Early Prediction of Later Struggle in Introductory Programming: Leakage, Time-Aware Validation, and Cross-Section Transfer

Bilal Mubarak Idris[1], Sodiq Sanusi[2]

[1]Iconic Open University, Abuja, Nigeria

[2]Department of Computer, Information Technology & Software Engineering, Iconic Open University, Abuja, Nigeria

[1]ORCID: 0009-0008-9760-5095

[2]ORCID: 0009-0001-2683-9993

DOI: https://doi.org/10.51244/IJRSI.2026.1311000042

Received: 01 November 2026; Accepted: 06 November 2026; Published: 21 November 2026

ABSTRACT

Feedback that reaches a novice programmer only after repeated failure comes too late to help. This study asks whether later struggle on a programming assignment can be anticipated from behavioral traces available early, and whether such predictions survive time-aware validation and a change of cohort or assignment set. We analysed 69,189 submissions from 411 students in two sections of an introductory Java course (CodeWorkout, Spring 2019). Whole-history aggregation is not early prediction: with success rate as the label, six behavioral features reached an AUC of 0.778, adding mean score raised it to 0.934, and restoring the label-defining success rate gave 0.99 or more, which demonstrates target leakage. Naive event filtering can also manufacture such a label, flagging 59.5% of students instead of 21.7%. A truncated-history design instead takes features only from a student's first k submissions on an assignment, takes the label only from later submissions, and uses rolling-origin validation over five weekly assignments. Early prediction was possible but weak: with history and code-level features, AUC was 0.60 to 0.68 (95% confidence intervals excluding 0.5 at every k) and PR-AUC was 0.26 to 0.32 against a prevalence of 0.16 to 0.22. A cost-tuned model was significantly cheaper than the best trivial policy only when a missed student cost five times a needless flag, and only at some k. Models transferred between concurrent sections about as well as they performed within a section (AUC about 0.70) but transferred unreliably to later assignments, and adding target-section data barely helped. The target is later low scoring, not misconceptions. The data cover one course in one semester, so transfer across semesters and institutions remains untested.

Keywords: early warning; programming education; target leakage; time-aware validation; transfer

INTRODUCTION

Learning to program is hard, and it is hard in much the same places all over the world. Reviews of introductory programming report the same conceptual difficulties for novices year after year (Luxton-Reilly et al., 2018; Robins et al., 2003), and failure rates in introductory courses have stayed close to one third (Bennedsen & Caspersen, 2007; Watson & Li, 2014). Abstract thinking and prior programming experience explain part of who struggles and who does not (Byrne & Lyons, 2001; Gomes & Mendes, 2007), but the struggle itself usually surfaces as syntax and logic errors. Errors are symptoms, and a good share of them come from misconceptions, that is, incorrect mental models of how a program executes (Qian & Lehman, 2017; Sorva, 2013). Because such models are stable, a student can repeat the same kind of mistake without seeing why it is wrong. Feedback that arrives only after several failed attempts therefore comes too late: the student has already spent time on an approach that cannot work, often at some cost to confidence and motivation.

Automated detection of misconceptions has advanced. The McMining benchmark introduced 67 misconception types together with large language model tools for recognising them (Al-Hossami & Bunescu, 2026), and Hoq et al. (2025) targeted logical errors in introductory programs. These systems read code after errors have appeared, so they describe a problem rather than anticipate it. A separate strand reads behavioral traces to anticipate difficulty: compilation behavior (Jadud, 2006), programming logs (Watson et al., 2013) and data from the first weeks of a course (Ahadi et al., 2015) have all been linked to later performance. Most of this work predicts course-level outcomes inside one institution, and the features it uses are often aggregates that would not be available at the moment an intervention is needed (Kaufman et al., 2012).

This paper sits between the two strands, and it is deliberately modest about what it predicts. We predict *struggle*: whether a student's later submissions on an assignment mostly earn no score. We do not label or predict misconceptions. Struggle is a behavioral, outcome-level construct; a misconception is a claim about a mental model that would require code to be coded against a taxonomy. Treating the first as a proxy for the second would be unsupported, so we do not do it.

A further reason for care is that event-level educational logs usually record several rows per student action. A run produces an execution event, a compile event and one or more compiler-message events, so a careless definition of "submission" changes both the counts and the labels. We document one such failure mode in the Dataset section. Our central methodological point is that three things must be kept apart: (i) the information used to predict, which must exist before the moment of prediction; (ii) the quantity predicted, which must not be recomputable from (i); and (iii) the evaluation, which must respect time and be judged against the decision it supports. We organise the empirical work around three research questions:

- **RQ1.** How early, measured in submissions k on an assignment, can later struggle be predicted better than base-rate and trivial-policy baselines under time-aware validation, and do code-level features (compile-error categories, error persistence, time to first success) add signal beyond volume and timing?

- **RQ2.** How well does a model trained on one cohort transfer to another, across sections of the same course, across assignment sets, and across both at once?

- **RQ3.** After transfer, how much labeled target-cohort data is needed to recover performance?

Contributions. (1) A demonstration of how naive event-level filtering produces an artefactual label, with a reproducible definition of the submission unit. (2) A clear separation of whole-history aggregation from truncated-history early prediction, with a demonstration that the whole-history label is leaky by construction. (3) A forward-looking label that cannot be recomputed from the predictors, validated with rolling-origin splits and reported as a function of k. (4) A decision-oriented evaluation (PR-AUC, cost-sensitive thresholds, calibration, student-level confidence intervals) showing when a model does and does not beat trivial policies. (5) A three-way transfer analysis with a target-sample learning curve. (6) A released script and result files that reproduce every table from the public raw log.

Related work

Difficulties and Misconceptions in Introductory Programming

Reviews of novice programming document persistent conceptual difficulties and misconceptions (Qian & Lehman, 2017; Robins et al., 2003), and the educational data mining literature on programming is now substantial (Ihantola et al., 2015). Early misconception work relied on expert-designed tests, interviews and surveys (Qian & Lehman, 2017). These methods are thorough but hard to scale to large classes or to apply while students are still practising.

Automated Detection of Misconceptions and Errors

Automated approaches analyse student code directly. McMining offers a benchmark of 67 misconception types plus LLM-based detection tools; the approach works well for known misconceptions but depends on

labeled data (Al-Hossami & Bunescu, 2026). Hoq et al. (2025) automated the identification of logical errors in introductory programs. These systems are valuable for diagnosis, but because they inspect work already submitted they say little about who is about to struggle. They also show what a true misconception label would require: code coded against a taxonomy, which our score-based label does not provide.

Early Warning from Programming Behavior

Jadud (2006) showed that novice compilation behavior can be summarised to characterise difficulty; Watson et al. (2013) predicted course performance from logged programming behavior; and Ahadi et al. (2015) applied machine learning to early-course data to identify students in need of assistance. Platforms such as CodeWorkout produce the fine-grained submission data these analyses need (Edwards & Murali, 2017), and knowledge tracing predicts future correctness from interaction histories (Piech et al., 2015). Most of this work targets course-level outcomes or later exercises rather than the first few submissions to an assignment, and it rarely reports performance as a function of how much history is available.

Leakage, Evaluation and Transfer

Target leakage arises when the label can be reconstructed from the predictors, which makes results look far better than they will be in deployment (Kaufman et al., 2012). In early prediction it happens whenever a feature such as overall success rate is used to predict a label defined by that same quantity, and also, less visibly, when features and labels are computed over the same time window. Evaluation has two further requirements. Splits must respect time, since random cross-validation over whole histories lets the model see the future (Bergmeir & Benítez, 2012). And when classes are imbalanced, ROC-AUC and accuracy-style scores can hide weak performance, so precision-recall analysis (Saito & Rehmsmeier, 2015), explicit costs (Elkan, 2001) and calibration (Niculescu-Mizil & Caruana, 2005) matter for the decision the model supports. Transfer learning addresses the situation where labeled target data are scarce and knowledge must be reused from a source domain (Pan & Yang, 2010), which is the position most institutions are in. Systematic evidence for programming-education data is thin.

Research Gap

Existing work either detects errors after they occur or predicts course-level outcomes from behavioral data. It rarely (a) restricts features to information available early and labels from later work, (b) reports the cost of that earliness, (c) compares against trivial policies under explicit costs, and (d) tests whether predictions survive a change in cohort or assignment set. This study addresses those four points on one dataset and states plainly where it stops.

Dataset

CodeWorkout Dataset

We used the CodeWorkout Spring 2019 release (PSLC DataShop dataset 3458) from Virginia Tech: an introductory Java course ("CS 1") that ran from 20 February to 6 May 2019. The release is an event log of 201,570 events from 413 students on 50 programming problems grouped into five assignments of ten problems each. Each assignment was open for about one week and the five windows do not overlap, which gives a natural time axis. The log has three event types: *Run.Program* (69,627 events, one per submission, carrying a score between 0 and 1), *Compile* (69,627 events, one child of each submission, carrying Success or Error) and *Compile.Error* (62,316 events, compiler messages attached to failed compiles). Throughout the paper, a **submission** is one Run.Program event, and "events" refers to the raw log. The problems cover loops, strings, arrays, conditionals and other fundamentals.

Submission Construction and a Filtering Pitfall

We define a submission as one Run.Program event, whatever its score. Its compile result is taken from the child Compile event, and its first compiler message from the child Compile.Error events. The pitfall to avoid is filtering the log by event outcome. Keeping every Compile event with a successful result together with

every Run.Program event with a score above zero, and dropping the rest, yields 81,357 rows from 407 students, a mixture of event types that omits all failed attempts. Every successful run is paired with a successful compile event, so the "success rate" over such rows (share of rows with a positive score) cannot exceed 0.5: its maximum is 0.500 and 90% of students lie between 0.468 and 0.500 (Figure 1). Because zero-score runs have been removed, this quantity no longer measures failure at all, and a label of the form "success rate below 0.5" flags 59.5% of students for reasons that reflect the ratio of event types, not struggle.

With submissions defined as Run.Program events, zero scores included, the share of students who succeed on fewer than half of their submissions is 21.7% (Table 3). All analyses in this paper use this definition.

Table 1. Data flow from the raw log to the analysis set.

Stage	Students	Section 1	Section 2	Records
Raw log	413 students	285 enrolments	130 enrolments	201,570 events (69,627 Run.Program + 69,627 Compile + 62,316 Compile.Error)
Submissions (Run.Program events)	413 (2 enrolled in both sections)	n/a	n/a	69,627 submissions
Analysis set (the 2 shared students excluded)	411	283	128	69,189 submissions (46,015 + 23,174)

The course ran two sections in parallel with the same five assignments, the same 50 problems and the same deadlines. Two students appear in both sections (285 + 130 - 2 = 413 unique students). We excluded both from every analysis, because a student present in training and test populations would inflate any transfer score. The analysis set therefore has 411 students and 69,189 submissions. Six of the 411 students have fewer than ten submissions; they are kept, and their whole-history labels are correspondingly noisy.

Table 2. Cohort statistics of the analysis set. The 50 problems and five assignments are shared by both sections.

Statistic	Section 1	Section 2	Total
Students	283	128	411
Submissions	46,015	23,174	69,189
Mean submissions per student	162.6	181	168.3
Mean attempts per problem (mean of per-student means)	4.15	4.76	4.34
Compile success rate	0.599	0.595	0.597
Share of submissions with score > 0	0.573	0.569	0.572
Whole-history positive students (success rate < 0.5)	61 (21.6%)	28 (21.9%)	89 (21.7%)

Table 3. Effect of naive event filtering on the whole-history label.

Quantity	Naive event filtering	Run.Program submissions
Rows treated as submissions	81,357 (compile successes + positive-score runs)	69,189 Run.Program events (analysis set)
Maximum per-student success rate	0.5	1
Spread of per-student success rate (5th to 95th percentile)	0.468 to 0.500	0.357 to 0.811 (full range 0 to 1)
Students labeled positive	59.50%	21.7% (89 of 411)

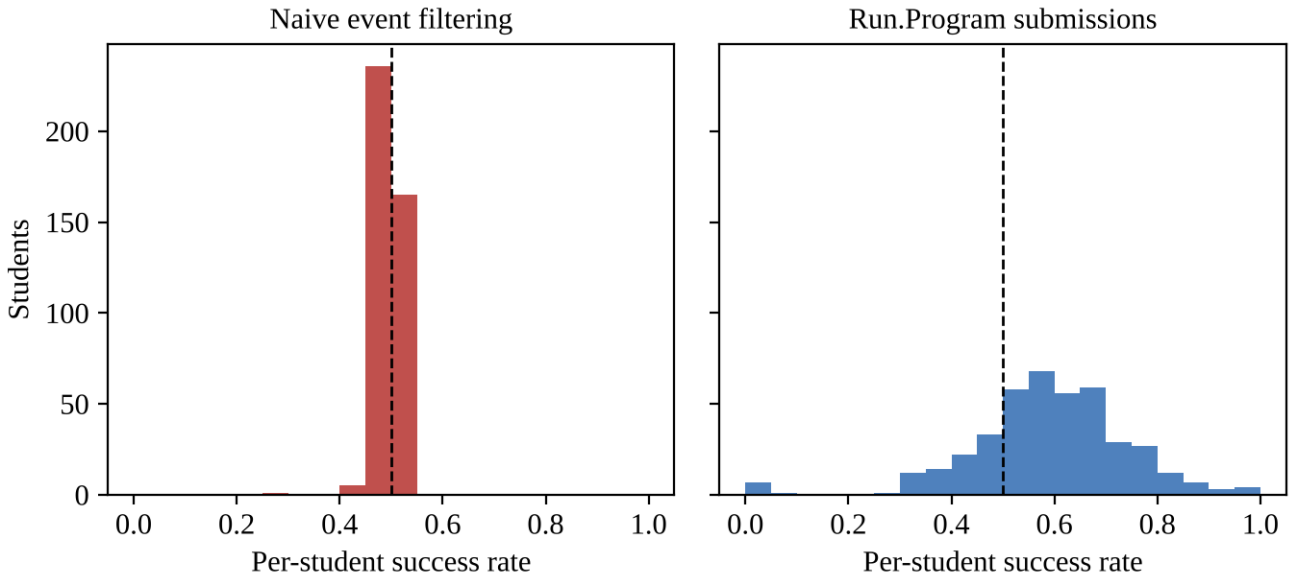

Figure 1. Per-student success rate under naive event filtering (left) and with Run.Program submissions (right). The dashed line is the 0.5 label cut-off.

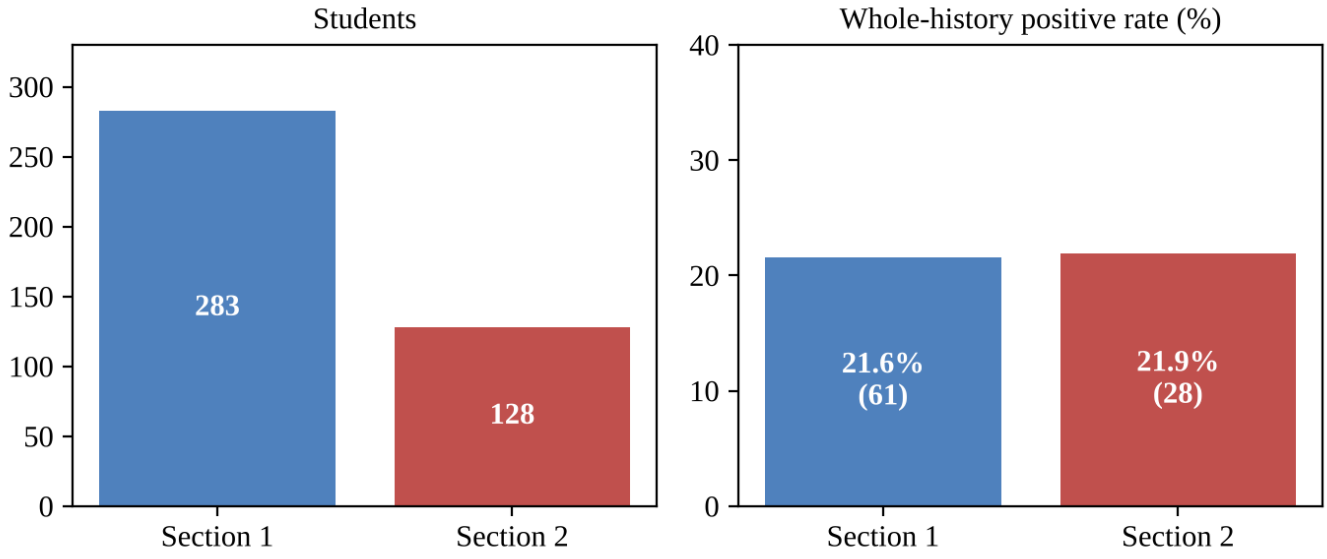

Figure 2. The two sections: size and whole-history positive rate. The sections do not differ in outcome rate.

METHODS

Problem Formulation

We use two formulations and keep them apart. The **whole-history diagnostic** (Experiment 1) aggregates a student's entire submission history into one row and labels the student from that same history. It is retained because it exposes leakage, but it is *not* early prediction: nothing in it is "before" anything. The **truncated-history early prediction** (Experiments 2 to 5) uses one unit per student and assignment. For a chosen k, features are computed from the student's first k submissions on that assignment (and from earlier assignments); the label is computed only from the submissions that follow. A unit is kept only if at least $m = 5$ later submissions exist, so that the label is defined. We report k taking the values 3, 5, 10, 20 and 40, with $k = 10$ as the headline setting for transfer. Units therefore differ across k: larger k keeps only students who submitted more (1,661 units from 405 students at $k = 3$, 562 units from 248 students at $k = 40$).

Labels

Whole-history label (Experiment 1 only). A student is positive when fewer than half of all their submissions earn a score above zero. There are 89 positives among 411 students (21.7%): 61 of 283 in section 1 and 28 of 128 in section 2. Because success rate is computable from the raw data, any predictor containing it recovers the label.

Forward label (Experiments 2 to 5). A unit is positive when fewer than half of the student's submissions *after the first k* on that assignment earn a score above zero. The label uses only work that no feature has seen, so it cannot be recomputed from the predictors. It is still derived from submission scores, because this dataset has no exam marks, instructor interventions or misconception codes. We therefore interpret it as *later struggle on the same assignment*. The 0.5 cut-off is a convention, and we report sensitivity at 0.3 and 0.7. The positive rate varies strongly by assignment (27%, 20%, 33%, 15% and 8% of units for assignments 1 to 5 at $k = 10$), which matters for transfer.

Features

Whole-history features (Experiment 1). Nine features were computed per student (Table 4). The strict set contains no outcome-derived feature. The honest set adds the mean score, which is closely related to the quantity the label is derived from. The leaky set adds success rate and compile success rate and contains the label itself. Time features use the gap between consecutive submissions to the same problem.

Table 4. Feature sets for the whole-history diagnostic.

Set	Features	Count
Strict	n_submissions, n_problems, avg_time, time_std, avg_attempts, max_attempts	6
Honest	strict + avg_score	7
Leaky	honest + success_rate, compile_success_rate	9

Early-prediction features (Experiments 2 to 5). All are computed from the first k submissions of the assignment, or from earlier assignments (Table 5). The total number of submissions is fixed at k and is dropped. Prefix scores and compile results are legitimate predictors here, because they describe work that precedes the label window; this distinguishes them from the leaky features of Experiment 1. Compiler-message categories come from the first message of each failed compile (for example "cannot find symbol", "missing return statement", "';' expected"); the six most frequent are used.

Table 5. Feature sets for early prediction.

Set	Features
Strict	n_problems touched; mean and SD of log inter-submission gap (capped at 30 min); mean and max attempts per problem; log elapsed time in the prefix
History	strict + success rate and compile-success rate in the prefix + success rate on earlier assignments (and an indicator that earlier work exists)
Code-level (added to strict or history)	share of prefix submissions in each of the six most frequent compiler-message categories; error persistence (longest run of consecutive compile failures); repeated-error rate (consecutive failures on the same problem with the same message category); submissions before first successful compile; submissions before first non-zero score; log time to first non-zero score

Models, Tuning and Calibration

The model is a class-balanced Random Forest (Breiman, 2001) with a fixed seed (20261019). In Experiment 1 it has 500 trees; Gradient Boosting and Logistic Regression (standardised inputs) serve as comparators. In Experiments 2 to 5 hyper-parameters are tuned inside each training set, over *max_depth* taking the values

unlimited and 6 and *min_samples_leaf* taking the values 1 and 5 with 200 trees, by PR-AUC under student-grouped three-fold cross-validation, so no test unit informs the choice. The grid is deliberately small; a larger grid was not explored. Probabilities are calibrated with isotonic regression fitted with student-grouped folds, and we report Brier score and expected calibration error (ECE, ten bins) before and after calibration. Decision thresholds are chosen on out-of-fold training predictions only, to minimise *c* times FN + FP for *c* taking the values 1, 2 and 5.

Validation

- **Experiment 1** (diagnostic): stratified five-fold cross-validation over the 411 students.
- **Experiment 2**: rolling-origin validation over assignments ordered in time. The model is trained on all assignments before assignment *j* and tested on *j*, for *j* = 3, 4, 5. A student may appear in training (earlier assignments) and in test (a later assignment), which mirrors deployment within a running course. Where students must not overlap (cross-validation references) folds are grouped by student.
- **Experiment 4** (transfer): train on one section and test on the other in both directions, under three shifts: new students on the same assignments; new assignments (train on assignments 1 to 3, test on 4 and 5) within a section; and both shifts at once.

Baselines and Evaluation

We report ROC-AUC, PR-AUC (read against prevalence, the PR-AUC of a model with no skill), and, for Experiment 1, F1, precision and recall at a 0.5 threshold. The trivial policies are *flag everyone* (recall 1, precision equal to prevalence) and *flag no one*. Costs are explicit: a missed struggling student costs *c* times a needless flag, *c* taking the values 1, 2 and 5, and the average cost per unit is (*c* times FN + FP)/*N*; the model with a cost-tuned threshold is compared with the cheaper of the two trivial policies. All intervals for AUC and PR-AUC are 95% percentile intervals from a student-level cluster bootstrap (500 resamples; students, not rows, are resampled), and differences between feature sets use paired resamples. Calibration scores are point estimates without intervals.

Experiments and results

Experiment 1: Whole-History Diagnostic

Table 6 shows five-fold cross-validation over whole histories. This experiment answers a limited question: how much of the label is recoverable from the features when features and label share a time window. It does *not* say how early struggle can be predicted.

Table 6. Experiment 1: stratified five-fold cross-validation over whole histories (411 students, 21.7% positive). AUC with 95% bootstrap CI.

Feature set	Model	F1	Precision	Recall	AUC-ROC [95% CI]	PR-AUC
Strict	Random Forest	0.484	0.455	0.517	0.778 [0.724, 0.829]	0.467
Strict	Gradient Boosting	0.403	0.477	0.348	0.751 [0.698, 0.803]	0.453
Strict	Logistic Regression	0.339	0.6	0.236	0.770 [0.712, 0.824]	0.501
Honest	Random Forest	0.734	0.664	0.82	0.934 [0.910, 0.954]	0.75
Honest	Gradient Boosting	0.711	0.703	0.719	0.931 [0.908, 0.953]	0.737
Honest	Logistic Regression	0.68	0.812	0.584	0.938 [0.913, 0.961]	0.798
Leaky	Random Forest	0.994	1	0.989	1.000 [1.000, 1.000]	1
Leaky	Gradient Boosting	0.994	1	0.989	0.992 [0.974, 1.000]	0.991
Leaky	Logistic Regression	0.948	0.976	0.921	0.998 [0.995, 1.000]	0.992

Baseline	flag everyone	0.356	0.217	1	0.5	0.217
Baseline	flag no one	0	0	0	0.5	0.217

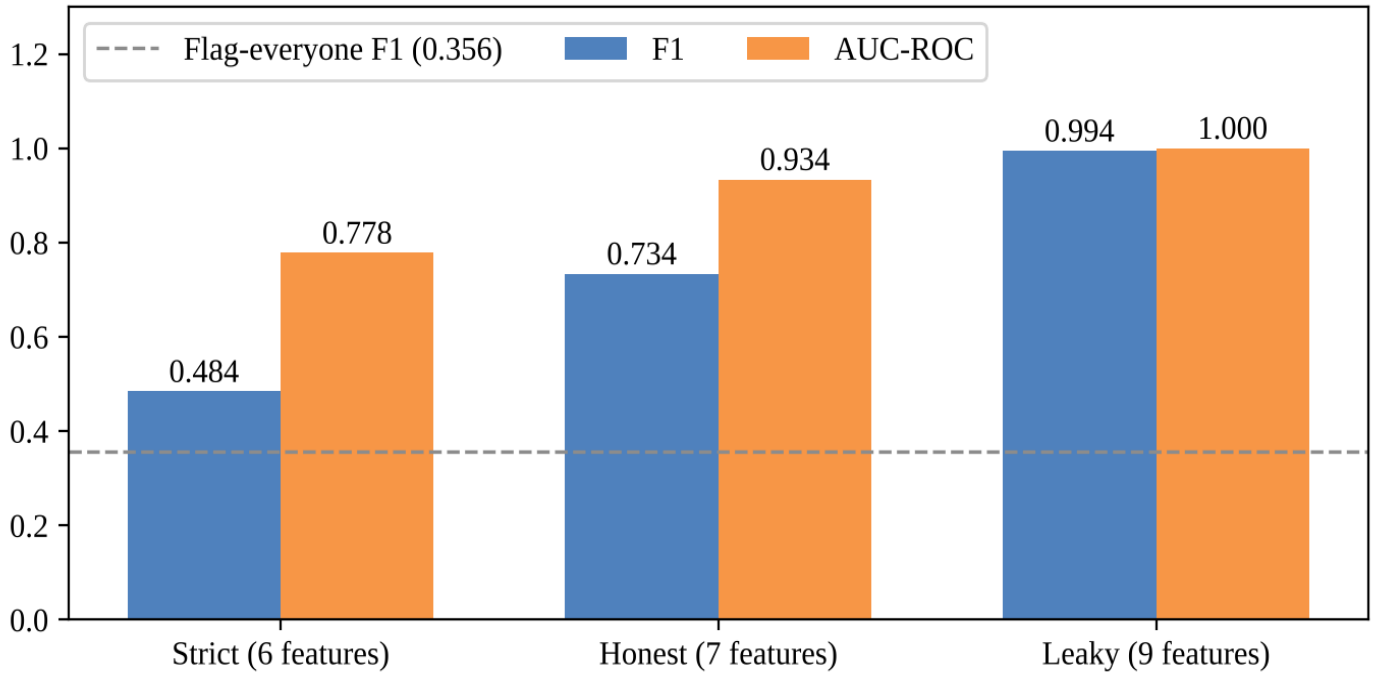

Figure 3. Experiment 1, Random Forest. Six strictly behavioral features give moderate performance; adding mean score raises it sharply, and adding the label-defining success rate saturates it. The dashed line is the F1 of flagging everyone.

Three readings follow. First, the leak is clear: restoring success rate gives an AUC of 0.99 to 1.00 and an F1 of 0.99 or more for the tree models, so near-perfect figures for this kind of label come from the feature set, not from a strong model. Second, the honest set, which adds only mean score, already reaches AUC 0.934, because mean score and the label measure almost the same thing; this is leakage by proxy and should not be read as prediction. Third, the strict behavioral model is modest: F1 = 0.484 against 0.356 for flagging everyone, AUC = 0.778 [0.724, 0.829].

Whether this is useful depends on costs. From the rounded table values, recall 0.517 and precision 0.455 on 89 positives and 322 negatives imply about 46 true positives, 43 missed students and 55 needless flags. Flagging everyone misses no one and flags 322 students unnecessarily; flagging no one misses 89. The model is cheaper than flagging no one when $43c + 55 < 89c$, that is $c > 1.2$, and cheaper than flagging everyone when $43c + 55 < 322$, that is $c < 6.2$. It beats both trivial policies only for cost ratios between about 1.2 and 6.2.

Experiment 2: Early Prediction as a Function of k

Experiment 2 implements the early-prediction design: features from the first k submissions, label from later submissions only, rolling-origin validation. Table 7 and Figure 4 give discrimination as a function of k. Performance is modest at every k. Strict behavioral features alone are barely above chance at $k = 3$ to 5 (AUC 0.56 and 0.62), and at $k = 20$ and 40 their intervals include 0.5. Adding earlier-work and prefix-score information raises AUC to 0.60–0.68, with intervals that exclude 0.5 at every k (lower limits 0.53 to 0.63). PR-AUC is 0.26–0.32 against a prevalence of 0.16–0.22, an absolute gain of roughly 0.06 to 0.12. Longer prefixes do not help: the curve is flat or declining with k, partly because larger k leaves fewer and different units.

Table 7. Experiment 2: ROC-AUC and PR-AUC [95% student-level bootstrap CI] by k, rolling-origin validation (test assignments 3 to 5). n is the number of pooled test units.

k	n	Prevalence	Strict AUC	History AUC	History+code AUC	History+code PR-AUC
3	954	0.179	0.564 [0.521, 0.611]	0.651 [0.608, 0.699]	0.652 [0.609, 0.698]	0.267 [0.224, 0.323]
5	946	0.164	0.619 [0.567, 0.665]	0.675 [0.626, 0.714]	0.681 [0.631, 0.724]	0.262 [0.219, 0.324]
10	744	0.198	0.577 [0.527, 0.628]	0.638 [0.590, 0.693]	0.614 [0.561, 0.668]	0.317 [0.255, 0.398]
20	549	0.217	0.536 [0.480, 0.600]	0.621 [0.568, 0.680]	0.626 [0.570, 0.684]	0.287 [0.235, 0.362]
40	304	0.207	0.481 [0.401, 0.556]	0.595 [0.519, 0.678]	0.597 [0.528, 0.669]	0.270 [0.212, 0.369]

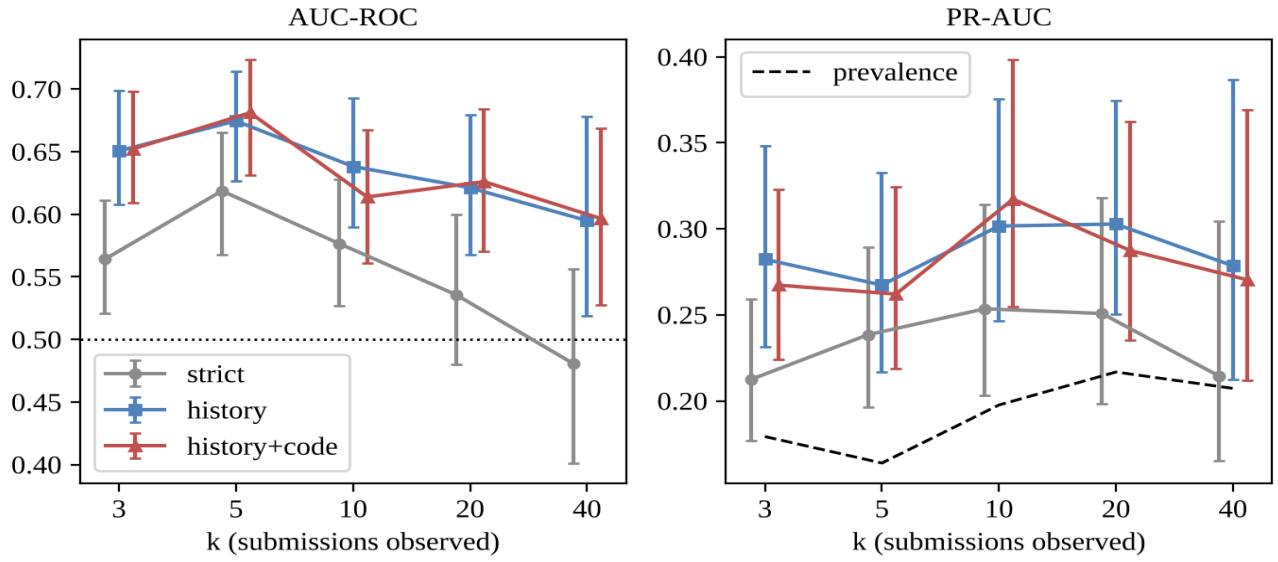

Figure 4. Experiment 2: discrimination by number of observed submissions. Left: AUC-ROC (dotted line = chance). Right: PR-AUC (dashed = prevalence). Bars are 95% student-level bootstrap intervals.

Table 8 compares the cost-tuned model with the cheaper of the two trivial policies, with student-level bootstrap intervals for the difference in average cost (negative favours the model). With equal costs ($c = 1$) the best policy is flag no one at every k, because struggling units are a minority. At $c = 2$ the model is never distinguishable from the best trivial policy. At $c = 5$ the model is significantly cheaper at $k = 5$ (-0.234 [-0.300, -0.124]) and $k = 20$ (-0.119 [-0.183, -0.054]), borderline at $k = 3$ (the interval reaches +0.006), and not distinguishable at $k = 10$ and 40. The saving comes from near-exhaustive flagging with a modest reduction in false alarms: at $k = 5$ and 20 the tuned model has recall 0.88 and 0.93 at precision 0.24 and 0.26.

Table 8. Experiment 2: difference in average cost per unit between the cost-tuned history+code model and the cheaper trivial policy (flag everyone or flag no one), with 95% student-level bootstrap CI; negative favours the model. The last column gives recall / precision of the model at the threshold tuned for c = 5.

k	Prevalence	Cost difference, c = 2 [95% CI]	Cost difference, c = 5 [95% CI]	Recall / precision, c = 5
3	0.179	-0.003 [-0.039, +0.030]	-0.097 [-0.179, +0.006]	0.57 / 0.24
5	0.164	+0.004 [-0.019, +0.025]	-0.234 [-0.300, -0.124]	0.88 / 0.24

10	0.198	+0.020 [-0.021, +0.055]	+0.006 [-0.105, +0.125]	0.52 / 0.24
20	0.217	+0.025 [-0.028, +0.072]	-0.119 [-0.183, -0.054]	0.93 / 0.26
40	0.207	+0.009 [-0.032, +0.048]	+0.007 [-0.145, +0.171]	0.63 / 0.24

Table 9 gives the equal-recall and equal-precision comparisons with flagging everyone, whose precision equals prevalence at recall 1. At 90% and 95% recall the model's precision is within about 0.05 of prevalence, so it cannot be used to catch nearly every struggling student efficiently. At a precision of 0.30, roughly 1.4 to 1.8 times the base rate, which flag-everyone cannot reach, the model recovers 33% to 43% of struggling units at k = 3 to 20 (19% at k = 40); a precision of 0.40 is attainable only at low recall (at most 0.18). Calibration is the clearest gain from post-processing: at k = 10 isotonic calibration lowers the Brier score from 0.199 to 0.158 and ECE from 0.201 to 0.053, so calibrated scores can be read as rates even when ranking is weak.

Table 9. Experiment 2: equal-recall and equal-precision comparison, history+code model. Flag everyone has precision equal to prevalence at recall 1.

k	Prevalence	Precision at recall of at least 0.90	Precision at recall of at least 0.95	Recall at precision of at least 0.30	Recall at precision of at least 0.40
3	0.179	0.23	0.192	0.327	0
5	0.164	0.213	0.194	0.381	0.026
10	0.198	0.21	0.199	0.367	0.177
20	0.217	0.246	0.234	0.429	0
40	0.207	0.238	0.224	0.19	0.063

Experiment 3: Do Code-Level Features Add Signal?

Table 10 gives paired differences in AUC from the strict set. Code-level features (compiler-message categories, error persistence, time to first success) help when little else is known: adding them to the strict set raises AUC by 0.07 at k = 3 and 0.05 at k = 5, with intervals excluding zero. The gain disappears at k = 10 and later, where intervals include zero. Once history is available, code-level features add nothing detectable: history+code against history differs by +0.001, +0.006, -0.024, +0.005 and +0.002 AUC at k = 3, 5, 10, 20 and 40. The most important features for the history+code model at k = 10 (Random Forest importance, full-data fit) are prior success on earlier assignments (0.129), prefix compile success (0.111), prefix success (0.087), submissions before the first non-zero score (0.064) and error persistence (0.062).

Table 10. Experiment 3: paired difference in AUC relative to the strict set [95% CI], p-value (rolling-origin).

k	strict+code - strict	history – strict	history+code - strict
3	+0.074 [+0.033, +0.110], p = 0.000	+0.088 [+0.046, +0.131], p = 0.000	+0.090 [+0.049, +0.129], p = 0.000
5	+0.053 [+0.019, +0.092], p = 0.004	+0.055 [+0.015, +0.095], p = 0.012	+0.062 [+0.026, +0.099], p = 0.000
10	-0.014 [-0.062, +0.033], p = 0.572	+0.062 [+0.010, +0.114], p = 0.028	+0.037 [-0.017, +0.093], p = 0.180
20	+0.043 [-0.032, +0.114], p = 0.244	+0.086 [+0.013, +0.158], p = 0.028	+0.090 [+0.014, +0.167], p = 0.032
40	+0.056 [-0.022, +0.142], p = 0.180	+0.112 [+0.029, +0.195], p = 0.004	+0.113 [+0.036, +0.201], p = 0.004

Experiment 4: Cross-Section, Cross-Assignment and Combined Transfer

Table 11 restates the whole-history transfer between sections. The strict model trained on one section scores

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue XI November 2026

0.770 (section 1 to 2) and 0.818 (section 2 to 1), at or above its within-section cross-validated AUC (0.642 and 0.777). The within-section reference for section 2 has a wide interval because only 28 students are positive, so the 1 to 2 transfer figure is not distinguishable from it. The sections have the same positive rate, so there is no prevalence shift to explain.

Table 11. Experiment 4a: whole-history transfer between sections (Random Forest). AUC with 95% bootstrap CI.

Direction	Feature set	F1	Precision	Recall	AUC-ROC [95% CI]
Within section 2 (CV)	Strict	0.379	0.367	0.393	0.642 [0.529, 0.756]
Within section 2 (CV)	Honest	0.623	0.576	0.679	0.850 [0.769, 0.915]
Within section 2 (CV)	Leaky	0.982	0.966	1	0.999 [0.996, 1.000]
Section 1 to 2	Strict	0.514	0.429	0.643	0.770 [0.664, 0.861]
Section 1 to 2	Honest	0.667	0.625	0.714	0.908 [0.853, 0.954]
Section 1 to 2	Leaky	0.982	1	0.964	1.000 [1.000, 1.000]
Within section 1 (CV)	Strict	0.47	0.437	0.508	0.777 [0.720, 0.834]
Within section 1 (CV)	Honest	0.746	0.685	0.82	0.936 [0.907, 0.962]
Within section 1 (CV)	Leaky	1	1	1	1.000 [1.000, 1.000]
Section 2 to 1	Strict	0.496	0.538	0.459	0.818 [0.761, 0.869]
Section 2 to 1	Honest	0.788	0.732	0.852	0.948 [0.921, 0.971]
Section 2 to 1	Leaky	0.984	0.968	1	1.000 [0.999, 1.000]

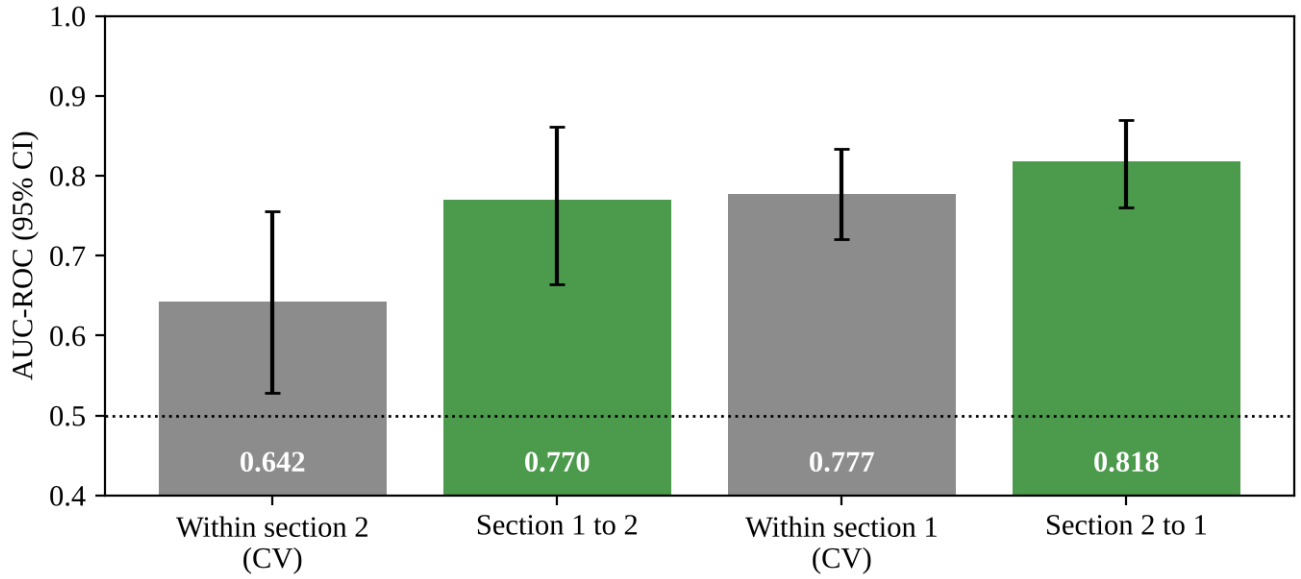

Figure 5. Experiment 4a, strict set. Grey bars: cross-validation within a section. Green bars: the model trained on the other section. Whiskers are 95% bootstrap intervals; the dotted line is chance.

Table 12 repeats the comparison for early prediction at $k = 10$ and adds the shifts that the original design could not reveal. Between concurrent sections on the same assignments, the history+code model reaches AUC 0.70 in both directions, close to its within-section reference (0.67 and 0.74, with overlapping intervals); transfer between sections holds up. Moving to later assignments is the hard case. Within a section, training on assignments 1 to 3 and testing on 4 and 5 gives AUC 0.58 (section 2) and 0.70 (section 1) for history+code, with intervals from about 0.40 to 0.80, and the strict model is at chance (0.53 and 0.56). The combined shift gives 0.61 and 0.63. The later assignments are also easier (positive rate 10–13% against 20–22%), so a model

trained early meets a different base rate. The two sections ran concurrently, so these results speak to new students and new assignments within one semester, not to new semesters.

Table 12. Experiment 4b: early-prediction transfer at k = 10. History+code AUC and PR-AUC [95% CI]; strict AUC for reference.

Scenario	n	Prevalence	History+code AUC	History+code PR-AUC	Strict AUC
Within section 2 (grouped CV reference)	443	0.199	0.669 [0.611, 0.727]	0.374 [0.300, 0.488]	0.616 [0.555, 0.676]
Section 1 to 2, same assignments	443	0.199	0.701 [0.645, 0.767]	0.413 [0.333, 0.531]	0.616 [0.555, 0.689]
Within section 1 (grouped CV reference)	931	0.224	0.736 [0.698, 0.772]	0.452 [0.389, 0.529]	0.640 [0.602, 0.683]
Section 2 to 1, same assignments	931	0.224	0.702 [0.660, 0.740]	0.471 [0.407, 0.543]	0.597 [0.554, 0.641]
Assignment shift within section 2 (1–3 to 4–5)	154	0.104	0.578 [0.400, 0.728]	0.214 [0.084, 0.386]	0.531 [0.355, 0.730]
Assignment shift within section 1 (1–3 to 4–5)	318	0.129	0.704 [0.617, 0.798]	0.351 [0.223, 0.496]	0.561 [0.464, 0.657]
Section 1 to 2, assignments 1–3 to 4–5	154	0.104	0.614 [0.437, 0.782]	0.246 [0.096, 0.458]	0.495 [0.333, 0.654]
Section 2 to 1, assignments 1–3 to 4–5	318	0.129	0.630 [0.536, 0.739]	0.295 [0.192, 0.452]	0.497 [0.403, 0.600]

Experiment 5 (RQ3): Target-Sample Learning Curve

We train on the full source section plus *n* randomly chosen target students (*n* taking the values 0, 10, 25, 50, 75 and 100, 20 draws per *n*) and test on the remaining target students with the history+code set at $k = 10$. Table 13 and Figure 6 show that target data adds little. Source-only models already give AUC 0.714 (1 to 2) and 0.704 (2 to 1), and adding up to 100 target students moves the mean to 0.735 and 0.720 with wide draw-to-draw intervals. A target-only model with 50 to 100 students approaches but does not exceed that level (0.66–0.68 for section 2, 0.70–0.72 for section 1). In this setting the source section is a sufficient starting point; local labeled data helps little within the range tested.

Table 13. Experiment 5: mean AUC [95% interval across 20 draws] against target sample size n. The n = 0 model is a single fit.

Direction	Model	n = 0	n = 10	n = 25	n = 50	n = 75	n = 100
1 to 2	source+target	0.714	0.713 [0.69, 0.73]	0.698 [0.67, 0.72]	0.708 [0.66, 0.76]	0.713 [0.64, 0.79]	0.735 [0.62, 0.89]
1 to 2	target-only	n/a	n/a	0.644 [0.59, 0.67]	0.660 [0.62, 0.74]	0.680 [0.62, 0.78]	0.677 [0.57, 0.80]
2 to 1	source+target	0.704	0.703 [0.69, 0.71]	0.706 [0.69, 0.72]	0.708 [0.69, 0.73]	0.715 [0.70, 0.73]	0.720 [0.68, 0.75]
2 to 1	target-only	n/a	n/a	0.683 [0.58, 0.72]	0.702 [0.68, 0.72]	0.703 [0.67, 0.72]	0.718 [0.67, 0.75]

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue XI November 2026

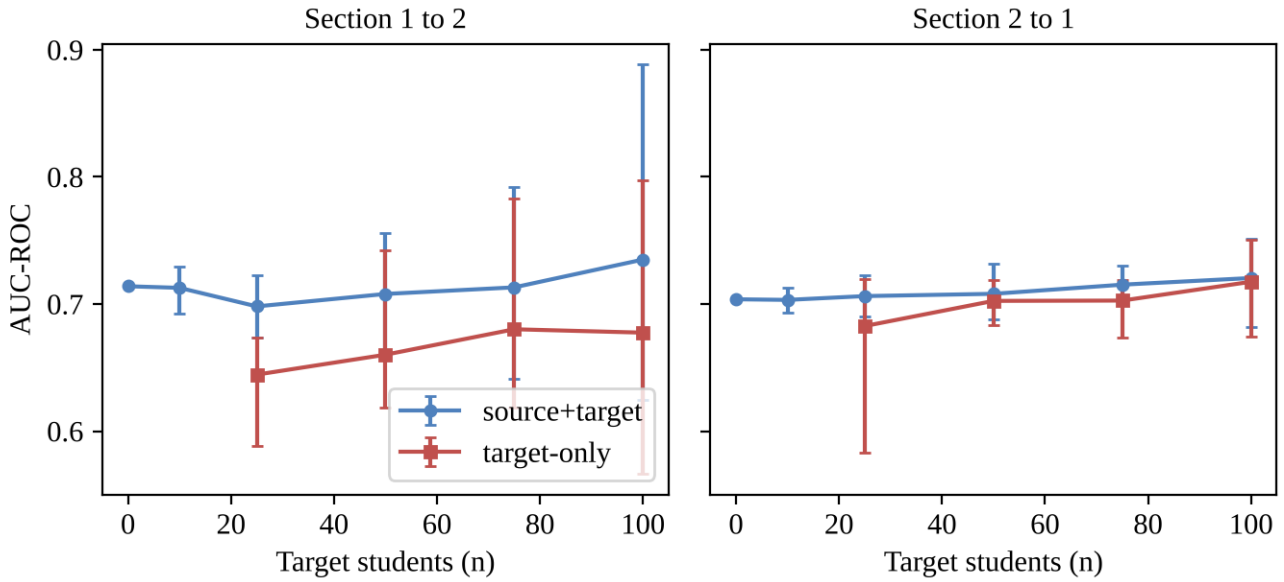

Figure 6. Experiment 5: AUC against the number of target students in training. Bars are 95% intervals across random draws.

Feature Importance and Label-Threshold Sensitivity

Table 14 shows importance for the Random Forest fitted on the strict whole-history set. No outcome feature is present, but the ranking comes from the whole-history design and does not describe what is available early. For the forward label, the 0.5 cut-off matters: at 0.3 only about 2% of units are positive, too few to evaluate; at 0.7 the prevalence is 0.644 and PR-AUC is 0.728 for the strict set and 0.784 for history+code at $k = 10$, a gain over prevalence of 0.08 to 0.14, similar in size to the main result.

Table 14. Feature importance (Random Forest, strict whole-history set).

Feature	Importance
avg_attempts	0.279
n_submissions	0.159
n_problems	0.152
max_attempts	0.143
time_std	0.136
avg_time	0.131

DISCUSSION

What the Evidence Supports

Four findings are secure. First, the definition of a submission matters: filtering an event log by outcome can cap a success rate at 0.5 and manufacture a label, so any study built on event logs with several rows per student action should check that a “submission” is one action. Second, the whole-history label is circular: any feature set that includes success rate reaches an AUC of 0.99 or more, and a mean-score feature recovers most of it by proxy. Third, early prediction of later struggle is possible but weak: AUC of 0.60–0.68 and PR-AUC of 0.26–0.32 against a base rate near 0.2, with code-level features helping only at the very start. Fourth, whether such a model is useful depends on cost. With a missed student five times as costly as a needless flag, the cost-tuned model is significantly cheaper than the best trivial policy at $k = 5$ and 20; with equal costs or a ratio of two, flagging no one is as good.

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue XI November 2026

These are modest results, and they should be read as such. A model with AUC near 0.65 is a weak triage aid, not a detector. The calibration result, in which post-hoc isotonic calibration cuts expected calibration error from about 0.20 to 0.05, matters more for practice than the ranking gain.

What Transferred and What Did Not

Transfer between concurrent sections is stable: a model trained on one section scores about as well on the other as it does within its own, and adding target-section students barely changes it. This is expected when the sections sat the same problems, under the same deadlines, in the same semester, on the same platform; it is not evidence that a model trained at one institution would work at another. The assignment-shift analysis relaxes one of those constraints and shows the real weakness: a model trained on early assignments loses much of its skill on later ones, and the strict behavioral set falls to chance. Part of the reason is drift in the base rate across assignments, which a deployed system would have to track and recalibrate.

Implications for Practice

A deployed system should be judged against trivial policies at an explicit cost ratio, not against accuracy. Three practical consequences follow. The threshold should be set from the institution's own cost ratio on out-of-fold training predictions. Calibrated probabilities allow instructors to read a score as a rate. And since the labels describe struggle, not a diagnosed misconception, the output suits low-stakes nudges, such as an offer of a hint or a prompt to review the previous attempt, and does not suit consequential decisions about a student. Because the base rate shifts across assignments, thresholds and calibration should be refreshed each assignment.

Limitations

- Everything comes from one course, one semester, one institution and one platform. Transfer across semesters, institutions, countries or platforms is untested.
- The sections ran concurrently, so section transfer is between populations, not across time. The assignment-shift analysis is a partial substitute, because all assignments are in one term.
- The forward label is still derived from submission scores. It cannot be recomputed from the predictors, but it is not an external outcome such as exam performance, instructor intervention or a misconception-coded rubric. The script accepts exam scores for a secondary analysis, but this dataset does not provide them.
- No misconception labels are used, so no claim about misconceptions is made. Producing them would need submissions coded against a taxonomy (for instance the McMining types) and a model that predicts those codes.
- Positives are a minority (about 20%), and section 2 has only 28 positive students in the whole-history analysis. Many intervals are wide, and several "differences" are within noise.
- The 0.5 label cut-off and the minimum of five later submissions are conventions. The 0.3 cut-off leaves too few positives to evaluate; results at 0.7 resemble the main results.
- The tuning grid is small (four settings, 200 trees) because of compute limits; larger searches or other model families might do somewhat better. Calibration scores are point estimates without intervals, and the cost-tuned thresholds are chosen on a single training partition per fold.
- Costs are stylised ratios, not measured costs of interventions. A deployment study would need the real cost of a missed student and of an unnecessary flag.
- Domain-adaptation methods were not tested; Experiment 5 only measures how far simple pooling of source and target data goes.

- Compiler-message categories depend on the Java compiler's wording, which limits portability of the code-level features, and all compile errors in this log are reported as syntax errors by the platform's type field, so categories were derived from the message text.

CONCLUSION

This paper separates whole-history aggregation from true early prediction and evaluates the latter on a public programming log. A whole-history label is circular, mean score recovers it by proxy, and six strictly behavioral whole-history features reach only AUC 0.78. Naive event filtering can even manufacture the label, so we define a submission as one program-run event. Truncating each student's history and labeling from later work gives a weak but real early signal (AUC 0.60–0.68, PR-AUC 0.26–0.32 against a 0.16–0.22 base rate) that does not improve with longer prefixes. Code-level features help only when almost nothing else is known. A cost-tuned model is significantly cheaper than the best trivial policy only at a 5:1 cost ratio and only at some k.

Models transfer between concurrent sections as well as they work within a section, and local data adds little; they transfer poorly to later assignments, where the base rate changes. The conclusions stop at the boundary we could test: one course, one semester, one platform, and a score-derived label. The next steps are an external outcome for the label, misconception coding if misconceptions are the real target, and tests across semesters and institutions.

REFERENCES

1. Ahadi, A., Lister, R., Haapala, H., & Vihavainen, A. (2015). Exploring machine learning methods to automatically identify students in need of assistance. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*. ACM.
2. Al-Hossami, E., & Bunescu, R. (2026). *McMining: Automated discovery of misconceptions in student code*.
3. Bennedsen, J., & Caspersen, M. E. (2007). Failure rates in introductory programming. *ACM SIGCSE Bulletin, 39*(2).
4. Bergmeir, C., & Benítez, J. M. (2012). On the use of cross-validation for time series predictor evaluation. *Information Sciences, 191*.
5. Breiman, L. (2001). Random forests. *Machine Learning, 45*(1).
6. Byrne, P., & Lyons, G. (2001). The effect of student attributes on success in programming. In *Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education*. ACM.
7. Edwards, S. H., & Murali, K. P. (2017). CodeWorkout: Short programming exercises with built-in data collection. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*. ACM.
8. Elkan, C. (2001). The foundations of cost-sensitive learning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*.
9. Gomes, A., & Mendes, A. J. (2007). Learning to program: Difficulties and solutions. In *International Conference on Engineering Education*.
10. Hoq, M., Rao, A., Jaishankar, R., Piryani, K., Janapati, N., Vandenberg, J., Mott, B., Norouzi, N., Lester, J., & Akram, B. (2025). *Automated identification of logical errors in programs: Advancing scalable analysis of student misconceptions*.
11. Ihantola, P., Vihavainen, A., Ahadi, A., Butler, M., Börstler, J., Edwards, S. H., Isohanni, E., Korhonen, A., Petersen, A., Rivers, K., Rubio, M. Á., Sheard, J., Skupas, B., Spacco, J., Szabo, C., & Toll, D. (2015). Educational data mining and learning analytics in programming: Literature review and case studies. In *Proceedings of the 2015 ITiCSE on Working Group Reports*. ACM.
12. Jadud, M. C. (2006). Methods and tools for exploring novice compilation behaviour. In *Proceedings of the Second International Workshop on Computing Education Research*. ACM.
13. Kaufman, S., Rosset, S., Perlich, C., & Stitelman, O. (2012). Leakage in data mining: Formulation, detection, and avoidance. *ACM Transactions on Knowledge Discovery from Data, 6*(4).

14. Luxton-Reilly, A., Simon, Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A. N., Ott, L., Paterson, J., Scott, M. J., Sheard, J., & Szabo, C. (2018). Introductory programming: A systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*. ACM.
15. Niculescu-Mizil, A., & Caruana, R. (2005). Predicting good probabilities with supervised learning. In *Proceedings of the 22nd International Conference on Machine Learning*. ACM.
16. Pan, S. J., & Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering, 22*(10).
17. Piech, C., Bassen, J., Huang, J., Ganguli, S., Sahami, M., Guibas, L. J., & Sohl-Dickstein, J. (2015). Deep knowledge tracing. In *Advances in Neural Information Processing Systems 28*.
18. Qian, Y., & Lehman, J. (2017). Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education, 18*(1).
19. Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education, 13*(2).
20. Saito, T., & Rehmsmeier, M. (2015). The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PLoS ONE, 10*(3).
21. Sorva, J. (2013). Notional machines and introductory programming education. *ACM Transactions on Computing Education, 13*(2).
22. Watson, C., & Li, F. W. B. (2014). Failure rates in introductory programming revisited. In *Proceedings of the 2014 Conference on Innovation and Technology in Computer Science Education*. ACM.
23. Watson, C., Li, F. W. B., & Godwin, J. L. (2013). Predicting performance in an introductory programming course by logging and analyzing student programming behavior. In *Proceedings of the 13th IEEE International Conference on Advanced Learning Technologies*. IEEE.