold (e.g., > 5) and the newer version does not merely contain the older one. This produces edited code segments with associated time spans, linking interaction sequences to code evolution (Figure 3).

Anchoring Edited Code to KCs. To attribute effort to learning content, the agent maps meaningful edited segments to a fixed offline KC inventory (Sec. 5). Because programming concepts may be implicit, distributed, or intertwined with control and data flow, this step is formulated as semantic alignment rather than lexical matching. An LLM assigns one or more KCs to each edited segment based on the code context and KC inventory. The resulting edit–KC links define the semantic targets over which behavioral effort can be interpreted and aggregated.

We also evaluated KC anchoring on 51 sessions. Across 10 independent mapping runs, the assignments showed high run-to-run consistency. Compared with expert-annotated KC sequences, agreement exceeded 0.95 for most sessions, with a minimum of 0.85.

KC-Level Effort Attribution. For each KC, the agent aggregates its active edit intervals and associates them with the corresponding cognitive-load evidence. During model development and validation, KC-level cognitive load is defined as the *maximum* self-reported load over those intervals, emphasizing peak difficulty episodes for targeted remediation. During deployment, the same interval structure supports attribution of inferred load probabilities to KCs, allowing the system to translate raw behavioral traces into interpretable concept-level effort signals.

4.5 Cognitive-Effort Interpretation Module

Finally, the agent converts each encoded behavioral window into the learner's minute-wise perceived load. It serves as a cognitive-effort interpretation module that transforms the agent's internal behavioral state into an explicit cognitive-effort judgment used by Adaptive Practice Agent. Specifically, after standardization with training-set statistics, X is encoded by the pretrained TS2Vec model into $F \in \mathbb{R}^{T \times C}$, where $C = 128$. Temporal mean pooling is then applied over the pretrained TS2Vec representations, and a lightweight classification head is used for efficient IDE deployment, with two hidden layers of 128 and 64 units followed by ReLU activations and a sigmoid output layer.

5 Adaptive Practice Agent

While the Behavior-Semantic Interpretation Agent handles in-situ sensing and KC-level cognitive load inference, the Adaptive Practice Agent closes the loop by learner modeling, priority-based scheduling, and adaptive content generation.

5.1 Learner State and Memory

The agent maintains a fixed 141-KC taxonomy (92 fundamental programming concepts and 49 data structure operations) as its

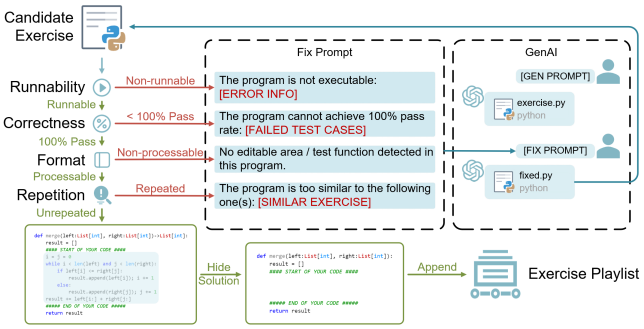

Figure 4: Validation-and-repair toolchain invoked by the Adaptive Practice Agent.

semantic state space. This taxonomy is constructed offline from established introductory Python learning materials [18, 30, 35, 41] and serves as the common representation for effort attribution, mastery tracking, and adaptive scheduling. For each KC, the agent tracks a scalar mastery score and updates it after every assessed exercise using the following load-aware rule:

$$S_{\text{new}} = S_{\text{old}} + k \times P \times F_{\text{CL}}, \tag{1}$$

where $k = 30$, $P \in [0, 1]$ is the pass rate of the exercise attributed to that KC, and F_{CL} is the cognitive-load factor ($F_{\text{CL}} = 1.0$ for low load and $F_{\text{CL}} = 0.3$ for high load). This rule down-weights mastery gains achieved under high cognitive effort. For exercises targeting multiple KCs, the update is applied independently to each associated KC using the same observed pass rate P.

In addition, the agent keeps a learner-specific practice memory that records previously practiced exercises and corresponding learner responses. It supports repetition avoidance, coverage tracking, and retrieval-time diversification during exercise generation.

5.2 Priority Planning

Mastery scores are discretized into three priority levels that guide exercise selection: **High** priority ($S < 60$), **Medium** priority ($60 \leq S < 80$), and **Low** priority ($S \geq 80$). The agent plans practice by preferentially sampling exercises targeting High-priority KCs, followed by Medium-priority KCs, subject to practice-history-based diversity constraints that prevent repeated practice. Because high-load success contributes less to mastery growth, effortful KCs remain in higher-priority regions longer, allowing the agent to continue remediation rather than advancing solely based on correctness.

5.3 Action via Tool-Augmented Generation

When the pool of suitable exercises for the current priority set is insufficient, the agent triggers a retrieval-augmented generation (RAG) pipeline to produce new exercises. The pipeline retrieves two types of context from the exercise repository: (i) validated exemplar exercises from different topics as positive few-shot examples, and (ii) the learner's own previously practiced exercises matching the target KCs as negative "do-not-repeat" references.

Conditioned on this retrieved context, the LLM synthesizes a candidate exercise under structural constraints that enforce priority alignment and require a canonical program structure (task description, editable region, and executable test harness). Each generated

Table 1: 5-fold Model Performance Comparison.

Model	Model Size (MB)	Inference Time (ms)	Accuracy (%)	F1 score (%)
LR	0.001	0.09	76.62 ± 3.64	77.49 ± 4.42
SVM	0.052	0.13	77.01 ± 2.66	78.56 ± 2.81
RF	1.972	2.08	85.37 ± 3.31	87.73 ± 2.88
LSTM	0.532	1.05	82.99 ± 1.65	84.94 ± 1.58
PatchTST	1.604	8.50	78.41 ± 4.40	79.13 ± 5.39
PatchTST + MLP	1.775	2.35	71.74 ± 3.33	78.58 ± 2.14
iTransformer	1.830	1.45	78.81 ± 2.10	82.00 ± 1.44
TS2Vec + MLP	**1.500**	**4.81**	**86.07 ± 1.89**	**88.15 ± 1.59**

candidate then passes through a deterministic validation-and-repair toolchain (Figure 4) invoked by the Adaptive Practice Agent. The toolchain checks runnability, 100% functional correctness under the generated test harness, format compliance, and semantic novelty against prior exercises with the same KCs. If any check fails, the agent uses the tool output as repair feedback and re-prompts the LLM; validated exercises have their reference solutions masked and are inserted into the playlist and repository for future practice and retrieval, thereby closing the RAG loop.

From eight candidate models, Gemini 3 Flash was selected and asynchronously generated exercises in 11.67 s per candidate and 25.27 s per validated exercise, demonstrating high efficiency.

6 Preliminary Evaluation

6.1 In-situ Field Study with Real-world Use

Participants and Data Collection. We recruited 11 volunteers with prior programming experience (8 male, 3 female; aged 18–35). Nine had over one year of Python experience, while the remaining two had less than three months of Python experience but prior C/C++ experience and basic data-structure knowledge.

The study protocol was IRB-approved, and all participant data were anonymized. After informed consent and a brief tutorial, participants used CodeSensei for at least 30 minutes per day over 7–14 consecutive days on self-selected topics, with guidance to focus on a small set. CodeSensei logged keyboard/mouse traces and minute-wise cognitive-load labels, yielding 423 completed exercises and 1005 KC-level samples: 587 high-load and 418 low-load.

6.1.1 *Cognitive Load Inference Validity.*

Baselines and Evaluation Metrics. To evaluate the validity and deployability of the cognitive-load inference pipeline, we compare TS2Vec+MLP with representative baselines from multiple modeling families: Logistic Regression (LR), Support Vector Machine (SVM), Random Forest (RF), Long Short-Term Memory (LSTM), iTransformer [26], and PatchTST [32].

All models are trained and tested with the same preprocessing and normalization pipeline. We report accuracy and F1 score as the mean and standard deviation over session-level five folds, preventing windows from one exercise from crossing folds. Serialized model size and per-sample inference time measured over the full prediction path are reported to contextualize deployment cost.

Overall Model Performance. As shown in Table 1, TS2Vec+MLP achieves the best overall performance, reaching 86.07% ± 1.89% accuracy and 88.15% ± 1.59% F1, while remaining compact (1.50 MB) and efficient (4.81 ms latency). In this comparison, models denoted by "+MLP" use representations learned by a self-supervised

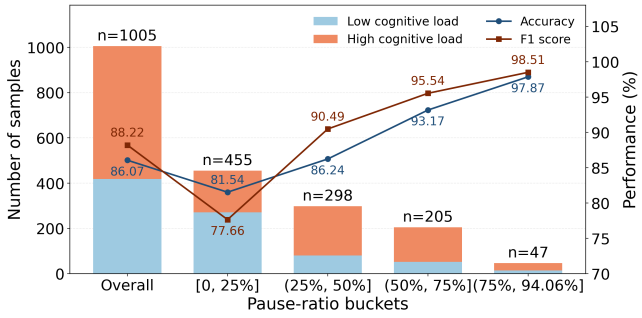

Figure 5: Sample composition and classification performance across pause-ratio buckets.

upstream encoder, followed by an MLP classification head, whereas models without this notation are trained end-to-end under supervised learning. RF is the strongest traditional baseline, whereas LR and SVM perform worse, indicating limited ability to model temporal interaction patterns. Transformer-based baselines also underperform, likely because they require larger labeled datasets than available here. In contrast, TS2Vec [53] leverages self-supervised temporal augmentations and hierarchical contrastive learning, producing robust representations from sparse behavioral traces; thus, a lightweight MLP head is sufficient for accurate and deployable cognitive load classification.

Under the same session-grouped five-fold protocol with logistic regression as classifier, pass-rate-only method only achieved 56.52% accuracy and 9.52% F1. This supports that using correctness alone is not a sufficient proxy for effort.

Low-interaction and Pause Analysis. Programming behavior is intermittent, with active keyboard and mouse interactions interleaved with pauses. We treat these pauses as part of the temporal behavioral signal rather than missing data, since they may reflect reading, planning, or debugging. We therefore examine whether cognitive-load inference remains reliable under different levels of interaction sparsity.

We stratify samples by pause ratio and analyze label composition and classification performance across groups (Figure 5). Low-interaction samples are common, with over 200 samples having pause ratios above 50%. Higher-pause groups contain more high-load samples, and model performance improves as pause ratio increases. This may indicate that longer pauses capture focused thinking or problem-solving, consistent with our protocol requiring participants to remain at the workstation. However, pauses are ambiguous in real-world deployment and may also reflect temporary absence. We note this limitation and discuss threshold-based or context-aware strategies for distinguishing productive pauses from disengagement in Sec. 7.

6.1.2 *User Experience.*

After the field deployment, participants completed the System Usability Scale (SUS) and Likert-scale questionnaires. The average SUS score reached 77.0, exceeding the usability benchmark of 68, with high ratings on satisfaction ($\mu = 8.64$), perceived improvement ($\mu = 8.09$), and mastery ($\mu = 7.82$).

Thematic analysis of open-ended feedback revealed four main advantages. Participants reported that the system provided **perceived**

adaptivity by targeting exercises to concepts they had previously struggled with. They appreciated the **fine-grained knowledge coverage**, which supported incremental understanding through KC decomposition. The **personalized learning pace** was viewed positively, as exercises progressed according to individual proficiency rather than a fixed schedule. Generated exercises were generally perceived as clear and relevant, and the IDE-integrated workflow offered good **usability and convenience**. Some participants noted occasional repetitiveness in highly targeted exercises and abrupt difficulty shifts during early sessions, suggesting that initial profile calibration can still be improved. Overall, these results indicate that the closed-loop adaptation is usable and perceived as beneficial in realistic IDE settings.

6.2 Short-term User Study

Participants and Data Collection. To complement the field deployment with controlled evidence, we conducted another IRB-approved short-term study comparing adaptive exercise sequencing against a fixed baseline under standardized conditions. We recruited 27 participants, with 24 valid participants after 3 dropouts. Each participant practiced two data-structure topics, linked lists and queues, with one topic assigned to the adaptive condition and the other to the fixed condition. Topic–condition assignment was counterbalanced across two groups, and participants were blinded to the condition. Participants first completed a 20-minute pre-test covering both topics. Each topic block then consisted of a 40-minute practice session followed by a 10-minute topic-specific post-test. In the adaptive condition, CodeSensei generated exercises based on the participant's current KC profile and priority levels. In the fixed condition, participants received curated exercises for the same topic, sequenced to cover relevant KCs without exercise repetition.

Evaluation Metrics. We used two complementary measures: (1) pupil dilation as an objective, non-intrusive proxy for cognitive effort [31, 46, 52], and (2) expert-graded learning gain, defined as the topic-specific post-test score minus the corresponding pre-test score, with two independent 0–100 expert scores averaged.

Pupil-based Evidence. For the pupil analysis, severely corrupted eye-tracking data were excluded. We collected over one million pupil-dilation samples during practice and used them as aggregate auxiliary evidence rather than independent ground-truth labels. At the aggregate sample level, KCs predicted as high load by the model showed higher pupil dilation than those predicted as low load ($\mu = 0.134$ vs. $\mu = 0.058$, $p < 0.001$), supporting the alignment between model-inferred load and established cognitive-science findings. Post-test pupil dilation reduction was also larger after adaptive practice than after fixed practice for the same topic, although the intra-individual difference did not reach statistical significance.

Learning Gains. Topic-wise Mann–Whitney U tests showed significantly higher learning gains under adaptive practice than under fixed practice for both linked lists and queues (both $p < 0.05$). Paired t-tests further showed a significant adaptive advantage for participants who practiced queues adaptively and linked lists under the fixed baseline ($p = 0.009$), but not for the reverse pairing ($p = 0.204$). These results provide exploratory evidence that closed-loop adaptation can yield measurable short-term benefits under controlled conditions, while suggesting that longer interaction may be needed for stable profile calibration.

7 Discussion and Future Work

Our studies show that fine-grained keyboard and mouse traces from everyday IDE use can provide practical, unobtrusive signals for inferring cognitive load during programming practice. At the same time, they surface several limitations that point to future directions for both sides of CodeSensei's agentic loop: behavioral trace interpretation and adaptive practice.

One central challenge for the Behavior-Semantic Interpretation Agent is interpreting pauses during interaction. Task-related thinking pauses can provide useful information for cognitive-load inference, but they are difficult to distinguish from disengagement or temporary absence. This ambiguity limits the robustness of interaction-only models in real-world settings. Future work could incorporate lightweight complementary signals, such as presence detection or duration heuristics, enabling the agent to better distinguish cognitive engagement from inactivity.

Although the Adaptive Practice Agent leverages the validation-and-repair toolchain to filter many low-quality generations, exercise diversity and semantic clarity can still be improved. Future work could explore more sophisticated prompting strategies, explicit planning constraints over KC transitions during generation, and lightweight verification mechanisms to increase single-pass reliability and reduce dependence on post-generation repair.

The current data scale also constrains deeper evaluation of personalization and long-term effects. With a limited number of participants and deployments lasting at most 14 days, our ability to observe stable personalization patterns and measure pedagogical impact remains restricted. Larger and longer deployments with more diverse learners would support finer-grained cognitive status modeling, more reliable estimation of learning gains, and better understanding of how KC-level mastery and cognitive-load profiles evolve over time.

More broadly, this work suggests that ubiquitous interaction traces can support behaviorally driven adaptive education beyond programming. The same interpret–plan–act–adapt loop could apply to other artifact-centered digital tasks, such as writing or design, pointing toward learning support embedded directly in the tools people already use rather than in separate educational platforms.

8 Conclusion

We presented CodeSensei, an IDE-integrated agentic sensing system that turns passive behavioral traces into adaptive programming education. In preliminary studies, CodeSensei achieved 88.15% F1 for binary cognitive-load classification. Our studies also suggest good usability, perceived adaptivity, and exploratory short-term learning benefits. These findings illustrate how agentic AI can move ubiquitous behavioral sensing beyond passive measurement toward in-situ educational intervention.

Acknowledgments

This work was supported in part by the Teaching and Learning Innovation Project funding from the Center for Education Innovation (CEI) at the Hong Kong University of Science and Technology.

References

[1] Murtaza Ali and Sayamindu Dasgupta. 2025. " Even Though I Went Through Everything, I Didn't Feel Like I Learned a Lot": Insights From Experiences of Non-Computer Science Students Learning to Code. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. 1–18.

[2] Alan Baddeley. 1988. Cognitive psychology and human memory. *Trends in neurosciences* 11, 4 (1988), 176–181.

[3] Jonathan Bassen, Bharathan Balaji, Michael Schaarschmidt, Candace Thille, Jay Painter, Dawn Zimmaro, Alex Games, Ethan Fast, and John C Mitchell. 2020. Reinforcement learning for the adaptive scheduling of educational activities. In *Proceedings of the 2020 CHI conference on human factors in computing systems*. 1–12.

[4] Jules Brooks Blanchet, Megan E. Hillis, Yeongji Lee, Qijia Shao, Xia Zhou, Devin Balkcom, and David J. M. Kraemer. 2025. Enhancing the Educational Potential of Online Movement Videos: System Development and Empirical Studies with TikTok Dance Challenges. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery, New York, NY, USA, Article 587, 19 pages. doi:10.1145/3706598.3714062

[5] Minghao Cai and Carrie Demmans Epp. 2022. Modeling Cognitive Load and Affect to Support Adaptive Online Learning. In *Proceedings of the 15th International Conference on Educational Data Mining*. 799.

[6] Minghao Cai and Carrie Demmans Epp. 2024. Exploring the Optimal Time Window for Predicting Cognitive Load Using Physiological Sensor Data. *arXiv preprint arXiv:2406.13793* (2024).

[7] Liuqing Chen, Shuhong Xiao, Yunnong Chen, Yaxuan Song, Ruoyu Wu, and Lingyun Sun. 2024. ChatScratch: An AI-augmented system toward autonomous visual programming learning for children aged 6-12. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 1–19.

[8] Ti-Chung Cheng, Yutong Zhang, Yi-Hung Chou, Vinay Koshy, Tiffany Wenting Li, Karrie Karahalios, and Hari Sundaram. 2025. Organize, Then Vote: Exploring Cognitive Load in Quadratic Survey Interfaces. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. 1–35.

[9] Kabdo Choi, Hyungyu Shin, Meng Xia, and Juho Kim. 2022. Algosolve: Supporting subgoal learning in algorithmic problem-solving with learnersourced microtasks. In *Proceedings of the 2022 CHI conference on human factors in computing systems*. 1–16.

[10] Konstantina Chrysafiadi and Maria Virvou. 2014. Fuzzy logic for adaptive instruction in an e-learning environment for computer programming. *IEEE transactions on Fuzzy Systems* 23, 1 (2014), 164–177.

[11] Phoebe Chua, Prasanth Sasikumar, Yadeesha Weerasinghe, and Suranga Nanayakkara. 2024. Motion as Emotion: Detecting Affect and Cognitive Load from Free-Hand Gestures in VR. *arXiv preprint arXiv:2409.12921* (2024).

[12] Fernanda M De La Torre, Javier Hernandez, Andrew D Wilson, and Judith Amores. 2025. Sonora: Human-AI Co-Creation of 3D Audio Worlds and its Impact on Anxiety and Cognitive Load. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. 1–22.

[13] Paul Denny, Juho Leinonen, James Prather, Andrew Luxton-Reilly, Thezyrie Amarouche, Brett A Becker, and Brent N Reeves. 2024. Prompt Problems: A new programming exercise for the generative AI era. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*. 296–302.

[14] Andrew T Duchowski, Krzysztof Krejtz, Izabela Krejtz, Cezary Biele, Anna Niedzielska, Peter Kiefer, Martin Raubal, and Ioannis Giannopoulos. 2018. The index of pupillary activity: Measuring cognitive load vis-à-vis task difficulty with pupil oscillation. In *Proceedings of the 2018 CHI conference on human factors in computing systems*. 1–13.

[15] James Finnie-Ansley, Paul Denny, Andrew Luxton-Reilly, Eddie Antonio Santos, James Prather, and Brett A Becker. 2023. My ai wants to know if this will be on the exam: Testing openai's codex on cs2 programming exercises. In *Proceedings of the 25th Australasian Computing Education Conference*. 97–104.

[16] Lex Fridman, Bryan Reimer, Bruce Mehler, and William T Freeman. 2018. Cognitive load estimation in the wild. In *Proceedings of the 2018 chi conference on human factors in computing systems*. 1–9.

[17] Ashley Ge Zhang, Yan-Ru Jhou, Yinuo Yang, Shamita Rao, Maryam Arab, Yan Chen, and Steve Oney. 2026. CodeStream: Augmenting Timelines with Code Annotation for Navigating Large Coding Histories. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*. 1–16.

[18] GeeksforGeeks. 2025. *Python Tutorial - Learn Python Programming Language*. Retrieved July 20, 2025 from https://www.geeksforgeeks.org/python/python-programming-language-tutorial/, archived at [https://web.archive.org/web/20250720000000/https://www.geeksforgeeks.org/python/python-programming-language-tutorial/]

[19] Tan Gemicioglu, Elijah Hopper, Brahmi Dwivedi, Richa Kulkarni, Asha Bhandarkar, Priyanka Rajan, Nathan Eng, Adithya Ramanujam, Charles Ramey, Scott M Gilliland, et al. 2024. Passive Haptic Rehearsal for Augmented Piano Learning in the Wild. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 8, 4 (2024), 1–26.

[20] Xinying Hou, Zihan Wu, Xu Wang, and Barbara J Ericson. 2024. Codetailor: Llm-powered personalized parsons puzzles for engaging support while learning programming. In *Proceedings of the Eleventh ACM Conference on Learning@ Scale*. 51–62.

[21] Hyoungwook Jin, Seonghee Lee, Hyungyu Shin, and Juho Kim. 2024. Teach ai how to code: Using large language models as teachable agents for programming education. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 1–28.

[22] Majeed Kazemitabaar, Runlong Ye, Xiaoning Wang, Austin Zachary Henley, Paul Denny, Michelle Craig, and Tovi Grossman. 2024. Codeaid: Evaluating a classroom deployment of an llm-based programming assistant that balances student and educator needs. In *Proceedings of the 2024 chi conference on human factors in computing systems*. 1–20.

[23] Paul A Kirschner. 2002. Cognitive load theory: Implications of cognitive load theory on the design of learning. 10 pages.

[24] Myungjae Kwak, Jonathan Jenkins, and Joobum Kim. 2023. Adaptive programming language learning system based on generative AI. *Issues in Information Systems* 24, 3 (2023).

[25] Hsiao-Chi Ling and Hsiu-Sen Chiang. 2022. Learning performance in adaptive learning systems: A case study of web programming learning recommendations. *Frontiers in psychology* 13 (2022), 770637.

[26] Yong Liu, Tengge Hu, Haoran Zhang, Haixu Wu, Shiyu Wang, Lintao Ma, and Mingsheng Long. 2023. itransformer: Inverted transformers are effective for time series forecasting. *arXiv preprint arXiv:2310.06625* (2023).

[27] Wenhan Lyu, Yimeng Wang, Yifan Sun, and Yixuan Zhang. 2025. Will Your Next Pair Programming Partner Be Human? An Empirical Evaluation of Generative AI as a Collaborative Teammate in a Semester-Long Classroom Setting. In *Proceedings of the Twelfth ACM Conference on Learning@ Scale*. 83–94.

[28] Qianou Ma, Hua Shen, Kenneth Koedinger, and Sherry Tongshuang Wu. 2024. How to teach programming in the ai era? using llms as a teachable agent for debugging. In *International Conference on Artificial Intelligence in Education*. Springer, 265–279.

[29] Shuai Ma, Junling Wang, Yuanhao Zhang, Xiaojuan Ma, and April Yi Wang. 2025. Dbox: Scaffolding algorithmic programming learning through learner-llm co-decomposition. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. 1–20.

[30] Eric Matthes. 2023. *Python crash course: A hands-on, project-based introduction to programming*. no starch press.

[31] Ritayan Mitra, Karen S McNeal, and Howard D Bondell. 2017. Pupillary response to complex interdependent tasks: A cognitive-load theory perspective. *Behavior Research Methods* 49, 5 (2017), 1905–1919.

[32] Yuqi Nie, Nam H Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. 2022. A time series is worth 64 words: Long-term forecasting with transformers. *arXiv preprint arXiv:2211.14730* (2022).

[33] Zachary A Pardos, Zihao Fan, and Weijie Jiang. 2018. Connectionist recommendation in the wild: on the utility and scrutability of neural networks for personalized course guidance. *arXiv preprint arXiv:1803.09535* (2018).

[34] James Prather, Brent N Reeves, Juho Leinonen, Stephen MacNeil, Arisoa S Randrianasolo, Brett A Becker, Bailey Kimmel, Jared Wright, and Ben Briggs. 2024. The widening gap: The benefits and harms of generative ai for novice programmers. In *Proceedings of the 2024 ACM Conference on International Computing Education Research-Volume 1*. 469–486.

[35] Refsnes Data. 1998. *W3Schools Online Web Tutorials*. Retrieved July 20, 2025 from https://www.w3schools.com/, archived at [https://web.archive.org/web/20250720000000/https://www.w3schools.com/]

[36] Hansol Rheem, Vipin Verma, and D Vaughn Becker. 2018. Use of mouse-tracking method to measure cognitive load. In *Proceedings of the human factors and ergonomics society annual meeting*, Vol. 62. SAGE Publications Sage CA: Los Angeles, CA, 1982–1986.

[37] Eva Ritz, Thiemo Wambsganss, Roman Rietsche, Anuschka Schmitt, Sarah Oeste-Reiss, and Jan Marco Leimeister. 2022. Unleashing process mining for education: Designing an IT-tool for students to self-monitor their personal learning paths. *Internationale Tagung Wirtschaftsinformatik (WI)* (2022).

[38] Sherry Ruan, Liwei Jiang, Justin Xu, Bryce Joe-Kun Tham, Zhengneng Qiu, Yeshuang Zhu, Elizabeth L Murnane, Emma Brunskill, and James A Landay. 2019. Quizbot: A dialogue-based adaptive learning system for factual knowledge. In *Proceedings of the 2019 CHI conference on human factors in computing systems*. 1–13.

[39] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic generation of programming exercises and code explanations using large language models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1*. 27–43.

[40] Florian Schaule, Jan Ole Johanssen, Bernd Bruegge, and Vivian Loftness. 2018. Employing consumer wearables to detect office workers' cognitive load for interruption management. *Proceedings of the ACM on interactive, mobile, wearable and ubiquitous technologies* 2, 1 (2018), 1–20.

[41] Robert Sedgewick, Kevin Wayne, and Robert Dondero. 2015. *Introduction to programming in Python: An interdisciplinary approach*. Addison-Wesley Professional.

[42] Ngo Tung Son, Jafreezal Jaafar, Izzatdin Abdul Aziz, and Bui Ngoc Anh. 2021. Meta-heuristic algorithms for learning path recommender at MOOC. *IEEE Access* 9 (2021), 59093–59107.

[43] John Sweller. 1994. Cognitive load theory, learning difficulty, and instructional design. *Learning and instruction* 4, 4 (1994), 295–312.

[44] Ha Trinh, Reza Asadi, Darren Edge, and T Bickmore. 2017. Robocop: A robotic coach for oral presentations. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 2 (2017), 1–24.

[45] Dishita G Turakhia, Andrew Wong, Yini Qi, Lotta-Gili Blumberg, Yoonji Kim, and Stefanie Mueller. 2021. Adapt2learn: A toolkit for configuring the learning algorithm for adaptive physical tools for motor-skill learning. In *Proceedings of the 2021 ACM Designing Interactive Systems Conference*. 1301–1312.

[46] Pascal WM Van Gerven, Fred Paas, Jeroen JG Van Merriënboer, and Henk G Schmidt. 2004. Memory load and the cognitive pupillary response in aging. *Psychophysiology* 41, 2 (2004), 167–174.

[47] Thiemo Wambsganss, Tobias Kueng, Matthias Soellner, and Jan Marco Leimeister. 2021. ArgueTutor: An adaptive dialog-based learning system for argumentation skills. In *Proceedings of the 2021 CHI conference on human factors in computing systems*. 1–13.

[48] Thiemo Wambsganss and Anuschka Schmitt. 2024. Enhancing Personalized Learning Through Process Mining: A Taxonomy and Design Patterns Approach. *Business & Information Systems Engineering* (2024), 1–24.

[49] Qiuzhen Wang, Sa Yang, Manlu Liu, Zike Cao, and Qingguo Ma. 2014. An eye-tracking study of website complexity from cognitive load perspective. *Decision support systems* 62 (2014), 1–10.

[50] Huan Wei, Haotian Li, Meng Xia, Yong Wang, and Huamin Qu. 2020. Predicting student performance in interactive online question pools using mouse interaction features. In *Proceedings of the tenth international conference on learning analytics & knowledge*. 645–654.

[51] Lennart Wittke, Peter El Hachem, Yuan Chen, Joseph Ollier, Bingjie Cheng, Shkurta Gashi, and Xiaoyu Zhang. 2025. Classifying Cognitive Load in Digital Textbook Reading Using Electroencephalogram Signals. In *2025 13th International Conference on Information and Education Technology (ICIET)*. IEEE, 316–321.

[52] Hoe Kin Wong and Julien Epps. 2016. Pupillary transient responses to within-task cognitive load variation. *Computer Methods and Programs in Biomedicine* 137 (2016), 47–63.

[53] Zhihan Yue, Yujing Wang, Juanyong Duan, Tianmeng Yang, Congrui Huang, Yunhai Tong, and Bixiong Xu. 2022. Ts2vec: Towards universal representation of time series. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 36. 8980–8987.

[54] Chengbo Zheng, Kangyu Yuan, Bingcan Guo, Reza Hadi Mogavi, Zhenhui Peng, Shuai Ma, and Xiaojuan Ma. 2024. Charting the future of AI in project-based learning: a Co-design exploration with students. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 1–19.

[55] Yihao Zhu, Zhoutong Ye, Yichen Yuan, Wenxuan Tang, Chun Yu, and Yuanchun Shi. 2025. AutoPBL: An LLM-powered Platform to Guide and Support Individual Learners Through Self Project-based Learning. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. 1–26.