Process-Level Analysis of Block-Based Robotics Programming Using Markov Chains

Leon Stjepan Uroic
University of Zagreb Faculty of Electrical Engineering and Computing
leon-stjepan.uroic@fer.hr

Liljana Puskar
University of Zagreb Faculty of Electrical Engineering and Computing
liljana.puskar@fer.hr

Ana Sovic Krzic
University of Zagreb Faculty of Electrical Engineering and Computing
ana.sovic.krzic@fer.hr

ABSTRACT

Understanding student problem-solving strategies in educational robotics is important for developing adaptive instructional support. However, most research in this area focuses on final outcomes rather than the underlying processes. In this exploratory paper, we use first-order Markov chains to model programming strategies used by high school students working with LEGO SPIKE Prime robots. Fine-grained interaction traces are transformed into structured action sequences and compared between low- and high-performing student groups using likelihood-ratio testing and standardized residual analysis. Even in the first task, the Markov chains reveal statistically significant behavioral differences that grow with task complexity, with high-performing students showing more systematic, goal-directed behavior and low-performing students showing more searching, reorganization, and trial-and-error. These exploratory findings suggest that process data from robotics environments can reveal meaningful differences well before final outcomes.

Keywords

Educational robotics, Markov chains, student programming strategies, block-based programming

1. INTRODUCTION

Educational robotics draws heavily on constructivism and Papert's idea that learners develop powerful ideas by building and debugging tangible artifacts [14]. Multiple reviews confirm that robotics activities can improve engagement and learning outcomes [3, 2, 11]. However, the vast majority of studies evaluate these benefits through pre- and post-tests or final solution assessments [13, 20]. This focus on outcomes is at odds with the constructivist premise: in a robotics programming task, iterative debugging and refinement is central, not incidental. Two students can arrive at the same correct solution through very different sequences of exploration, debugging, and self-regulation. An analysis of these processes, rather than only their endpoints, may yield actionable insights for instructors and adaptive systems.

Liljana Puskar, Leon Stjepan Uroic, and Ana Sovic Krzic. Process-Level Analysis of Block-Based Robotics Programming Using Markov Chains. 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. 754–759. International Educational Data Mining Society (2026).

https://doi.org/10.5281/zenodo.21039754

Process-level analysis has gained traction in block-based programming research more broadly. Dong et al. [6] analyzed trace data from Snap! to detect struggling moments and identify when intervention might be needed. Filvà et al. [7] modified Scratch to log every interaction and used those traces to identify behavior profiles and predict performance. Fu et al. [8] showed that process features extracted from MixToy improve student performance prediction. Beyond block-based environments, trace-based analysis has a longer history in text-based programming contexts, where IDE interaction logs have been used to study debugging strategies [10, 4] and to detect productive versus unproductive programming patterns [15]. However, research that applies interaction trace analysis to educational robot programming specifically remains sparse. Scaradozzi et al. [17] collected and analyzed data from LEGO Mindstorms EV3 workshops to extract problem-solving pathways across student teams. We applied clustering to programming trace data from robotics workshops to study students' programming strategies [21].

Given the sequential nature of trace data, Markov chains are a natural modeling choice. Hensen et al. [9] modeled interactions with a math learning platform as a single Markov chain per student. The resulting chains were subsequently clustered to draw conclusions about different behavioral types. Markov chain transition probabilities have also been used as inputs for a neural network trained to predict student performance [16]. Aside from simple Markov models, Hidden Markov Models are also widely used for modeling learning processes [5, 19], with the hope that hidden states can represent higher-order behaviors than raw interaction logs.

In this work, we present an exploratory process-level analysis of students' block-based programming with LEGO Education SPIKE Prime robots [12]. We propose a four-step pipeline: (1) extraction of semantically meaningful action sequences from raw interaction traces, (2) construction of task-specific first-order Markov chains for high- and low-performing groups, (3) likelihood-ratio testing to identify statistically significant group differences, and (4) standardized residual analysis to pinpoint the transitions that contribute most to these differences. **Our contribution is threefold**: (1) we present, to our knowledge, the first application of Markov-chain process analysis to block-based educational robotics programming, extending a method established in screen-only block environments [7, 8] to a tangible-artifact context; (2) we demonstrate that strategy differences are

detectable in the very first task, before any problem-solving demands, suggesting that early process signals can precede outcome signals by an entire workshop; (3) we identify specific transition patterns (run–reorganize–run, remove-and-rebuild, attention-spreading across irrelevant block types) that are candidate features for adaptive support.

2. DATA COLLECTION AND SEQUENCE EXTRACTION

A total of 97 first-year students (average age = 15.22, SD = 0.35) from a public high school participated[1] in the study during regular class hours. Each 90-minute lesson was delivered as part of the existing programming curriculum. Students programmed a pre-built LEGO SPIKE Prime Break Dancer robot in the accompanying block-based environment based on Scratch. The programming environment features a left panel containing available blocks categorized by type and a right-side canvas where programs are constructed by dragging, connecting, modifying, and removing blocks.

Participants completed six programming tasks arranged in increasing difficulty, structured to build foundational concepts while managing cognitive load. Task 1 introduced the programming interface by asking students to copy a simple motor program from a reference image. Task 2 required students to independently program both arm and leg motors. Task 3 asked students to set specific parameter values (e.g., move legs by 375° at 30% speed and arms for 5 s at 50% speed). Tasks 4–6 progressed into simultaneous motor coordination, conditional logic with sensors, and an open-ended capstone activity. Throughout the workshop, we captured mouse and keyboard interaction data, took screenshots at each mouse event, and recorded complete program snapshots following every modification.

The raw data was transformed into action sequences representing students' interactions with the programming environment. We extracted five action types: block ADDED, block REMOVED, block MOVED, block MODIFIED, and RUN. The RUN action was detected via a computer vision algorithm that identified yellow circular UI elements (the run button) in captured screenshots. The remaining actions were derived by comparing abstract syntax trees (ASTs) of consecutive program snapshots. ADDED indicated a new block on the canvas, MODIFIED indicated a parameter change within an existing block, and REMOVED and MOVED captured deletion or repositioning. Because MOVED and REMOVED actions could affect multiple blocks simultaneously (e.g., dragging a root block also moves all subsequent blocks), we aggregated them into single actions while recording the number and types of affected blocks.

After extracting sequences and evaluating each student's solutions, 76 complete instances remained for analysis. The reduction from 97 to 76 resulted from excluding students with incomplete data: those who missed part of the workshop, experienced technical failures in data recording, or whose solutions could not be reliably evaluated. Summary statistics are shown in Table 1.

[1]The study protocol was reviewed and approved by the Ethics Committee of the University of Zagreb Faculty of Electrical Engineering and Computing. Informed consent in accordance with GDPR was obtained from both participating students and their parents or legal guardians prior to the workshop. All interaction data were pseudonymized at collection, and no personally identifying information is retained in the analysis dataset.

Table 1: Summary Statistics of the Data.

Statistic	Value
Average sequence length	280.09 ± 111.52 events
Number of unique states	38
Total transitions counted	45,396
High performers (top 25%)	20 students
Low performers (bottom 25%)	17 students

3. METHOD

Our main goal in this exploratory study is to ascertain whether students' programming process can be meaningfully analyzed using trace data from the LEGO block-based environment. To this end, we categorized students into high-performing (top 25% by assessment score) and low-performing (bottom 25%) groups and tested for statistical differences between their processes. We adopted this extreme-group design rather than a median split because our exploratory aim is to surface qualitatively distinct strategic patterns: trimming the middle 50% maximizes behavioral contrast between groups and reduces noise from students near the score median, who tend to differ less in programming strategy than in incidental factors such as time on task or momentary attention. This design is well suited to identifying which transitions distinguish strategies; estimating population-level effect sizes is left to future work with larger samples. Because students varied substantially in time on task and total number of actions, raw transition counts are not directly comparable across groups. We therefore constructed, for each group and task, a first-order Markov chain in which states corresponded to the action–block-type combinations described above, and based all subsequent analyses on transition probabilities rather than raw counts.

Transition probabilities were estimated by tallying the frequency of each transition across all students within a given group. Although the first-order assumption is clearly simplified, transition probabilities are influenced by many factors beyond the immediately preceding state, more complex models would be impractical given the relatively small sample size per task. The first-order model proved sufficient to capture meaningful patterns.

The MOVED and REMOVED actions presented a methodological challenge, as they could affect multiple block types simultaneously (e.g., moving a motor block that is connected to a control block moves both). To address this, we counted all pairwise combinations of affected states. For instance, if a student moved motor and sensor blocks and subsequently removed event and control blocks, we incremented the count for each combination: *Moved Motor* $\rightarrow$ *Removed Event*, *Moved Motor* $\rightarrow$ *Removed Control*, *Moved Sensor* $\rightarrow$ *Removed Event*, and *Moved Sensor* $\rightarrow$ *Removed Control*. Because this pairwise expansion could in principle inflate transition counts for connected subgraphs, we examine its

impact in Section 4.4 by comparing against an alternative encoding that attributes each composite event only to the block type that the student directly grabbed.

A likelihood-ratio test [1] was used to determine whether observed differences between the two groups were statistically significant rather than attributable to sampling variability. This test is particularly important for early tasks, where transition counts are low due to the simplicity of the activities. Following a significant difference, we computed standardized residuals:

$$r_{ij} = \frac{O_{ij} - E_{ij}}{\sqrt{E_{ij}}}, \tag{1}$$

where O_{ij} and E_{ij} denote observed and expected transition counts under the null hypothesis that both groups share the same underlying Markov chain. Expected counts were computed by pooling all sequences, calculating transition frequencies, normalizing them per state, and multiplying by each group's total transitions. Differences between groups were quantified as:

$$r_{ij}^{\text{diff}} = r_{ij}^{\text{high}} - r_{ij}^{\text{low}}. \tag{2}$$

Standardized residuals quantify the difference between the observed counts and the counts that we would expect to see if both groups shared the same underlying stochastic process. Furthermore, standardized residual analysis accounts for differing group sizes and normalizes transition counts into comparable distributions, allowing it to identify which transitions drive these differences.

Positive values indicate transitions more frequent among high performers; negative values indicate transitions more common among low performers. Following [18], $|r_{ij}^{\text{diff}}| > 2$ is considered notable and $|r_{ij}^{\text{diff}}| > 3$ highly notable.

4. RESULTS

The high- and low-performer groups originate from statistically different Markov chains on all six tasks ($p < 0.01$). Throughout, we interpret the observed transition patterns in terms of the programming environment's mechanics and the task requirements; these interpretations are intended as plausible explanations to guide future research rather than as established causal claims. We restrict the analysis to Tasks 1–3 because these tasks have the most complete data, all 76 students contributed sequences, whereas later tasks are affected by attrition among low-performing students who did not finish the workshop.

4.1 Task 1: Copying a reference program

The standardized residual differences for Task 1 are shown in Figure 1. Although this task involved only copying code from a reference image and required no debugging or problem-solving, the estimated Markov chains for high- and low-performing students differed significantly ($G = 54.44$, $df = 33$, $p < 0.01$).

The largest residual difference corresponds to the transition Modified Motor → Added Motor ($r = -3.04$), indicating that low-performing students frequently added new motor blocks after modifying existing ones. In the programming environment, this transition means a student changed parameters on an existing motor block and then dragged a new motor block from the side menu onto the canvas. Since the task only requires placing the correct blocks and setting values, repeatedly returning to the menu after modifying a block suggests uncertainty about which block is needed. In contrast, high-performing students more often followed the transition Added Motor → Modified Motor ($r = 2.51$), reflecting a more systematic strategy of first selecting the appropriate block and then refining its parameters.

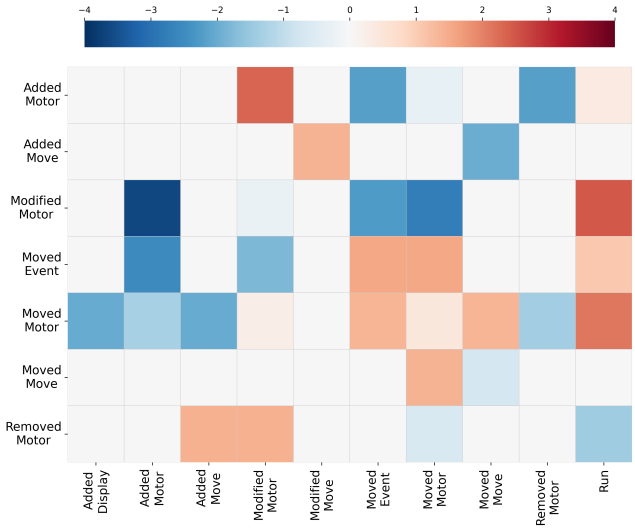

Figure 1: Standardized residual differences (high − low) for Task 1. Rows denote current states and columns denote subsequent states; states with $|r| < 1.0$ are omitted.

Additional low-performer transitions involved event block movements: Moved Event → Added Motor ($r = -2.56$) and Added Motor → Moved Event ($r = -2.09$), suggesting that these students spent time repositioning the program's starting event block alongside their block-searching behavior. Notably, these differences emerge despite the absence of genuine problem-solving demands in the task, suggesting that they reflect differences in familiarity with the interface or approach to the task rather than debugging strategy.

4.2 Task 2: Independent multi-motor programming

Figure 2 shows the standardized residual differences for Task 2 ($G = 62.35$, $df = 40$, $p < 0.01$). In this task, students were asked to independently program both arm and leg motors, and behavioral differences became more pronounced as students began reorganizing their programs.

Low-performing students exhibited substantially more transitions involving movement and modification of move blocks, particularly Added Move → Modified Move ($r = -3.06$) and Modified Move → Moved Move ($r = -3.61$). These patterns indicate frequent restructuring of code: students added move blocks, changed their parameters, and then repositioned them on the canvas. This behavior is consistent with a trial-and-error approach in which students tested partial solutions and subsequently reorganized blocks when the outcome did not match expectations. Importantly, move

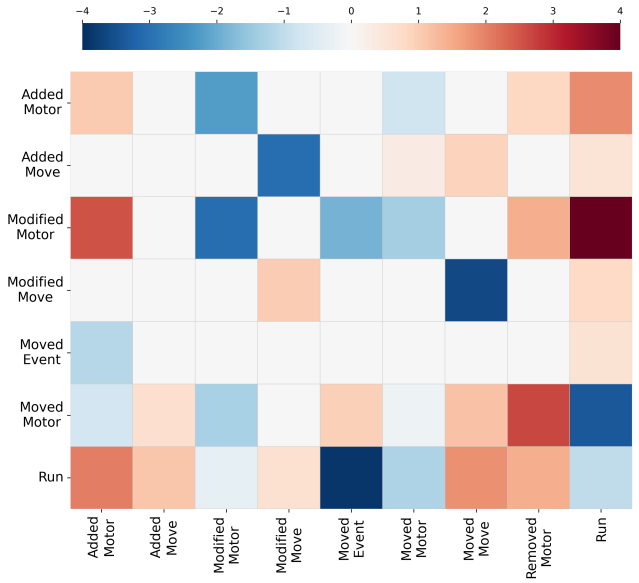

Figure 2: Standardized residual differences (high − low) for Task 2. Rows denote current states and columns denote subsequent states; states with $|r| < 1.0$ are omitted.

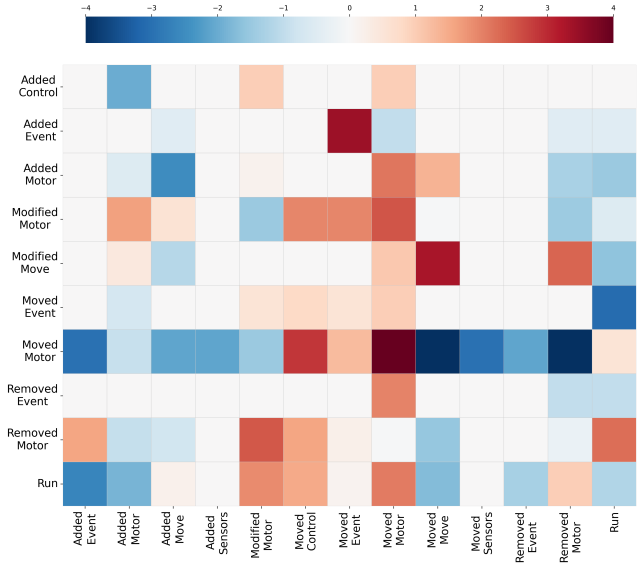

Figure 3: Standardized residual differences (high − low) for Task 3. Rows denote current states and columns denote subsequent states; states with $|r| < 1.9$ are omitted.

blocks were not covered in Task 1, and the subsequent tasks were designed to be solvable using only the previously introduced motor blocks. While Task 2 could technically be solved using move blocks, none of the students successfully did so, meaning these move block manipulations represent unproductive exploration.

A further notable transition was Run → Moved Event ($r = -3.90$), indicating that low-performing students frequently repositioned event blocks immediately after running their programs. Combined with Moved Motor → Run ($r = -3.03$), this pattern suggests a cycle of run–reorganize–run characteristic of unsystematic debugging. High-performing students, by contrast, showed fewer such reorganizations and more direct transitions between adding motor blocks and running (Added Motor → Run, $r = 1.98$; Run → Added Motor, $r = 2.04$), indicating a more stable and anticipatory approach to constructing the program.

4.3 Task 3: Parameter tuning and block ordering

Figure 3 presents the standardized residual differences for Task 3 ($G = 174.54$, $df = 104$, $p < 0.001$). This task required students to set specific motor parameters (e.g., 375° at 30% speed for legs, 5 s at 50% speed for arms), and the common error was placing the speed block after the movement block rather than before it. By this point, differences between groups were both larger in magnitude and more diverse in nature.

A key difference emerged in how students handled block ordering. Low-performing students more often moved motor blocks and then removed them (Moved Motor → Removed Motor, $r = -5.06$), the largest residual for this task, suggesting recognition of incorrect ordering without successfully correcting it through repositioning. Instead, they deleted blocks and re-added them. In contrast, high-performing students more frequently performed consecutive move operations on motor blocks (Moved Motor → Moved Motor, $r = 3.73$), indicating deliberate reordering of the existing program structure rather than removal and reconstruction. High performers also showed a stronger Moved Motor → Run transition ($r = 2.59$), suggesting that after reordering blocks they tested the result, consistent with a hypothesis-driven approach.

Low-performing students were more likely to follow motor movements with structural changes to unrelated block types (Moved Motor → Moved Move, $r = -3.26$; Moved Motor → Moved Sensors, $r = -2.66$; Moved Motor → Added Event, $r = -2.66$), spreading their attention across multiple block categories rather than focusing on the motor blocks central to the task. They also more frequently added motor blocks after running (Run → Added Motor, $r = -2.82$), consistent with a cycle of testing and then starting over with new blocks rather than modifying the existing program.

4.4 Sensitivity to composite-action encoding

To assess whether the pairwise encoding of composite MOVED and REMOVED actions biased the results, we recomputed the Markov chains under a *root-only* alternative in which only the block type that the student directly grabbed contributes to the transition; downstream blocks in the dragged subtree are ignored. This scheme avoids any inflation of transition counts from connected subgraphs at the cost of discarding information about which block types co-occur in composite operations.

The qualitative findings are stable across both encodings. The likelihood-ratio tests remain significant on every task under both schemes (Task 1: $p = 0.011/0.020$; Task 2: $p = 0.013/0.019$; Task 3: both $p < 10^{-4}$), and the ranking, magnitude, and sign of the largest residuals are preserved (Table 2). Across Tasks 1–3, 8–9 of the top-10 transitions per task appear in both schemes; transitions that move in or out are predominantly those near the notability threshold of $|r^{\text{diff}}| = 2$. This indicates that the strategy differences reported in Sections 4.1–4.3 reflect genuine behavioral contrasts rather than artifacts of how composite actions are decomposed.

Table 2: Stability of the largest residuals across encoding schemes (Tasks 1–3). Values are r^{diff} under the pairwise (default) and root-only schemes. The final column indicates whether the transition appears in the top-10 of both schemes for the corresponding task.

T	Transition	Pair.	Root	Top-10
1	Mod. Motor → Add. Motor	−3.04	−3.10	both
1	Add. Motor → Mod. Motor	2.51	2.16	both
1	Mov. Event → Add. Motor	−2.56	−2.00	both
1	Add. Motor → Rem. Motor	−2.09	−2.30	both
1	Add. Motor → Mov. Event	−2.09	−2.30	both
2	Run → Mov. Event	−3.90	−4.02	both
2	Mod. Move → Mov. Move	−3.61	−3.61	both
2	Add. Move → Mod. Move	−3.06	−2.99	both
2	Mov. Motor → Run	−3.03	−2.60	both
2	Mod. Motor → Mov. Motor	−2.70	−2.07	both
3	Mov. Control → Rem. Control	−6.07	−4.82	both
3	Mov. Motor → Rem. Motor	−5.06	−5.37	both
3	Mov. Motor → Mov. Motor	3.73	3.08	both
3	Mod. Move → Mov. Move	3.66	3.61	both
3	Run → Add. Motor	−2.82	−2.85	both
3	Mov. Motor → Run	2.59	2.54	both

5. DISCUSSION

The Markov-chain analysis surfaces three patterns that together sketch a coherent picture of how high- and low-performing students engage with the programming environment. We discuss each in turn, relate it to prior work, and consider its implications for instructional design.

The most striking finding is that Task 1, which required only copying a reference program, already produces statistically significant group differences. Low-performing students repeatedly returned to the block palette after modifying existing blocks (*Modified Motor* → *Added Motor*), while high performers more often followed the reverse sequence. Because the task has no debugging component, the contrast cannot be attributed to differential problem-solving skill; a more plausible reading is that it reflects differences in interface familiarity or in how students plan a sequence of actions before executing it. The practical implication is that process-level signals can precede outcome-level signals by an entire workshop, echoing Jadud's broader argument [10] that trace data serves as a *formative* indicator where assignments and exams only offer *summative* ones, and opening a window for instructional support before students have visibly fallen behind.

A second pattern emerges in Task 2, where low performers exhibit a clear *Run* → *Moved Event* → *Run* cycle together with disproportionate exploration of move-type blocks that none of the students used productively. This is precisely the kind of behavior Dong et al. [6] identify as a "struggling moment" in Snap!: extended periods of activity without meaningful progress, which they argue warrant proactive intervention. Our contribution is to show that these patterns are detectable in a robotics environment with a substantially smaller behavioral vocabulary (38 action–block-type combinations), and that the transitions themselves carry the signal. A natural application is real-time detection: when a student's recent trace shows a high rate of *Run* → *Move* transitions on irrelevant blocks, an adaptive system could surface a reflection prompt asking the student to predict the outcome before the next run.

Task 3 produces the largest residual in the study: low performers recognized incorrect block ordering but resolved it by deletion and reconstruction (*Moved Motor* → *Removed Motor*) rather than repositioning. High performers instead produced consecutive move operations followed by a run, consistent with hypothesis-driven debugging. This mirrors the distinction Piech et al. [15] draw between alpha and gamma trajectory clusters, though their patterns emerge at the level of entire development paths while ours localize to specific transitions. Pedagogically, explicit instruction that blocks can be reordered without being removed could reduce the cost of correction for students who default to deletion. More generally, these transitions are candidate features for the kind of adaptive support that Fu et al. [8] build from aggregate event and sequence features, with the advantage that transition-level signals are directly interpretable as moments where intervention could be inserted.

Taken together, these patterns suggest transition-based indicators that could drive lightweight adaptive support without requiring full sequence modeling, whether through automated prompts or instructor dashboards. The patterns are also entirely behavioral; combining trace data with psychological instruments, eye-tracking, or verbal protocols in future workshops would help disambiguate whether the differences reflect cognitive load, prior experience, or self-regulation.

6. CONCLUSION

This exploratory work-in-progress demonstrates that first-order Markov chains applied to fine-grained programming traces can reveal meaningful behavioral differences between high- and low-performing students in educational robotics, with differences detectable already in the first task, before any problem-solving is required. The consistency of the patterns across tasks suggests Markov chain analysis is a promising tool for surfacing strategic differences well before final outcomes, and the pipeline itself is not specific to the LEGO SPIKE Prime environment but applies to any block-based platform supporting program snapshot extraction. The primary limitation is that data comes from a single workshop context; future work should replicate the analysis across different cohorts and curricula, and evaluate the identified transition patterns as real-time indicators for adaptive instructional support.

7. ACKNOWLEDGMENTS

This work was supported by the Croatian Science Foundation under the project number HRZZ-IP-2022-10-1915 DESCARTES, European Regional Development Fund under grant agreement PK.1.1.10.0007 (DATACROSS). The work of doctoral student Leon Stjepan Uroić has been fully supported by the "Young researchers' career development project – training of doctoral students" of the Croatian Science Foundation DOK-2025-02-8664.

8. REFERENCES

[1] T. W. Anderson and L. A. Goodman. Statistical inference about markov chains. *The Annals of Mathematical Statistics*, 28(1):89–110, 1957.

[2] S. Anwar, N. A. Bascou, M. Menekse, and A. Kardgar. A systematic review of studies on educational robotics. *Journal of Pre-College Engineering Education Research (J-PEER)*, 9(2):Article 2, 2019.

[3] F. B. V. Benitti. Exploring the educational potential of robotics in schools: A systematic review. *Computers & education*, 58(3):978–988, 2012.

[4] P. Blikstein. Using learning analytics to assess students' behavior in open-ended programming tasks. In *Proceedings of the 1st international conference on learning analytics and knowledge*, pages 110–116, 2011.

[5] K. Damevski, H. Chen, D. Shepherd, and L. Pollock. Interactive exploration of developer interaction traces using a hidden markov model. In *Proceedings of the 13th International Conference on Mining Software Repositories*, pages 126–136, 2016.

[6] Y. Dong, S. Marwan, P. Shabrina, T. Price, and T. Barnes. Using student trace logs to determine meaningful progress and struggle during programming problem solving. *International Educational Data Mining Society*, 2021.

[7] D. A. Filvà, M. A. Forment, F. J. García-Peñalvo, D. F. Escudero, and M. J. Casañ. Clickstream for learning analytics to assess students' behavior with scratch. *Future Generation Computer Systems*, 93:673–686, 2019.

[8] Q. Fu, W. Tang, Y. Zheng, H. Ma, and T. Zhong. Predicting programming performance by using process behavior in a block-based programming environment. *Interactive Learning Environments*, 32(6):2371–2385, 2024.

[9] C. Hansen, C. Hansen, N. Hjuler, S. Alstrup, and C. Lioma. Sequence modelling for analysing student interaction with educational systems. *arXiv preprint arXiv:1708.04164*, 2017.

[10] M. C. Jadud. Methods and tools for exploring novice compilation behaviour. In *Proceedings of the second international workshop on Computing education research*, pages 73–84, 2006.

[11] S. E. Jung and E.-s. Won. Systematic review of research trends in robotics education for young children. *Sustainability*, 10(4):905, 2018.

[12] LEGO Education. Lego education spike prime set 45678. Robotics kit, 2020. Available at: `https://education.lego.com/en-us/products/lego-education-spike-prime-set/45678`, accessed February 9, 2026.

[13] S. Lu, Z. Nuryana, X. Ni, W. Xu, and M. N. Alias. A decade of educational robotics: trends and sdg contributions. *Humanities and Social Sciences Communications*, 12(1):1–16, 2025.

[14] S. A. Papert. *Mindstorms: Children, computers, and powerful ideas*. Basic books, 2020.

[15] C. Piech, M. Sahami, D. Koller, S. Cooper, and P. Blikstein. Modeling how students learn to program. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 153–160, 2012.

[16] N. Rohani, K. Gal, M. Gallagher, and A. Manataki. Early prediction of student performance in a health data science mooc. In M. Feng, T. KÃ¤ser, and P. Talukdar, editors, *Proceedings of the 16th International Conference on Educational Data Mining*, pages 325–333, Bengaluru, India, July 2023. International Educational Data Mining Society.

[17] D. Scaradozzi, L. Cesaretti, L. Screpanti, and E. Mangina. Identification of the students learning process during education robotics activities. *Frontiers in Robotics and AI*, 7:21, 2020.

[18] D. Sharpe. Your chi-square test is statistically significant: Now what? *Practical Assessment, Research & Evaluation*, 20(8):1–10, 2015.

[19] B. Shih, K. R. Koedinger, and R. Scheines. Unsupervised discovery of student learning tactics. In *Proceedings of the 3rd International Conference on Educational Data Mining*, pages 201–210, 2010.

[20] N. Spolaôr and F. B. V. Benitti. Robotics applications grounded in learning theories on tertiary education: A systematic review. *Computers & Education*, 112:97–107, 2017.

[21] L. S. Uroić, L. Puskar, and A. Sovic Krzic. Preliminary analysis of student programming strategies in an educational robotics workshop. In *16th IEEE Integrated STEM Education Conference*, 2026.