# Multimodal Learning Analytics for Programming: Cognitive Difficulties, Attention Dynamics, and Physiological Regulation

Huiyong Li
Kyushu University
li.huiyong.194@m.kyushu-u.ac.jp

Boxuan Ma
Kyushu University
boxuan@artsci.kyushu-u.ac.jp

## ABSTRACT

AAT-based programming practice often relies on log outcomes that obscure learners' underlying cognitive difficulties, behavioral workflows, and regulation dynamics. This study synchronizes submission logs, eye tracking, and wearable physiology to examine how these processes unfold during authentic AAT-based programming and how they differ by prior knowledge. Twenty-two university students completed a 35-minute session with four beginner Python questions while wearing a Pupil Core eye tracker and an EmbracePlus wristband. We coded attempt-level syntactic, conceptual, and strategic difficulties, annotated seven task-relevant behavioral states from screen–gaze recordings, and analyzed physiological signals across preparation and programming stages. Results showed that low prior-knowledge learners produced more syntactic and conceptual difficulty attempts, cycled more often between learning materials, feedback, and code revision, and showed larger reductions in physiological regulation during programming. These findings demonstrate how temporally aligned multimodal evidence can move AAT analytics beyond outcome tracking toward process-sensitive learner models that adapt support to cognitive difficulty, behavioral transition patterns, and regulation states.

## Keywords

Introductory programming, automated assessment tools, learning analytics, eye-tracking, physiological regulation

## 1. INTRODUCTION

Learning to program involves progressing from mastering language features, to forming domain concepts, and ultimately to solving novel problems [34, 28, 21]. In large programming courses, scaling such formative practice increasingly relies on automated assessment tools (AATs) embedded in learning management systems. AATs execute submissions against predefined test cases and return error messages, failing test indicators, and scores [22], that making practice frequent, measurable, and comparable across students.

Huiyong Li, and Boxuan Ma. Multimodal Learning Analytics for Programming: Cognitive Difficulties, Attention Dynamics, and Physiological Regulation. In Anthony Botelho, Maria Mercedes T. Rodrigo, Adish Singla, Hiroaki Ogata, Hyojeong So, and Young Hoan Cho (eds.) Proceedings of the 19th International Conference on Educational Data Mining, Seoul, Republic of Korea, June, 2026, pp. 516–523. International Educational Data Mining Society (2026).



https://doi.org/10.5281/zenodo.21039831

This fast feedback has enabled a growing body of computing education and educational data mining research on engagement, persistence, and performance in AAT-based settings [8]. However, the same observable trace can be produced by fundamentally different cognitive states: a student may repeatedly fail due to a minor syntax slip, a fragile misconception, or an unproductive strategy (e.g., trial-and-error), yet these processes can appear similar in logs. In this sense, log traces primarily capture what learners did and whether it worked, offering limited evidence about how and why those behaviors unfold. Complementary evidence streams point to this missing process layer. Prior work suggests that programming success and failure are associated with distinct visual attention strategies [35], and physiological sensing can index variations in cognitive and affective effort during code comprehension [7]. However, in AAT-based problem solving, explaining why superficially similar log traces arise often requires linking observable actions to learners' attention allocation and regulation dynamics at a fine temporal granularity. Multimodal learning analytics (MMLA) provides a methodological basis for synchronizing and triangulating these heterogeneous streams [27]. Yet such synchronized, fine-grained integration remains uncommon in authentic AAT task solving, where behavioral traces, attention dynamics, and regulation signals must be aligned to reveal process mechanisms rather than outcomes alone.

Despite the data richness of AAT-based programming learning, three gaps remain. First, prior work typically analyzes a single modality in isolation, limiting explanatory power because each stream is ambiguous on its own [36, 18]. Second, process-level differentiation of learners remains underspecified: we still lack sequence-sensitive characterizations of behavioral states, which cognitive difficulties emerge and how they co-occur with strategy-relevant behaviors across problem-solving episodes [16, 14]. Third, physiological regulation dynamics are rarely examined alongside fine-grained programming behaviors in authentic AAT task solving, leaving unclear whether and how physiological regulation signals difficulty, supports recovery, or differentiates effective versus ineffective strategy shifts.

To address these gaps, we integrate log-based cognitive difficulty coding, eye-tracking attention dynamics, and physiological regulation measures to characterize programming processes and contrast high and low prior-knowledge learners. We address the following research questions:

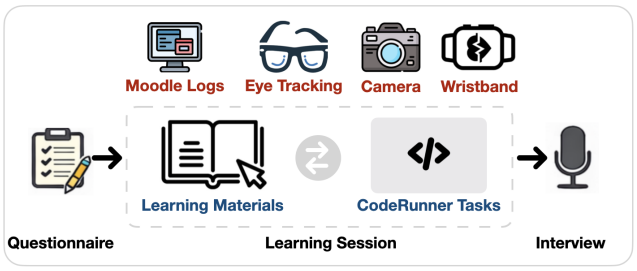


**Figure 1: Overview of the study procedure.**


**Figure 2: CodeRunner interface in Moodle.**

**RQ1:** What cognitive difficulties (syntactic, conceptual, and strategic) emerge in programming tasks, and how do they differ between high and low prior-knowledge learners?

**RQ2:** What visual attention and behavioral transitions across task-relevant states emerge, and how do they differ between high and low prior-knowledge learners?

**RQ3:** What physiological regulation patterns/indices emerge, and how do they differ between high and low prior-knowledge learners?

## 2. RELATED WORK

Automated assessment tools are widely used in introductory programming because executable submissions enable scalable, consistent evaluation and rapid feedback [2, 13]. In the EDM and LA field, AAT logs (e.g., attempts, compiler outcomes, timestamps, and pass or fail test results) have been mined to model performance, persistence, and iterative behaviors in problem-level analyses [20, 1, 16, 15]. However, similar log traces can reflect different underlying processes, motivating process-oriented modeling beyond outcome indicators.

To interpret these ambiguous traces, computing education research commonly adopts a multi-layer knowledge framework with syntactic knowledge of language features, conceptual understanding of execution and constructs, and strategic knowledge for problem solving and debugging [21, 28, 34]. Large-scale analyses of submission corpora further show that student errors span additional categories (e.g., misinterpretation or carelessness), cautioning that code-only evidence may under-specify underlying causes [3]. Recent work operationalizes closely related difficulty categories (syntactic, conceptual, strategic) with explicit definitions and expert labeling [12], while semi-automated approaches such as code-embedding-based clustering have been proposed to discover problem-specific misconceptions from incorrect submissions at scale [31]. These lines of work support interpretable difficulty characterization, but typically do not connect difficulties to concurrent attention allocation and regulation dynamics.

To bridge this gap, complementary modalities provide process evidence that logs alone cannot. Eye-tracking studies in programming report systematic differences in attention strategies across skill levels and performance, motivating transition-based analyses over task-relevant regions and representations [25, 11, 5]. Physiological sensing has also been used to estimate cognitive effort and affect during programming and code comprehension (e.g., via EDA and heart-rate-related measures), with validation work supporting wearable instrumentation for more naturalistic settings [7, 23, 10]. Methodologically, multimodal learning analytics provides frameworks for synchronizing and fusing heterogeneous streams, highlighting the role of fusion design choices for validity and interpretability and showing benefits over single-modality models [26, 4, 24, 9, 33, 27].

While prior multimodal work demonstrates feasibility in programming education [19], relatively few studies jointly link AAT log-derived cognitive difficulties, attention transitions, and physiological regulation in authentic AAT-based programming task solving. Building on these strands, our study synchronizes AAT logs, eye movements, and wearable physiology to examine how these process indicators co-occur and differ by prior knowledge.

## 3. METHOD

### 3.1 Participants

Twenty-two university students voluntarily participated in this study (14 males and 8 females, with an average age of 24). Participants were recruited through campus announcements at a national university in Japan and included both undergraduate and graduate students from diverse academic disciplines. Programming experience ranged from no prior experience to nine years, covering both novice and experienced learners. The study protocol was reviewed and approved by the authors' university Research Ethics Committee and conducted in accordance with institutional guidelines for human-participant research. Participants provided written informed consent and could withdraw at any time.

### 3.2 Study Design and Learning Tasks

The overview of the study procedure is shown in Figure 1. This study was conducted in a controlled laboratory environment. Firstly, the students completed a pre-test and pre-questionnaire after a brief experimental introduction. Then the students were asked to access an introductory programming course in the Moodle learning management system (LMS). They were asked to read the lecture's introductory materials, which covered fundamental concepts in Python programming (e.g., basic functions) and solve four beginner-level programming questions using CodeRunner, an open-source Moodle plugin automated assessment tool [17]. Students were given the flexibility to complete the task at their

Table 1: Coding scheme for student cognitive difficulties, descriptions, and examples.

| Difficulty | Description | Example |
|---|---|---|
| **Syntactic** | Misunderstanding of syntactic facts related to a particular programming language or failure on applying rules of syntax. | (1) Missing colon in `def` function header:<br>`def print_morning print("")`<br>(2) Non-standard symbols (fullwidth parentheses in function)<br>(3) Wrong newline escape (uses /n instead of \n) |
| **Conceptual** | Misunderstanding of programming constructs or execution semantics. | (1) `return` vs. `print` confusion (prints but should return):<br>`def convert_days_to_seconds(d): print(d*86400)`<br>(2) Identifier misuse (uses undeclared name):<br>`def repeat_five_times(s): print(sssss)`<br>(3) Misunderstanding "repeat 5 times" (single call / one line):<br>`def repeat_five_times(s): print(s,s,s,s,s)` |
| **Strategic** | Difficulties applying programming to solve novel problems by design, coding, and test. | (1) Task confusion (wrong function name than specified)<br>`def print_morning(): print("...") ...`<br>(2) Output-format planning issue (unwanted newlines):<br>`for i in range(5): print(s)` (*missing* `end=""`) |

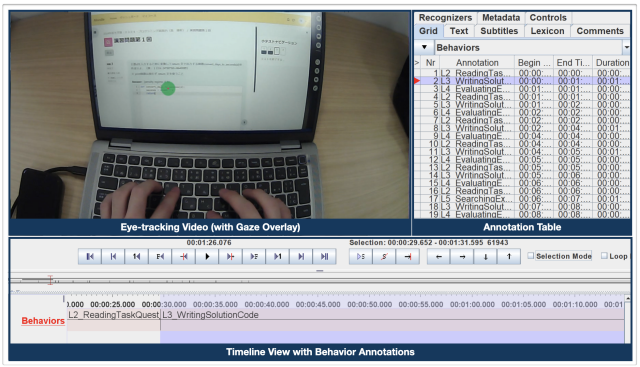


**Figure 3: ELAN software used to annotate learning behaviors from eye-tracking videos.**

own pace within a 35-minute learning session. An example CodeRunner question is shown in Figure 2. They could read the problem statement, input their code, and submit their solutions to be automatically checked against pre-defined test cases.

Students wore a 200-Hz Pupil Core eye tracker while solving programming tasks. Pupil Core (Pupil Labs GmbH, Berlin, Germany) is a pair of eye-tracking glasses endowed with two infrared real-time eye cameras and an environmental camera focused straight ahead of the screen. The eye-tracking screen data was recorded from the cameras and processed via the Pupil Player software later. Students also wore an EmbracePlus wristband before and during the learning session. They spent approximately 20 minutes in the preparation stage before they started 35-minute programming tasks. EmbracePlus wristband is a medical-grade wearable device that offers real-time physiological data collection [6]. The wristband embeds 3-axis accelerometer, electrodermal activity, temperature, and gyroscope sensors. It is a non-invasive device worn like a wristwatch, minimizing any discomfort during usage.

After completing the learning session, all participants took part in semi-structured, face-to-face interviews. The interviews probed students' learning experiences, focusing on perceived successes as well as aspects of the learning process that required improvement.

## 3.3 Data Collection and Analysis

Students' prior knowledge of Python programming was assessed using a 10-item pre-test covering core concepts, including data types, control flow, functions, and basic syntax in Python. Based on their pre-test scores, students were classified into two prior-knowledge groups: high prior-knowledge who scored 8 or higher out of 10 ($n = 10$), and low prior-knowledge who scored 6 or lower ($n = 12$). The cut-off leverages a natural gap in the distribution (no student scored 7), producing clearly separated groups for subsequent multimodal comparisons.

A pre-questionnaire was administered to collect participants' demographic information, including gender, age, and academic major.

All student interactions with the Moodle platform were logged automatically. CodeRunner problem-solving behavior was captured in learning logs, including assessment logs, submitted codes, and system feedback. The raw logs contained question-level information, including question contents, question access timestamps, and question start times, as well as attempt-level step histories for each learner and question. Each step record included the step number, timestamp, action type, submitted answer, correctness state, score, and CodeRunner feedback such as error messages. These learning logs were classified into cognitive difficulties including syntactic, conceptual, and strategic types, adopted a prior conceptual framework [28]. The coding scheme is summarized in Table 1.

Students' behaviors during programming tasks were captured using eye-tracking video recordings. The raw eye-tracking data consisted of time-stamped gaze recordings collected with the Pupil Core eye tracker and synchronized screen recordings of students' interactions with the Moodle-based CodeRunner environment. Fixations were automati-

Table 2: Coding scheme of student learning behaviors during programming tasks.

| Code | Behavior | Description |
|---|---|---|
| **Lecture** | Watching Lecture Material | Students viewed the instructional materials provided in the Moodle course. |
| **Question** | Reading Task Question | Students read and examined the programming task statement. |
| **Code** | Writing Solution Code | Students wrote program code in the coding box to solve the task. |
| **Fail** | Execution Results Fail | Students submitted their code for automated assessment, but it failed one or more test cases or returned an execution error. |
| **Pass** | Execution Results Pass | Students submitted their code for automated assessment, and it successfully passed all test cases. |
| **Search** | Searching External Resources | Students searched the internet for task-related information, programming concepts, or error explanations. Use of generative AI tools like ChatGPT was not allowed. |
| **Note** | Taking Notes | Students wrote notes or sketched ideas on paper during the learning session. |

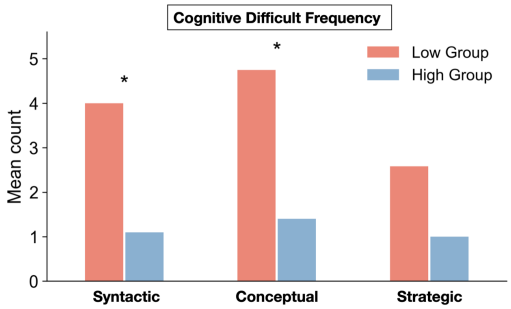


Figure 4: The distribution of cognitive difficulties faced by learners with high and low prior-knowledge.

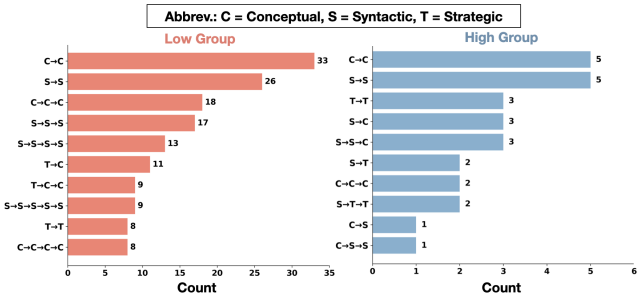


Figure 5: Top 10 most frequent cognitive transition sequences by group.

cally extracted in Pupil Player using the dispersion-threshold identification (I-DT) algorithm, with a dispersion threshold of 1.5° visual angle and a minimum fixation duration of 100 ms [29]. The synchronized screen and gaze videos were manually annotated to extract observable programming behaviors as shown in Figure 3. Following prior research on SRL in programming [32], seven programming behaviors were coded as summarized in Table 2. These behaviors captured how students engaged with lecture materials, understood problem requirements, created and evaluated solutions, sought external information, and noted their thinking processes. The annotated behaviors were used to compute behavioral frequency, duration, and transitions, enabling comparison of behavioral dynamics across different task contexts. Specifically, temporal behavioral sequences were extracted at the programming question level, with each programming question treated as an independent learning episode.

To compare programming behaviors between groups, we analyzed both overall activity and temporal dynamics. For frequency and duration measures, we used independent-samples $t$-tests or Wilcoxon rank-sum tests after a normality check. To model behavioral dynamics, we applied Transition Network Analysis (TNA) [30], representing each learner's question-level sequence as a weighted directed network (nodes = behavioral states; edges = transition probabilities). Transition probabilities were aggregated within each group. Between-group differences in transition probabilities were tested using Fisher's exact test.

Students' physiological signals were collected using the EmbracePlus wristband during the learning session. The raw data consisted of time-stamped electrodermal activity (EDA) and blood volume pulse (BVP) streams synchronized with the task timeline. Because wrist-based physiological data are sensitive to movement artifacts and physiological responses can appear with a delay after cognitive or affective events, we used a window-based analysis. Specifically, signals were segmented into non-overlapping 1-minute windows and aligned with two task stages: preparation and programming. We derived four features: (1) tonic EDA as log-transformed skin conductance level (log(SCL)); (2) phasic activity as log1p-transformed skin conductance response rate (log1p(SCR/min)); (3) heart rate as mean beats per minute (BPM); and (4) heart-rate variability as log-transformed RMSSD (log(RMSSD)). To reduce between-person baseline differences, we computed within-person z-scored composites: (5) an *Arousal Index* as the mean of z-scored log(SCL) and log1p(SCR/min), and (6) a *Regulation Index* as z-scored log(RMSSD). We tested main effects and interactions of *Group* (low vs. high prior knowledge) and *Stage* (preparation vs. programming) on each feature and index using mixed-effects models.

# 4. RESULTS

## 4.1 Log-based Cognitive Difficulties (RQ1)

Figure 4 shows the distribution of cognitive difficulties in the syntactic, conceptual and strategic types faced by learners with high and low prior-knowledge. Low prior-knowledge students produced significantly more *Syntactic* (U = 97.0, p = .037, r = 0.62) and *Conceptual* (U = 96.5, p = .037, r = 0.61) difficulty attempts than high prior-knowledge stu-

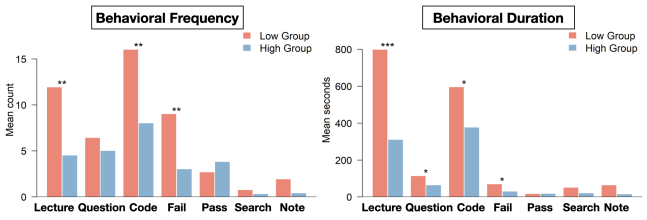


**Figure 6: Programming behavioral differences in frequency and duration between high and low prior-knowledge learners.**

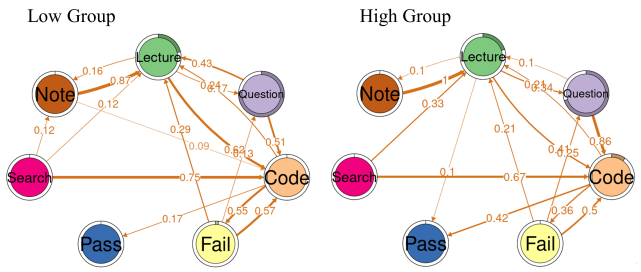


**Figure 7: Transition network of programming behaviors between high and low prior-knowledge learners. Nodes represent each coded behavior. Edges represent transition probabilities between nodes.**

dents. *Strategic* differences trend higher for low prior-knowledge students but were not significant (U = 83.5, p = .119).

We further analyzed sequence patterns by extracting learner's attempt-level contiguous subsequences. Mann–Whitney U tests show no subsequence differences that remained significant ($p_{\text{Holm}} = 1.0$ for all). Low prior-knowledge learners exhibited more persistence of conceptual difficulty, more longer repeated syntactic failures, and more strategic-to-conceptual shifts, whereas high prior-knowledge learners show fewer failure episodes overall and fewer transitions. Figure 5 shows top 10 most frequent cognitive transition sequences for low (left) and high (right) prior-knowledge groups.

## 4.2 Eye-tracking Attention Dynamics (RQ2)

Figure 6 shows the programming behavioral differences in frequency and duration between high and low prior-knowledge learners. Results show that low prior-knowledge learners spent significantly more time on Lecture viewing (M = 797s vs. 310s, p < .001), Question reading (M = 113s vs. 63s, p = .017), Code writing (M = 594s vs. 376s, p = .033), and examining Fail results (M = 69s vs. 29s, p = .012). Notably, Lecture duration showed the largest effect size (d = 1.81), indicating that low-group learners spent more than double the time revisiting lecture materials.

Figure 7 shows the transition network of programming behaviors between high and low prior-knowledge learners. Seven significant transition differences were found between low and high prior-knowledge learners using Fisher's exact test. The high prior-knowledge group followed a more execution-oriented path, with higher *Question*→*Code* (86.0% vs. 50.6%; $p = 5.1 \times 10^{-5}$) and *Code*→*Pass* (42.5% vs. 16.8%; $p = 1.9 \times 10^{-5}$), and fewer *Code*→*Fail* transitions (36.2% vs. 55.5%; $p = 5.0 \times 10^{-3}$). In contrast, the low prior-knowledge group relied more on materials, showing higher *Question*→*Lecture* (42.9% vs. 10.0%; $p = 5.9 \times 10^{-5}$) and *Lecture*→*Code* (62.2% vs. 41.5%; $p = 3.0 \times 10^{-2}$). Lecture use also differed in integration: high prior-knowledge learners more often returned to refine task understanding (*Lecture*→*Question*: 34.1% vs. 17.0%; $p = 2.8 \times 10^{-2}$) and were more likely to reach success after consulting materials (*Lecture*→*Pass*: 9.8% vs. 0.0%; $p = 2.6 \times 10^{-3}$). Overall, high prior-knowledge learners progressed more directly from problem understanding to coding and success, whereas low prior-knowledge learners showed a lecture-mediated and more failure-prone workflow.

## 4.3 Physiological Regulation Signals (RQ3)

Figure 8 presents minute-level physiological trajectories (panels a–d) and Group × Stage comparisons (panels e–h) for low and high prior-knowledge learners. Panels (a)–(d) show GAM-smoothed group trajectories with ±1 SEM bands for tonic EDA [log(SCL)], phasic SCR rate [log1p(SCR/min)], heart rate (BPM), and RMSSD [log(RMSSD)], respectively. Mixed-effects models controlling for MET indicated a robust main effect of stage for three features: tonic EDA increased from Preparation to Programming ($b = 0.33$, $p < .001$), phasic SCR rate increased ($b = 0.24$, $p < .001$), and RMSSD decreased ($b = -0.36$, $p < .001$), whereas heart rate showed no stage shift ($b = -0.12$, $p = .859$). A significant Group x Stage interaction was observed for phasic SCR rate (b = -0.21, p = .006; panel f), indicating a larger Programming-related increase for low prior-knowledge learners than for high prior-knowledge learners. No Group × Stage interactions were found for tonic EDA ($p = .680$), heart rate ($p = .260$), or RMSSD ($p = .095$).

Figure 9 summarizes the physiological results using within-person standardized, minute-level composite indices. Panels (a–b) show GAM-smoothed trajectories (±1 SEM) for the Arousal Index, combining tonic SCL and phasic SCR activity, and the Regulation Index, based on RMSSD. These smoothed trajectories indicate relative changes in physiological activation and regulation across the preparation and programming stages. Panels (c–d) show the mixed-effects model estimates for Group × Stage effects. Arousal increased from Preparation to Programming ($b = 0.59$, $p < .001$), with no Group × Stage interaction ($b < 0.01$, $p = .992$), suggesting comparable increases in sympathetic activation for both prior-knowledge groups. Regulation decreased from Preparation to Programming ($b = -0.52$, $p < .001$), indicating lower RMSSD-based regulation during active programming. This decrease was qualified by a significant Group × Stage interaction ($b = 0.40$, $p < .001$): low prior-knowledge learners showed a larger programming-related reduction in the Regulation Index than high prior-knowledge learners.

# 5. DISCUSSION

In this study, we conduct a multimodal study that synchronizes submission logs, eye tracking, and wearable physiology to characterize how cognitive difficulties, visual attention transitions, and physiological regulation unfold during an AAT-based programming learning environment and how they differ by prior knowledge. Across learning log, attention-based behavior, and physiological signal, the re-

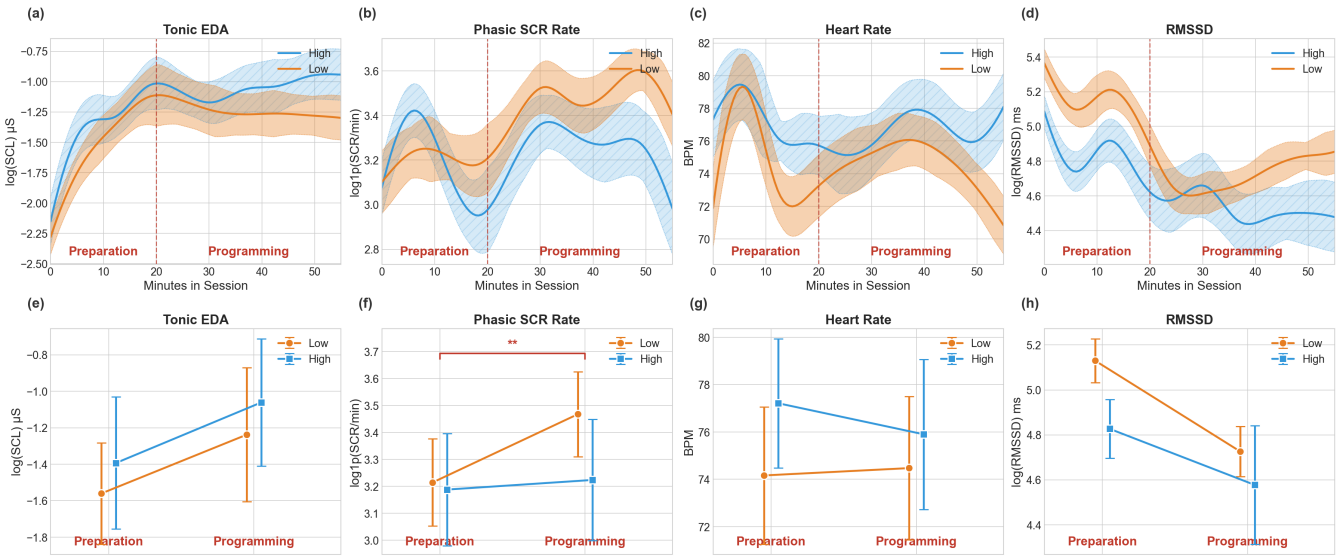


**Figure 8: Physiological trajectories and Group × Stage effects for low and high prior-knowledge learners.**

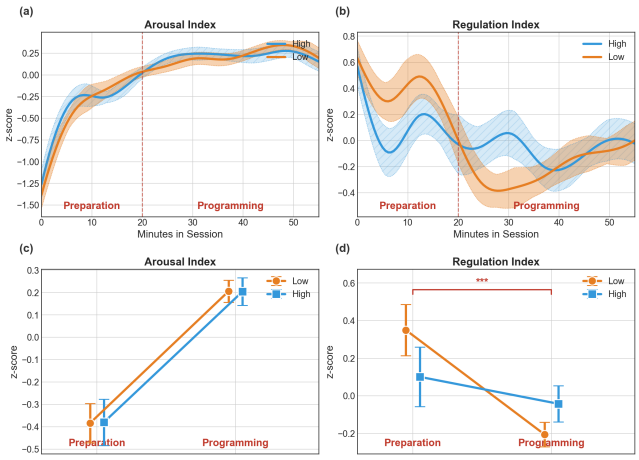


**Figure 9: Higher-order physiological indices (within-person z-scored) and Group × Stage effects for low and high prior-knowledge learners.**

sults consistently indicate that prior knowledge shapes how learners allocate resources and regulate effort during AAT-based programming.

**Syntactic and conceptual difficulties.** Low prior-knowledge learners exhibited more frequent difficulty episodes, particularly in *syntactic* and *conceptual* categories, whereas strategic differences were weaker. Sequence-level analyses suggested that low prior-knowledge learners experienced more extended failure episodes (e.g., repeated syntactic failures) and sustained conceptual difficulty, while high prior-knowledge learners showed fewer difficulty episodes and less complex difficulty transitions overall.

**Lecture-mediated cycle vs. execution-oriented workflow.** Eye-tracking measures revealed that low prior-knowledge learners spent substantially more time consulting learning materials and feedback, including longer lecture viewing, more question rereading, longer code-writing time, and more time inspecting failing outcomes. Transition Network Analysis further showed distinct workflows: high prior-knowledge learners followed a more execution-oriented pathway. Lecture use also differed qualitatively: high prior-knowledge learners more often returned from lecture to the question and were more likely to succeed after lecture consultation.

**Increased arousal and reduced regulation during programming.** The shift from Preparation to Programming elicited a clear physiological change in both groups: sympathetic arousal increased (higher tonic and phasic EDA) and parasympathetic regulation decreased (lower RMSSD), while heart rate remained comparatively stable. Importantly, group differences emerged in *dynamic regulation* rather than average arousal: low prior-knowledge learners showed a larger programming-related increase in phasic SCR and a larger decrease in the Regulation Index, indicating stronger moment-to-moment arousal reactivity and weaker regulation during programming.

**Multimodal analytics based learner model.** Taken together, the multimodal results suggest a coherent pattern: low prior-knowledge learners encountered more basic (syntax/concept) breakdowns, engaged in more material- and feedback-driven cycles, and showed heightened physiological reactivity with larger reductions in regulation during programming. In contrast, high prior-knowledge learners progressed more directly from problem understanding to code execution and success, and their lecture use appeared more selective and integrative. These findings motivate learner models and supports that adapt not only to error types but also to workflow dynamics and regulation states during feedback-intensive programming.

**Implications for adaptive support and learning design.** Our findings may have implications for adaptive support and learning design in programming education. For low prior-

knowledge learners, supports should prioritize (a) syntax-concept remediation (common error patterns, execution semantics checks), and (b) workflow guidance that helps convert lecture consultation into progress (e.g., prompts that connect a specific lecture snippet to a concrete code revision, or structured question rereading before coding). For high prior-knowledge learners, lighter supports that preserve autonomy may suffice (e.g., optional checks for edge cases or output formatting). Physiological findings suggest that regulation signals can complement logs/gaze to flag moments where learners may benefit from brief metacognitive prompts (e.g., pause–plan–test routines) or reduced cognitive load (e.g., simplified feedback, stepwise hints) rather than repeated automatical feedback.

**Limitations.** Several limitations should be noted. First, the sample was relatively small and drawn from a single university, which may limit the generalizability of the findings. Thus, our results should be interpreted as exploratory, process-level evidence from multimodal triangulation in AAT-based programming. Second, categorizing learners into low and high prior knowledge enabled interpretable contrasts but may obscure graded relationships between prior knowledge and multimodal processes. Third, wrist-worn physiological signals can be noisy, highly individualized, and may lag behind task events and therefore our minute-level aggregation and stage-level comparisons may miss event-locked regulation changes.

# 6. CONCLUSION

This work contributes temporally aligned evidence linking difficulty types, attention transitions, and regulation dynamics in an authentic AAT setting. It demonstrates the value of synchronizing AAT logs, eye tracking, and wearable physiology to model programming task solving beyond outcome-based indicators. Our findings motivate process- and regulation-aware learner models and targeted supports (e.g., difficulty-specific scaffolds and workflow guidance) to better assist learners who struggle in AAT-based programming environments.

# 7. ACKNOWLEDGMENTS

This work was supported by JSPS KAKENHI Grant Numbers JP24K20903 and JP25K17078.

# 8. REFERENCES

[1] A. Ahadi. Early identification of novice programmers' challenges in coding using machine learning techniques. In *Proceedings of the 2016 ACM Conference on International Computing Education Research*, pages 263–264, 2016.

[2] K. M. Ala-Mutka. A survey of automated assessment approaches for programming assignments. *Computer science education*, 15(2):83–102, 2005.

[3] E. Albrecht and J. Grabowski. Sometimes it's just sloppiness-studying students' programming errors and misconceptions. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pages 340–345, 2020.

[4] P. Blikstein and M. Worsley. Multimodal learning analytics and education data mining: Using computational technologies to measure complex learning tasks. *Journal of learning analytics*, 3(2):220–238, 2016.

[5] T. Busjahn, R. Bednarik, A. Begel, M. Crosby, J. H. Paterson, C. Schulte, B. Sharif, and S. Tamm. Eye movements in code reading: Relaxing the linear order. In *2015 ieee 23rd international conference on program comprehension*, pages 255–265. IEEE, 2015.

[6] Empatica. EmbracePlus. `https://www.empatica.com/embraceplus/`, 2026. Accessed: 2026-02-07.

[7] D. Fucci, D. Girardi, N. Novielli, L. Quaranta, and F. Lanubile. A replication study on code comprehension and expertise using lightweight biometric sensors. In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, pages 311–322. IEEE, 2019.

[8] H. Gabbay and A. Cohen. Investigating the effect of automated feedback on learning behavior in moocs for programming. *International Educational Data Mining Society*, 2022.

[9] M. N. Giannakos, K. Sharma, I. O. Pappas, V. Kostakos, and E. Velloso. Multimodal data as a means to understand the learning experience. *International Journal of Information Management*, 48:108–119, 2019.

[10] J. Gorson, K. Cunningham, M. Worsley, and E. O'Rourke. Using electrodermal activity measurements to understand student emotions while programming. In *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1*, pages 105–119, 2022.

[11] P. Hejmady and N. H. Narayanan. Visual attention patterns during program debugging with an ide. In *proceedings of the symposium on eye tracking research and applications*, pages 197–200, 2012.

[12] M. Hoq, J. Vandenberg, B. Mott, J. Lester, N. Norouzi, and B. Akram. Towards attention-based automatic misconception identification in introductory programming courses. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2*, pages 1680–1681, 2024.

[13] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä. Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli calling international conference on computing education research*, pages 86–93, 2010.

[14] T. Iwanaga, H. Li, B. Ma, and C. Yin. Visual attention transitions and self-regulated help seeking in programming comprehension. In *Proceedings of the International Conference on Artificial Intelligence in Education 2026 (AIED 2026)*, 2026. In press.

[15] H. Li and B. Ma. Coderunner agent: Integrating ai feedback and self-regulated learning to support programming education. In *Proceedings of the 33rd International Conference on Computers in Education*. Asia-Pacific Society for Computers in Education, 2025.

[16] H. Li, B. Ma, and C. Yin. Examining metacognitive difficulties in learning programming: Analysis of student behavior and strategy. In *Proceedings of the 1st International Conference on Learning Evidence and Analytics*. Asia-Pacific Society for Computers in

Education, 2025.

[17] R. Lobb and J. Harlow. Coderunner: A tool for assessing computer programming skills. *ACM Inroads*, 7(1):47–51, 2016.

[18] S. López-Pernas and M. Saqr. Bringing synchrony and clarity to complex multi-channel data: A learning analytics study in programming education. *IEEE Access*, 9:166531–166541, 2021.

[19] K. Mangaroska, K. Sharma, D. Gaševic, and M. Giannakos. Multimodal learning analytics to inform learning design: Lessons learned from computing education. *Journal of Learning Analytics*, 7(3):79–97, 2020.

[20] J. McBroom, B. Jeffries, I. Koprinska, and K. Yacef. Mining behaviours of students in autograding submission system logs. *International Educational Data Mining Society*, 2016.

[21] T. J. McGill and S. E. Volet. A conceptual framework for analyzing students' knowledge of programming. *Journal of research on Computing in Education*, 29(3):276–297, 1997.

[22] M. Messer, N. C. Brown, M. Kölling, and M. Shi. Automated grading and feedback tools for programming education: A systematic review. *ACM Transactions on Computing Education*, 24(1):1–43, 2024.

[23] N. Milstein and I. Gordon. Validating measures of electrodermal activity and heart rate variability derived from the empatica e4 utilized in research settings that involve interactive dyadic states. *Frontiers in Behavioral Neuroscience*, 14:148, 2020.

[24] S. Mu, M. Cui, and X. Huang. Multimodal data fusion in learning analytics: A systematic review. *Sensors*, 20(23):6856, 2020.

[25] U. Obaidellah, M. Al Haek, and P. C.-H. Cheng. A survey on the usage of eye-tracking in computer programming. *ACM Computing Surveys (CSUR)*, 51(1):1–58, 2018.

[26] X. Ochoa and M. Worsley. Augmenting learning analytics with multimodal sensory data. *Journal of Learning Analytics*, 3(2):213–219, 2016.

[27] H. Ouhaichi, D. Spikol, and B. Vogel. Research trends in multimodal learning analytics: A systematic mapping study. *Computers and Education: Artificial Intelligence*, 4:100136, 2023.

[28] Y. Qian and J. Lehman. Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education (TOCE)*, 18(1):1–24, 2017.

[29] D. D. Salvucci and J. H. Goldberg. Identifying fixations and saccades in eye-tracking protocols. In *Proceedings of the 2000 symposium on Eye tracking research & applications*, pages 71–78, 2000.

[30] M. Saqr, S. López-Pernas, T. Törmänen, R. Kaliisa, K. Misiejuk, and S. Tikka. Transition network analysis: A novel framework for modeling, visualizing, and identifying the temporal patterns of learners and learning processes. In *Proceedings of the 15th International Learning Analytics and Knowledge Conference*, pages 351–361, 2025.

[31] Y. Shi, K. Shah, W. Wang, S. Marwan, P. Penmetsa, and T. Price. Toward semi-automatic misconception discovery using code embeddings. In *LAK21: 11th International Learning Analytics and Knowledge Conference*, pages 606–612, 2021.

[32] L. Silva, A. Mendes, A. Gomes, and G. Fortes. What learning strategies are used by programming students? a qualitative study grounded on the self-regulation of learning theory. *ACM Transactions on Computing Education*, 24(1):1–26, 2024.

[33] M. Worsley, R. Martinez-Maldonado, and C. D'Angelo. A new era in multimodal learning analytics: Twelve core commitments to ground and grow mmla. *Journal of Learning Analytics*, 8(3):10–27, 2021.

[34] S. Xinogalos. Designing and deploying programming courses: Strategies, tools, difficulties and pedagogy. *Education and Information Technologies*, 21(3):559–588, 2016.

[35] W. Yang and U. Obaidellah. Attention dynamics in programming: Eye gaze patterns of high-vs. low-ability novice coders. In *Proceedings of the 2024 Symposium on Eye Tracking Research and Applications*, pages 1–6, 2024.

[36] A. F. Zambrano, J. Zhang, M. Pankiewicz, and R. S. Baker. Practice as the key to success: Understanding the role of prior knowledge, affective states and learning resources in computer science education. In *Proceedings of the LAK26: 16th International Learning Analytics and Knowledge Conference*, pages 85–95, 2026.