Proceedings of the
13th International Conference on Applied Informatics
Eger, Hungary, February 19–21, 2026
DOI: 10.17048/icai.2026.202
pp. 202–215

AI-Based Estimation of the Zone of Proximal Development in Programming Education

Bendegúz Nagy, Győző Horváth

Department of Media & Educational Informatics,
Eötvös Loránd University, Budapest, Hungary
brandy@inf.elte.hu
gyozke@inf.elte.hu

Abstract. The *Zone of Proximal Development* (ZPD), introduced by Vygotsky [11], describes the range of learning tasks that learners can successfully complete with appropriate support. Closely related to this concept, the notion of flow experience emphasizes the balance between task challenge and learner skill as a key condition for sustained engagement and intrinsic motivation [1].

Recent research suggests that artificial intelligence (AI) can support ZPD-oriented adaptation in modern educational environments by dynamically adjusting task difficulty and feedback [10]. However, practical and lightweight implementations of such approaches in programming education remain limited.

This paper presents a proof-of-concept AI-based system for estimating learners' ZPD using a minimal set of background information and task interaction data. The approach records simplified learner profiles from task interaction data – task success, error frequency, and completion time – and derives a skill estimate from task success and difficulty; this estimate defines a difficulty band that a large language model uses to generate subsequent programming tasks at an adjusted difficulty level.

The system is developed as part of the continued evolution of the *Codia* gamified programming education platform, building on earlier findings related to student motivation in secondary school programming education [6]. The design focuses on feasibility and pedagogical plausibility rather than predictive accuracy. The study examines whether a simplified AI-supported approach can approximate ZPD-aligned task selection in introductory programming education.

Keywords: zone of proximal development, adaptive learning, ELO rating, large language models, programming education, task generation

2020 *Mathematics Subject Classification:* Primary 97P30; Secondary 68T50

1. Introduction

Programming education at the secondary school level faces a structural challenge: teachers must design tasks for an entire class, yet individual students differ substantially in prior experience, cognitive pace, and intrinsic motivation. Assigning identical tasks to all learners inevitably places some students below their challenge threshold – risking boredom and disengagement – and others above it – risking frustration and dropout. Addressing this mismatch requires both a theoretical model of optimal challenge and a practical mechanism for adapting task difficulty to individual learners in real time.

Vygotsky's concept of the *Zone of Proximal Development* (ZPD) provides exactly such a model [11]. The ZPD denotes the gap between what a learner can achieve independently and what they can achieve with guidance; instructional tasks that fall within this zone maximise learning efficiency. Closely related is Csikszentmihalyi's theory of *flow*: sustained engagement arises when both task difficulty and individual skill are simultaneously high [1]. Together, ZPD and flow theory motivate an educational design in which each student receives tasks calibrated to their evolving competence.

Recent developments in large language models (LLMs) open a practical path towards such personalisation: LLMs can generate contextually appropriate programming tasks conditioned on skill level, topic preferences, and prior performance [9], yet lightweight end-to-end implementations in introductory programming remain scarce. The proof-of-concept system described here, *Codia Path*, is built using the T3 Stack (Next.js, tRPC, Prisma, PostgreSQL) with React Flow for graph visualisation and supports OpenAI (`o4-mini`), Anthropic Claude, and Google Gemini. Throughout, we distinguish two systems: *Codia* is the deployed platform used in the classroom pilot (Section 3), whereas *Codia Path* is the separate proof-of-concept adaptive system evaluated here on simulated data (Section 5); the pilot motivates but does not validate Codia Path.

This paper makes the following contributions:

- We report the results of a real-world pilot study with the *Codia* gamified programming platform, motivating a shift from extrinsic-reward-based gamification to ZPD-centred task design [6].
- We describe the design of *Codia Path*, a proof-of-concept system that combines a graph-based curriculum model, an ELO-inspired ZPD estimator, and LLM-based task generation to produce personalised programming tasks.
- We illustrate the system using three simulated learner trajectories and qualitatively assess the pedagogical plausibility of the generated tasks; we make no claim of improved learning outcomes.

2. Background and Related Work

Vygotsky's *Zone of Proximal Development* (ZPD) denotes the distance between what a learner can achieve independently and what they can achieve with guidance [11]. In a software-based learning environment the ZPD can be operationalised through measurable proxies – response accuracy, error count, and time-on-task – all of which are naturally observable in a programming IDE. Closely related is Csikszentmihalyi's *flow* theory: optimal experience arises when both perceived challenge and perceived skill are high [1], a finding confirmed by meta-analytic evidence [3]; when this balance is absent, the result is either boredom (low challenge) or anxiety (excessive challenge) [7].

Gamification has been widely studied as a motivational tool in education. Hamari et al. [4] report generally positive effects on engagement, but note that outcomes are greatly dependent on context. Koivisto and Hamari [5] confirm that mixed results are remarkably common, underscoring the importance of design quality and situational fit. Palmquist [8] further shows that the success of gamification in education is mediated by factors such as teacher buy-in and implementation quality.

AI-supported adaptive learning ranges from classical intelligent tutoring systems [10] to modern LLM-based approaches. Sarsa et al. [9] demonstrate automatic generation of programming exercises and code explanations using LLMs, though rigorous classroom evaluation of such methods remains limited. Yang et al. [12] combine LLM-derived difficulty features with deep knowledge tracing, improving difficulty prediction accuracy in programming education. Our work focuses on *lightweight* operationalisation: Codia Path records per-attempt interaction data (success, error count, and completion time) and derives a self-calibrating skill estimate from task success and difficulty using an ELO-inspired rating rule [2]; the resulting difficulty band is then supplied to an LLM prompt that generates the next task.

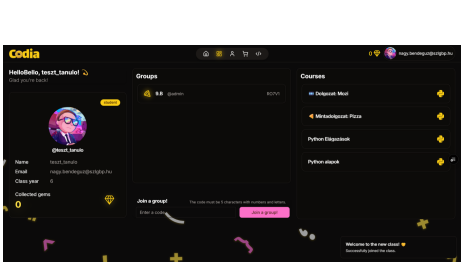
3. The Codia Platform and Pilot Study

3.1. Platform Description

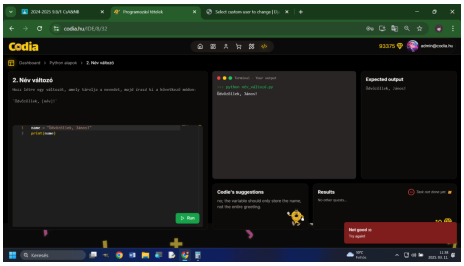
Codia (available at `https://codia.hu`) is a web-based, mobile-optimised programming education platform for introductory Python programming at the secondary school level. Its main features include:

- **Online IDE:** A browser-based Python interpreter optimised for mobile and desktop use.
- **AI code feedback (Codie):** An LLM-based assistant that detects errors and provides hints (Figure 1).

- **Gamification:** A diamond-based reward system where correct solutions earn virtual currency exchangeable for rewards.
- **Teacher dashboard:** Task assignment creation, progress monitoring, and group management.



(a) Student dashboard.



(b) AI code feedback (Codie).

Figure 1. The Codia programming education platform.

3.2. Pilot Study

We conducted a five-week deployment of Codia at *Kőbányai Szent László Gimnázium*, Budapest, with 17 ninth-grade students [6]. Students solved teacher-assigned tasks in the online IDE and received AI feedback from Codie. At the end of the study, 12 students completed a Likert-scale questionnaire (Figure 2).

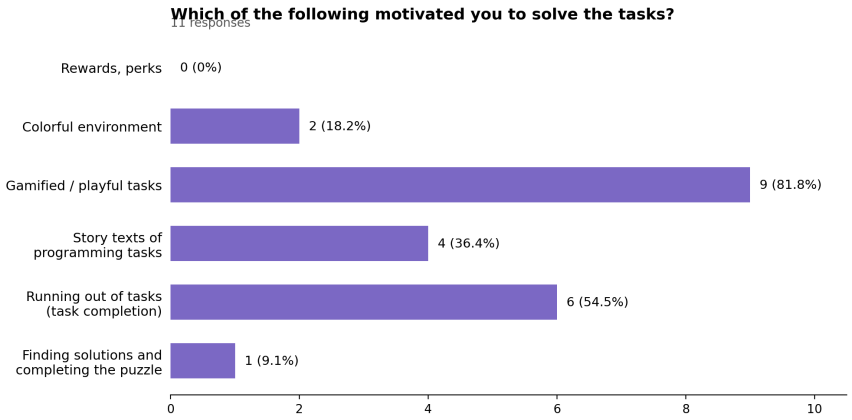


Figure 2. Student attitudes toward programming and Codia ($n = 12$).

Key findings: 10/11 students preferred Codia over prior coding environments; Codie's feedback helped students self-correct code independently; the reward/dia-

mond system was rarely used; and not a single student selected rewards as their primary motivator – the dominant motivators were *problem-solving challenge* and *task completion*. These results, consistent with flow theory [7], directly motivated the design of Codia Path.

4. System Design: Codia Path

Codia Path is a proof-of-concept web application built on top of the data infrastructure of Codia. It implements three interconnected subsystems: (1) a graph-based curriculum model, (2) an ELO-based ZPD estimator, and (3) an LLM-driven task generator.

4.1. Graph-Based Curriculum Model

Rather than presenting courses as a flat list, Codia Path models the curriculum as a *directed acyclic graph* (DAG) of courses. A directed edge from course A to course B indicates that A is a prerequisite for B. The current curriculum graph covers eight courses, from *Variables & Types* to *Recursion & Algorithms*. In the future, the topics and competence areas defined in the Hungarian National Core Curriculum (NAT) could serve as a solid basis for structuring these courses.

Each course additionally contains a *task dependency graph*: a second DAG in which nodes represent individual programming tasks and edges indicate which earlier tasks must be attempted before a given task becomes available to a learner. Figure 3 shows an example: nodes represent tasks, edges encode prerequisites, and colours indicate attempt outcome. A learner's personal instance of this graph is annotated with the outcome of each prior attempt, enabling the visualisation of their individual learning trajectory within the course.

4.2. Learning Objectives

Each course is associated with a set of *learning objectives* (e.g., "Declare and initialise variables", "Use basic data types", "Read user input with `input()`"). The objectives and topics prescribed by the NAT could provide a useful foundation for defining these in a real deployment. They serve as a progress checklist for learners (Figure 4) and are included verbatim in the LLM prompt during task generation, ensuring that generated tasks target specific learning goals.

4.3. ELO-Inspired ZPD Estimator

After each recorded attempt, Codia Path recomputes the learner's skill estimate from their attempt history using a bounded, rating-style rule inspired by ELO systems [2]: instead of a fixed reference scale, competence is estimated by comparing attempt outcomes against the difficulty of the attempted tasks.

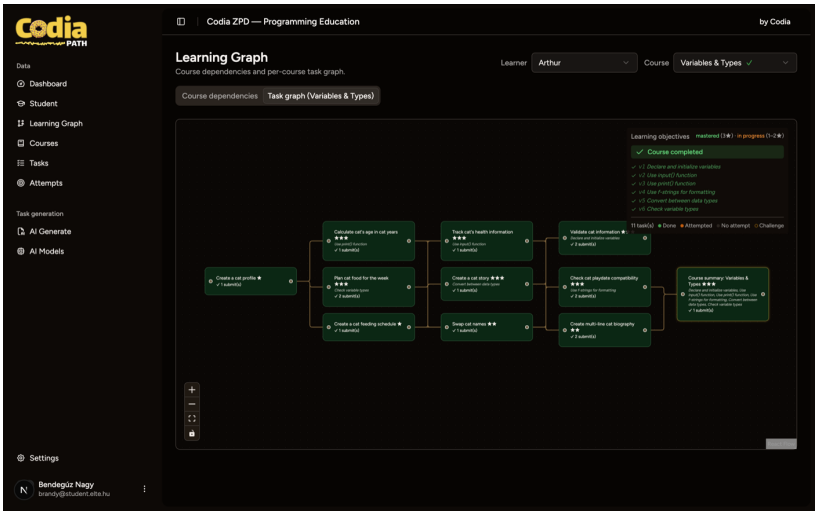


Figure 3. Task dependency graph for *Variables & Types*.

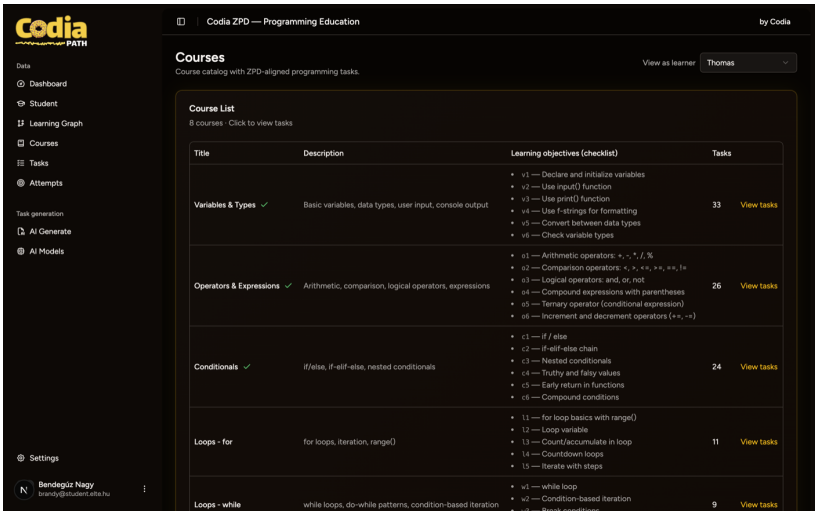


Figure 4. Learning objectives and mastery tracking.

Let a learner have n recorded attempts, where attempt i has difficulty $d_i \in \{1, 2, 3\}$ and outcome $o_i \in \{0, 1\}$ ($1 =$ success). Each attempt contributes a difficulty-anchored target t_i and a weight w_i,

$$t_i = \begin{cases} d_i & \text{if } o_i = 1, \\ d_i - \beta & \text{if } o_i = 0, \end{cases} \qquad w_i = \begin{cases} w^+ & \text{if } o_i = 1, \\ w^- & \text{if } o_i = 0, \end{cases}$$

so that a success anchors the estimate to the task's difficulty while a failure pulls it slightly below. The skill estimate is the weighted mean of these targets, clamped to the difficulty scale:

$$S = \mathrm{clip}\left(\frac{\sum_{i=1}^{n} w_i \, t_i}{\sum_{i=1}^{n} w_i}, \; 1, \; 3 \right),$$

with $w^+ = 1.2$, $w^- = 0.6$, and $\beta = 0.4$ in the current implementation. Successful attempts thus carry more weight than failures, and the estimate remains within the $[1, 3]$ difficulty range by construction, avoiding the unbounded drift and scale-calibration problems that a raw ELO K-factor update would introduce on such a short scale. The estimate depends only on the outcome o_i and the difficulty d_i; per-attempt error count and completion time are stored in the learner profile as descriptive statistics (average error count, average completion time) but do *not* enter the skill estimate in the current prototype. The prototype maintains a single skill estimate per learner, aggregated over all attempted courses; per-course and per-objective estimates are natural extensions discussed in Section 7.

ZPD band. The ZPD band for a learner is the interval $[S - \delta, \; S + \delta]$ intersected with the difficulty scale $[1, 3]$, where $\delta = 0.5$ by default. This band is passed to the LLM as a *soft* constraint on the difficulty of the generated task; Section 4.4 describes how an explicit target difficulty can override it.

Worked example. A learner who has succeeded on two difficulty-2 tasks and failed one difficulty-3 task obtains

$$S = \frac{1.2 \cdot 2 + 1.2 \cdot 2 + 0.6 \cdot (3 - 0.4)}{1.2 + 1.2 + 0.6} = \frac{6.36}{3.0} \approx 2.12,$$

yielding a ZPD band of $[1.62, \; 2.62]$.

Parameter choices. Table 1 lists the estimator and generation parameters with their default values and rationale.

4.4. AI Task Generation

The system assembles a structured prompt containing:

1. **Learner trajectory:** The learner's recent attempt history, each entry listing the task title, its difficulty level, and the success/failure outcome.

Table 1. Estimator and generation parameters (current implementation).

Parameter	Value	Rationale
Success weight w^+	1.2	Successes weigh more than failures.
Failure weight w^-	0.6	Failures still inform the estimate, but less.
Failure offset β	0.4	A failed task pulls the estimate slightly below its difficulty.
Band half-width δ	0.5	Width of the ZPD band around the skill estimate.
Difficulty scale	1–3	Manually assigned per-task difficulty labels.
Skill range (clip)	$[1, 3]$	Keeps the estimate on the difficulty scale.

2. **ZPD band:** The numerical difficulty interval [*zpdMin*, *zpdMax*] computed as described in Section 4.3.

3. **Course learning objectives:** The full list of objectives for the selected course, annotated with the learner's current mastery status.

4. **Interest keywords:** Thematic tags provided by (or assigned to) the learner, e.g., `space`, `cooking`, `sports`. The LLM incorporates these as context and narrative elements, personalising the task scenario without changing its computational content.

5. **Generation history:** Previous AI-generated tasks for this learner, with any recorded feedback, so that the model avoids repetition.

The LLM is instructed to return a structured JSON object containing: task title, description, starter code, sample solution, difficulty estimate (1–3), targeted learning objectives, and a brief *pedagogical rationale* explaining why this task is appropriate for the learner's current ZPD.

Target difficulty and scaffolding. The ZPD band acts as a *soft* constraint. Alongside it, the generator accepts an explicit target-difficulty control (a 1–3 rating together with a challenge preference such as *easier*, *steady*, or *challenge*), and the LLM is instructed to match the requested target difficulty. When a learner first enters a new course, the system requests a difficulty-1 scaffolding task to introduce the topic gently, even if the learner's skill-derived band lies higher; in this case the requested target difficulty deliberately overrides the lower bound of the band. This mechanism explains the difficulty-1 task generated for Arthur in Section 5 despite a band of $[2.09, 3.00]$.

5. Evaluation with Simulated Learners

The classroom pilot in Section 3 evaluated the *Codia* platform, not Codia Path; the two systems are distinct, and the pilot motivates but does not validate the adaptive method proposed here. In the absence of real-world classroom data from Codia Path (the system is not yet integrated with the production Codia platform), we report an *illustrative simulation study* – not a controlled experiment – using three *simulated learner trajectories* constructed with the assistance of Claude Opus 4.

5.1. Simulated Trajectories

Three learner personas were designed to represent distinct learning profiles typical of secondary school programming classes:

Thomas Motivated beginner, no prior experience; progresses slowly with frequent errors. Seeded: dog-themed tasks, *Variables & Types* through *For Loops* (42 attempts).

Arthur Moderate prior experience (Scratch, Micro:bit); average pace, occasionally revisits earlier concepts. Seeded: cat-themed tasks across six courses up to *Functions – Basics* (54 attempts).

John Fast learner with strong Python background; few errors, consistently targets higher difficulty. Seeded: skateboard-themed tasks, *Variables & Types* and *Operators & Expressions* (16 attempts).

Success rates, error counts, and completion times were set consistently with each persona's profile. Interest keywords are supplied at generation time; for the three demonstrated tasks we used `Dune` for Thomas, `M&M's` for Arthur, and `watermelon` for John.

5.2. Generated Tasks and Results

For each simulated learner, we triggered the AI Generate module after the full trajectory had been recorded, targeting courses at or near the learner's current frontier. Table 2 summarises the key parameters and the resulting generated tasks.

5.3. Detailed Task Descriptions

Below we reproduce the AI-generated task descriptions for each generation in Table 2, assessed along three dimensions: difficulty appropriateness, objective alignment, and narrative coherence.

Table 2. Generation inputs and AI-generated tasks for the three illustrated examples. All generated with `o4-mini` (OpenAI). Column **D** is the requested target difficulty; Arthur's D = 1 is a deliberate new-course scaffolding override of the ZPD band (Section 4.4), not a constraint violation.

Learner	Course	Skill	ZPD band	D	Generated task
Thomas	For Loops	2.47	[1.97, 2.97]	2	*Spice Harvest Accumulator*
Arthur	Arrays & Iter.	2.59	[2.09, 3.00]	1	*Accessing M&M Colors in a List*
John	Oper. & Expr.	2.73	[2.23, 3.00]	3	*Watermelon Juice Estimator*

Thomas: *Spice Harvest Accumulator* (D2, keywords: "Dune (the movie)").

> House Atreides needs your help on Arrakis to keep track of the melange (spice) they collect. Write a program that: (1) asks the user for the number of days of harvesting (an integer); (2) for each day from 1 to that number, prompts the user to enter the pounds of spice harvested on that day and adds that amount to a running total; (3) after all days have been processed, prints the total in the format: `"Total spice harvested: X pounds"`.

The task targets "count and accumulate values in loops" at difficulty 2, within Thomas's ZPD band [1.97, 2.97]. The Dune narrative reflects the supplied keyword, and the accumulation-loop structure suits a persistent beginner entering the loops course (Figure 5).

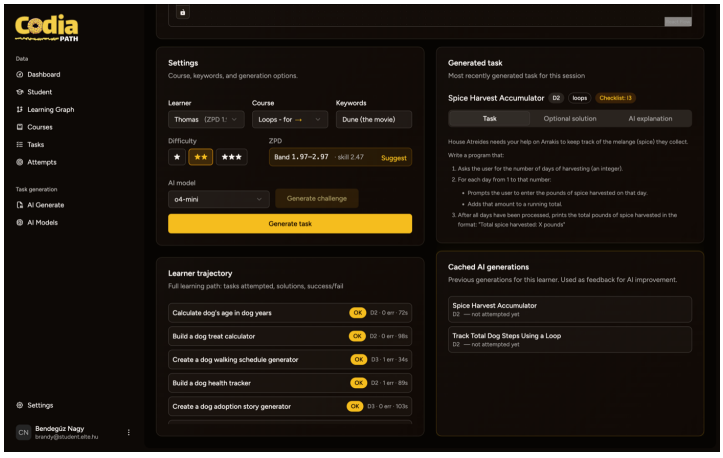


Figure 5. Thomas's generated task, illustrating how a beginner's ZPD band and interest keywords produce a contextually appropriate accumulation-loop exercise at difficulty 2.

Arthur: *Accessing M&M Colors in a List* (D1, keywords: "M&M's").

> Create a list named `mm_colors` with these M&M colors in this exact order: red, blue, green, yellow, orange. Then: (1) print the first color; (2) print the third color; (3) print the fifth color. Use list indexing (with indices 0, 2, and 4) and f-strings to format your output.

Arthur had completed six courses and was entering *Arrays & Iteration*. Although his skill-derived ZPD band was $[2.09, 3.00]$, the generator was asked for a difficulty-1 task because this was his first attempt in a new course: entering a new topic triggers a scaffolding override that introduces the concept with a low-pressure task (Figure 6), regardless of the higher global skill band. The M&M theme personalises the exercise without altering its core requirement – list creation and index access.

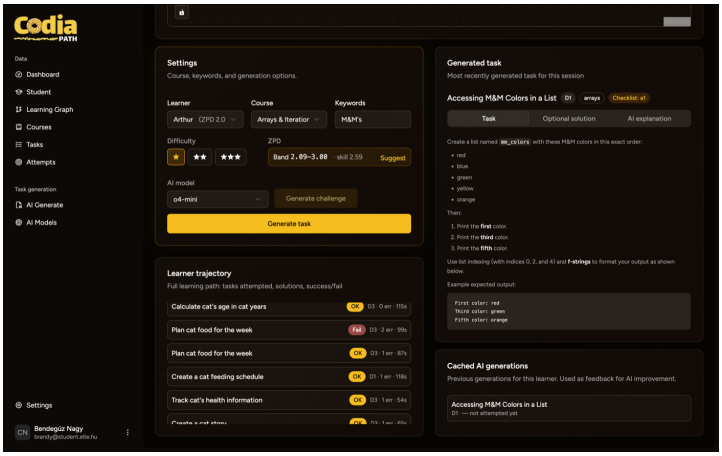


Figure 6. Arthur's generated task, showing that the system selects difficulty 1 when a learner enters a new course, producing a gentle, interest-themed introduction to list indexing.

John: *Watermelon Juice Estimator with Parentheses* (D3, keywords: "watermelon").

> Write a program that helps a watermelon vendor estimate how much juice they can get: (1) prompt the user to enter the weights (kg) of three sample watermelons; (2) prompt for the total number of watermelons; (3) compute the **average weight** – use parentheses to group the sum before dividing by 3; (4) compute the **total juice** in litres (1 kg yields 0.9 l) – use parentheses for correct operator precedence; (5) print both values formatted to two decimal places.

This targets "build compound expressions using parentheses" at difficulty 3, matching John's skill of 2.73 (Figure 7). The multi-step arithmetic with explicit parenthesisation challenges an advanced learner.

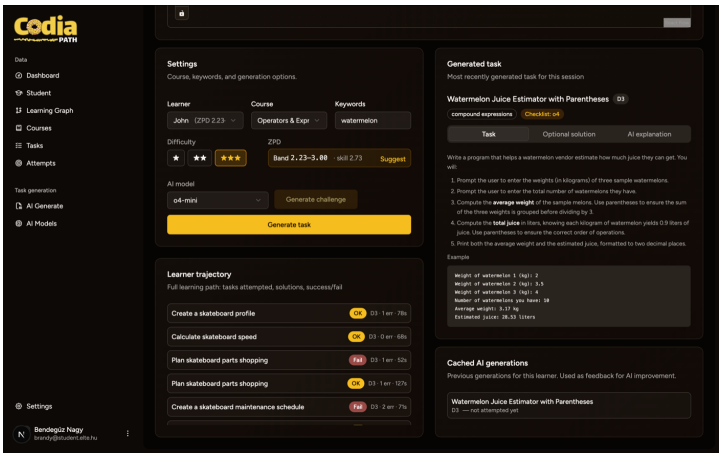


Figure 7. John's generated task, demonstrating that a high skill score shifts the ZPD band upward, prompting the LLM to produce a difficulty 3 compound-expression task.

5.4. Discussion

Two of the three generated tasks (Thomas and John) fell strictly within the computed ZPD band; Arthur's difficulty-1 task was a deliberate new-course scaffolding override rather than a constraint violation (Section 4.4). Each task incorporated the supplied interest keywords naturally. The three trajectories produced differentiated skill values (2.47, 2.59, 2.73), and the LLM responded with correspondingly differentiated structures – from a scaffolded list-indexing exercise (D1) for a learner entering a new course to a multi-step arithmetic task with explicit parenthesisation (D3) for an advanced learner. These illustrative results are consistent with the working hypothesis that even a lightweight ZPD estimator can guide differentiated task personalisation; they do not, however, constitute evidence of improved learning outcomes, which would require a controlled study with real learners.

6. Conclusions

We have presented Codia Path, a proof-of-concept system that combines a graph-based curriculum model, an ELO-inspired ZPD estimator, and LLM-based task generation into a lightweight pipeline for personalised programming education. A pilot study with 17 students of the *Codia* platform indicated that intrinsic challenge – rather than gamification rewards – drives motivation, which motivated the design of Codia Path; an illustrative evaluation of Codia Path with three simulated learner profiles produced pedagogically plausible tasks. We emphasise that these are proof-of-concept results: we do not yet claim improved learning outcomes relative to a baseline, which remains future work.

7. Limitations and Future Work

The evaluation has several important limitations:

- **Simulated, LLM-assisted data.** The system has been exercised only with three simulated learner trajectories, themselves constructed with LLM assistance, rather than with real classroom data; this carries a risk of circularity and cannot demonstrate learning gains.
- **No baseline comparison.** We do not compare ZPD-personalised sequences against teacher-assigned tasks or any other baseline.
- **No automated correctness verification.** Generated starter code and reference solutions were not checked by an automated test harness, and LLM-generated tasks occasionally required minor syntactic corrections.
- **Author-only task review.** Generated tasks were assessed by the authors along difficulty appropriateness, objective alignment, and narrative coherence; independent expert (teacher) review is planned but not yet performed.
- **Coarse, single-scalar skill model.** The estimator collapses multidimensional competence into a single skill estimate per learner on a discretised 1–3 difficulty scale, with manually assigned difficulty labels.
- **Unused interaction signals.** Error count and completion time are recorded but do not yet influence the skill estimate or task generation.

More broadly, LLM-based generation carries inherent risks – including factual inaccuracies (hallucinations), implicit biases in task framing, and potential over-reliance that may weaken learners' independent problem-solving – all of which necessitate expert review of generated content before classroom deployment.

The immediate priority is integration into the production Codia platform and a controlled classroom experiment comparing ZPD-personalised sequences against teacher-assigned tasks [7]. Further directions include per-course and per-objective skill estimates, incorporating error count and completion time into the estimator, expanded course coverage, and collaborative ZPD estimation through peer comparison.

References

[1] M. Csikszentmihalyi: *Flow: The Psychology of Optimal Experience*, New York: Harper & Row, 1990.

[2] A. E. Elo: *The Rating of Chessplayers, Past and Present*, New York: Arco, 1978.

[3] C. J. Fong, D. J. Zaleski, J. K. Leach: *The Challenge–Skill Balance and Antecedents of Flow: A Meta-Analytic Investigation*, The Journal of Positive Psychology 10.5 (2015), pp. 425–446, doi: 10.1080/17439760.2014.967799.

[4] J. Hamari, J. Koivisto, H. Sarsa: *Does Gamification Work? – A Literature Review of Empirical Studies on Gamification*, in: Proceedings of the 47th Hawaii International Conference on System Sciences (HICSS), IEEE, 2014, pp. 3025–3034, doi: 10.1109/HICSS.2014.377.

[5] J. Koivisto, J. Hamari: *The Rise of Motivational Information Systems: A Review of Gamification Research*, International Journal of Information Management 45 (2019), pp. 191–210, doi: 10.1016/j.ijinfomgt.2018.10.013.

[6] B. Nagy, G. Horváth: *A gamifikáció hatása a tanulói motivációra egy középiskolai csoport programozásóráin*, in: InfoDidact 2025 – Az informatika tanításának módszertana, Tanulmánykötet, Budapest, Hungary: Eötvös Loránd University, 2025, isbn: 978-963-489-851-1, url: https://infodidact.elte.hu/2025/infodidact2025_tanulmanykotet.pdf.

[7] J. Nakamura, M. Csikszentmihalyi: *The Concept of Flow*, in: Handbook of Positive Psychology, ed. by C. R. Snyder, S. J. Lopez, Oxford: Oxford University Press, 2002, pp. 89–105, doi: 10.1093/oxfordhb/9780195187243.013.0018.

[8] A. Palmquist: *"Gamification was not the problem": A Case Study Exploring Factors Affecting Teachers' Approval of Gamification*, in: Proceedings of the 24th International Academic Mindtrek Conference (Mindtrek '21), ACM, 2021, pp. 106–116, doi: 10.1145/3464327.3464347.

[9] S. Sarsa, P. Denny, A. Hellas, J. Leinonen: *Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models*, in: Proceedings of the 2022 ACM Conference on International Computing Education Research (ICER '22), ACM, 2022, pp. 27–43, doi: 10.1145/3501385.3543957.

[10] K. VanLehn: *The Relative Effectiveness of Human Tutoring, Intelligent Tutoring Systems, and Other Tutoring Systems*, Educational Psychologist 46.4 (2011), pp. 197–221, doi: 10.1080/00461520.2011.611369.

[11] L. S. Vygotsky: *Mind in Society: The Development of Higher Psychological Processes*, Cambridge, MA: Harvard University Press, 1978, doi: 10.2307/j.ctvjf9vz4.

[12] L. Yang, X. Sun, H. Li, R. Xu, X. Wei: *Difficulty Aware Programming Knowledge Tracing via Large Language Models*, Scientific Reports 15 (2025), p. 11475, doi: 10.1038/s41598-025-96540-3.