Hybrid Approach Combining Markov Chains, HMM, and Bidirectional GRU to Detect Blockages in Programming

Grota Abdelkader[1], Mohammed Erritali[2], Patrick Etcheverry[3], Thierry Nodenot[4]
Data4Earth Laboratory, Faculty of Sciences and Techniques, Beni Mellal, Morocco[1,2]
Laboratory LIUPPA, University of Pau and the Adour Region, IUT Bayonne, France[3,4]

***Abstract*—A student stuck in a failing compile loop looks productive from across the room. The instructor sees keystrokes; the blockage is invisible until the student raises a hand or falls silent. Existing detection methods work at session level: they report how many compilations occurred, not what happened between them. Sub-minute resolution is needed. We built FusionAdaptative, a hybrid model that reads raw keystroke traces at 30-second windows, combining Markov Chains on a 13-state behavioral taxonomy, HMM latent state inference, and BiGRU with attention, combined at the level of representation. On 70 first-year CS students, 287,236 actions, 220 sequences, a counter-intuitive result holds: all eight evaluated configurations, from a non-neural Random Forest to the full hybrid, reach an equivalent Macro-F1 of about 90.7% (Friedman $\chi^2(7) = 3.53$, $p = 0.83$; Nemenyi post-hoc: no pairwise difference, critical difference $= 4.70$). This is not a weakness of the model but a property of the design: once cognitive states are defined by quantitative thresholds validated at $\kappa = 0.81$, classification is constrained enough that architecture no longer matters. The discriminative power is in the taxonomy, not the complexity of the model. FusionAdaptative's contribution is temporal. It detects blockages on average 2.8 minutes before they become visible, a lead that only sequential modeling (BiGRU attention and HMM Viterbi decoding) can produce and that aggregated classifiers such as Random Forest cannot. The lead varies by blockage type: 3.2 minutes for conceptual, 2.6 for syntactic, 2.4 for strategic, each anticipated earliest by a different component of the model.**

***Keywords*—Attention mechanism; bidirectional GRU; blockage detection; early detection; educational data mining**

I. Introduction

An instructor managing twenty students in a programming lab cannot watch all of them at once. That is the core problem. Blockage events happen silently: a student stuck in repetitive failed compilations, disorganized navigation, and growing inactivity, inside an individual workstation, while the instructor helps someone else across the room.

AI coding assistants have made this harder. A student who pastes a compiler error into ChatGPT and gets working code back has not resolved the blockage; the conceptual gap is still there [1], [2], [3]. The code runs. The student does not understand why. Detection methods that look at the output miss this entirely [4]. You have to observe what the student does, not what they produce.

Development environments do record what students do. Every keystroke, every compilation, every pause has a timestamp. These traces have structure: rhythm, acceleration, deceleration, repetition. That structure reflects cognitive states the final code does not [5]. The question is whether a model can read this structure early enough to be useful before the blockage is visible, and specifically enough to guide an intervention.

A. Research Gap

Three families of methods have been tried. Session-level statistics (compilation count, time-on-task) correlate with difficulty but average over the full exercise duration and cannot distinguish a student pausing to think from one who has stopped progressing [5]. Markov Chains capture transition patterns between behavioral states but depend only on the current state; everything earlier in the session is discarded [6], [7]. HMMs infer latent cognitive states from observable sequences but assume conditional independence between observations, which fails for autocorrelated keystroke data [9]. Deep learning models (RNN, LSTM) handle long-range dependencies but produce no interpretable state representation, a prediction without a reason [13], [14].

Each family has one of three things: interpretable states, long-range memory, or temporal resolution. None has all three. FusionAdaptative combines them.

B. Contributions

This study presents four contributions:

- (C1) A three-dimensional behavioral taxonomy of 13 action types with quantitative thresholds, validated at $\kappa = 0.81$ inter-annotator agreement. It distinguishes three blockage types: syntactic, semantic, and conceptual. Each has a distinct observable signature and a distinct appropriate intervention.
- (C2) FusionAdaptative, a hybrid model combining Markov Chain transition modeling, HMM latent state inference, and Bidirectional GRU sequence learning. Component weights are adjusted at each time step by instantaneous prediction confidence, so the model down-weights whichever component is currently uncertain. The value of this mechanism is not higher classification accuracy but the ability to identify, per blockage type, which component anticipates the difficulty earliest.

- (C3) Early detection on average 2.8 minutes before blockage becomes visible, a temporal lead that aggregated classifiers cannot produce because they operate on session-level features rather than the sequence itself. The lead breaks down by blockage type: 3.2 minutes for conceptual (RNN attention), 2.6 for syntactic (Markov transitions), 2.4 for strategic (HMM latent state).
- (C4) A counter-intuitive comparison result: all eight configurations, from a non-neural Random Forest to the full hybrid, reach an equivalent Macro-F1 of about 90.7% (Friedman $\chi^2(7) = 3.53$, $p = 0.83$). This validates the behavioral taxonomy rather than any single architecture: with thresholds fixed at $\kappa = 0.81$, the discriminative power lies in the feature definition, not the model.

The remainder of the study is organized as follows. Section II reviews related work. Section III formalizes the problem. Section IV-A describes the architecture. Section IV-B gives the experimental protocol. Section V reports results. Section VI covers explainability, Section VII scalability. Section VIII discusses and Section IX concludes.

II. Related Work

A. AI Coding Assistants and the New Educational Challenge

Prather et al. [1] measured a 39% drop in compilation errors after Copilot was introduced; students submitted working code more often. Finnie-Ansley et al. [2] found that ChatGPT solved introductory programming problems at above-median student level. Both results share the same implication: a student can now get a passing solution without understanding it. The conceptual gap is masked by correct output [3], [4].

Detection methods that watch output therefore miss the gap entirely. A student who paste-replaced their broken code with a chatbot solution and a student who fixed it by reasoning look identical at submission. What differs is the sequence of edits, deletions, and pauses in between, and those remain recorded in the activity log [3].

B. Evolution of Student Behavior Modeling

Blikstein [5] showed that compilation counts and time-on-task correlate with final grade, useful at population level, useless for detecting a single student's difficulty in real time. Two students can compile twenty times in an hour: one testing incremental changes, one cycling through the same broken line. The count is the same; the situation is not.

Markov Chains were the first attempt to add structure. They model transitions between discrete behavioral states and show that productive students tend toward structured sequences (Edit $\rightarrow$ Compile $\rightarrow$ Test) while struggling students loop on Compile $\rightarrow$ Error $\rightarrow$ Compile [6]. The limit is direct: the model depends only on the current state. Everything earlier in the session is gone [7].

HMMs addressed this with latent states. A student making many small edits and recompiling constantly might be in a confusion state, not a productive one, even if the surface behavior looks busy [8]. HMMs capture that distinction. What they cannot capture is long-range dependency: if a student made a strategic mistake at minute 2, the HMM at minute 8 does not know [9].

C. Deep Learning for Sequence Modeling

Piech et al. [12] used LSTMs to track knowledge state across exercises; Nguyen et al. [13] applied GRUs to action traces and beat HMM baselines by 8–11 points. The gain is real. The cost is that neither model says which actions drove the prediction; a binary score delivered to an instructor who needs to decide where to walk is not enough [14].

Transformers [15] added attention weights as a proxy for explanation. Jain and Wallace [20] tested whether those weights correlate with gradient-based attribution on real data: they do not, consistently. On the same input, attention can assign high weight to a step that has near-zero gradient influence on the output. In educational data mining, adoption has been slow precisely because of this [16], [17].

D. Hybrid and Multidimensional Approaches

Rahman and Liu [18] used an HMM to infer high-level cognitive states, then fed those states into an RNN for sequential prediction, reaching 87.3% accuracy on a comparable detection task. Zhang et al. [19] added code complexity metrics and emotional indicators alongside behavioral traces and reported gains over behavior-only baselines.

Both studies used fixed weights between components. The contribution of each model was set once and stayed constant. FusionAdaptative differs: each component's weight changes at every time step, based on its confidence at that moment. When the Markov component is uncertain (high transition entropy), its weight drops. When BiGRU attention is diffuse, its weight drops too. No prior hybrid model in this domain uses this mechanism.

E. Explainable AI in Educational Data Mining

SHAP [21] gives feature attributions for any model by treating prediction as a cooperative game between input features. Integrated Gradients [22] attributes neural network predictions to individual input dimensions. Both have been applied in educational settings and increase instructor trust compared to raw prediction scores.

The catch is that both assume feature independence. Keystroke sequences are not independent: what a student does at second 30 correlates with what they did at second 15. FusionAdaptative handles interpretability at two levels: HMM latent states give a built-in human-readable label (Progress, Hesitation, Blockage, Confusion), and SHAP applied to aggregated 30-second feature vectors gives post-hoc attribution. Section VI checks whether these two sources agree.

F. Positioning: Temporal Resolution of Existing Approaches

The studies reviewed above differ on one axis that is rarely made explicit: the temporal resolution at which student behaviour is observed. Table I organises them along it. The distinction matters because it bounds what a method can detect. An approach that aggregates over a whole session can report

TABLE I. TEMPORAL RESOLUTION OF REPRESENTATIVE APPROACHES TO PROGRAMMING-DIFFICULTY DETECTION

Study	Unit of analysis	Resolution	Interpretable output
Blikstein [5]	Session	Session-level	Aggregate indicators
Jadud [26]	Compilation event	Event-level	Error quotient
Carter and Hundhausen [6]	Action transitions	Sub-session	State transitions
Piech et al. [12]	Exercise response	Exercise-level	None (latent vector)
Nguyen et al. [13]	Action sequence	Sub-session	None
Rahman and Liu [18]	Action sequence	Sub-session	Latent states
Zhang et al. [19]	Session + code	Session-level	Feature importance
This work	**30-second window**	**Sub-minute**	**Latent state, attention, transitions**

how many compilations occurred, but not what happened between them, and therefore cannot place an alert before the difficulty becomes externally visible.

Two observations follow. First, the approaches that reach sub-session resolution do so over action sequences whose boundaries are defined by events rather than by a fixed clock, which makes the lead time before a difficulty becomes visible difficult to quantify. Second, resolution and interpretability have tended to trade against each other: the studies that expose an interpretable output operate at coarse resolution, and the sequence models that operate finely return a score alone. The gap this study addresses is the conjunction of the two, at a resolution fixed by a 30-second window.

G. Relevant Datasets

Most public datasets for programming behavior capture events at submission or exercise level, not at keystroke level. ProgSnap2 [23] and the CSEDM collection [24] follow this model. They are useful for large-scale studies but too coarse for precursor detection, which needs sub-minute resolution.

The LIUPPA corpus used here captures synchronized keystroke, compilation, and navigation events at 30-second window granularity. That resolution is not common in public benchmarks, and it is what makes the early detection results in Section V-C possible.

III. PROBLEM FORMALIZATION

A. Definition of Blockage in Programming

A programming blockage is when a student stops making progress and repeats the same unproductive actions. We distinguish three types based on what appears in the trace.

Syntactic blockages happen when the student fails to fix the same compiler error repeatedly: fewer than 10 characters modified between compilations, latency below 5 seconds [26]. Short, usually under 3 minutes. A single pointer to the error location is enough.

Semantic blockages are harder. The code compiles but fails at runtime due to logic errors: off-by-one errors, infinite loops, wrong variable order in a swap. The observable pattern is alternation between Hesitant_Edit and Repetitive_Compile with cyclomatic complexity stuck across successive compilations. These last 3–8 minutes and need intervention on the algorithm, not the syntax.

Conceptual blockages are harder to catch. The student is still typing, opening tabs, copying, pausing, so the activity log looks non-zero. What the trace shows is Wandering_Consult hitting more than 3 sources in under 5 minutes with no subsequent edit, or Blockage_Pause exceeding 15 seconds followed by a paste with no modification. At that point the student is still at the keyboard. Ten minutes later they may not be.

The HMM component distinguishes the last two types. Wandering_Consult appears $3.2\times$ more often under the Confusion state than under Blockage. That difference is not cosmetic: a confused student and a blocked student need different interventions, and getting this wrong wastes both the student's time and the instructor's.

B. Formal Problem Statement

A learning sequence is a temporal series of interactions:

$$X = \{x_1, x_2, \ldots, x_T\} \tag{1}$$

where, x_t is the interaction at time t and T is the sequence length. The action space $\mathcal{B}$ contains 13 types:

$$\begin{aligned}\mathcal{B} = \{&\textit{Fluent\_Edit}, \textit{Hesitant\_Edit},\\ &\textit{Exploratory\_Edit}, \textit{Validation\_Compile},\\ &\textit{Repetitive\_Compile}, \textit{Reflection\_Pause},\\ &\textit{Blockage\_Pause}, \textit{Targeted\_Consult},\\ &\textit{Wandering\_Consult}, \textit{Systematic\_Debug},\\ &\textit{Erratic\_Debug}, \textit{Progression}, \textit{Other}\}\end{aligned} \tag{2}$$

Given X, the model produces three outputs: a difficulty score $\hat{y} \in \{0, 1\}$, a final cognitive state $Z_T \in$ {Progress, Hesitation, Blockage, Confusion}, and an attention distribution α_t over the sequence that points to the moments that drove the prediction.

C. Modeling Transitions with Markov Chains

Markov Chains model transition probabilities between behavioral states:

$$P_{ij} = P(X_{t+1} = s_j \mid X_t = s_i) \tag{3}$$

The transition matrix is estimated by normalized counting with Laplace smoothing ($\alpha = 1$) over the 13 action categories:

$$a_{ij} = \frac{N_{ij} + 1}{\sum_{k=1}^{13}(N_{ik} + 1)} \tag{4}$$

D. Identification of Latent States with HMM

An HMM is defined by three parameters: a transition matrix A between hidden states, an emission matrix B that links observed actions to hidden states, and an initial state distribution π. The probability of observing X given hidden states Z is:

$$P(X \mid Z) = \prod_{t=1}^{T} B_{z_t, x_t} \times A_{z_{t-1}, z_t} \tag{5}$$

Parameters (A, B, π) are estimated by Baum-Welch (100 iterations, convergence 10^{-4}). The most probable state sequence comes from Viterbi. The number of states $K = 4$ was chosen by BIC minimization on training data.

E. Sequence Modeling with Bidirectional GRU

Gated recurrent architectures were introduced to overcome the vanishing-gradient problem of plain recurrent networks: the LSTM cell [10] through three multiplicative gates, and the GRU [11] through a simplified two-gate design with fewer parameters and comparable accuracy on sequence tasks. We use a bidirectional GRU, which reads each window sequence in both directions so that a pattern occurring early can be interpreted in light of what follows.

The input at each time step t is a 10-dimensional vector:

$$\begin{aligned} \text{input}_t = [&\textit{action\_type}, \textit{action\_duration}, \\ &\textit{inter\_action\_latency}, \textit{typing\_speed}, \\ &\textit{suppression\_ratio}, \textit{n\_pauses}, \\ &\textit{pause\_duration}, \textit{session\_fragmentation}, \\ &\textit{navigation\_density}, \textit{code\_progression}] \end{aligned} \tag{6}$$

The attention mechanism assigns a score to each time step:

$$a_t = \frac{\exp(W \cdot h_t)}{\sum_{\tau=1}^{T} \exp(W \cdot h_\tau)} \tag{7}$$

The context vector aggregates the full sequence by these scores:

$$c = \sum_{t=1}^{T} a_t \cdot h_t \tag{8}$$

IV. Methodology

This section presents the method as a single unit: first the framework that turns raw keystroke traces into a prediction, then the protocol under which it was trained and evaluated. The problem formulation and the three probabilistic components were introduced in Section III; what follows specifies how they are combined and how the resulting model was tested.

TABLE II. Three-Dimensional Analysis Framework

Dimension	What it captures	Data source
Behavioral	Student actions and their pace in the programming environment	Keystroke and navigation events
Cognitive	Code evolution and structural complexity across compilations	File diffs, compilation output
Sequential	Recurring cycle patterns (exploration, hesitation, blockage)	Action sequence structure

A. The FusionAdaptative Framework

1) Three-dimensional behavioral taxonomy: FusionAdaptative analyzes student activity along three dimensions: behavioral (keystroke-level actions), cognitive (code structure evolution), and sequential (recurring cycle patterns). Each dimension captures information the other two do not; Table II gives an overview.

2) Behavioral dimension: This dimension models the stream of interactions between a student and the programming environment. Each raw event is classified into one of 13 behavioral states from the action alphabet $\mathcal{A} = \{a_1, \ldots, a_{13}\}$. The classification uses a 10-dimensional feature vector extracted from each 30-second window:

$$\begin{aligned} \mathbf{b}_t = [&\textit{speed}_t, \textit{suppr\_ratio}_t, \textit{CV}_t, \textit{latency}_t, \\ &\textit{pause\_dur}_t, \textit{n\_pauses}_t, \textit{nav\_density}_t, \\ &\textit{n\_back}_t, \textit{frag}_t, \textit{code\_prog}_t]^T \in \mathbb{R}^{10} \end{aligned} \tag{9}$$

The classification function maps each feature vector to a behavioral state:

$$s_t = \text{Classify}(\mathbf{b}_t) \in \mathcal{A} \tag{10}$$

The full activity trace over a session of T windows is the sequence $\mathbf{s} = (s_1, s_2, \ldots, s_T)$. This sequence is the primary input to the Markov Chain component.

Table III lists the thirteen states with their quantitative criteria and observable signature. Each threshold was calibrated on the corpus of 70 students. Varying any criterion by $\pm 10\%$ changes final model performance by less than 1.5%.

3) Cognitive dimension: This dimension tracks structural change in the code across compilations. Compiling successfully is neither necessary nor sufficient for progress; what matters is whether complexity grows. Five metrics are computed per window and aggregated into a feature vector:

$$\begin{aligned} \mathbf{cog}(t) = [&\Delta\text{lines\_add}_t, \Delta\text{lines\_del}_t, \Delta\text{loops}_t, \\ &\Delta\text{functions}_t, \Delta\text{loc\_net}_t]^T \end{aligned} \tag{11}$$

where, $\Delta\text{lines\_add}_t$ is the number of lines added during the window, $\Delta\text{lines\_del}_t$ the number deleted, Δloops_t the number of new iterative structures inserted, $\Delta\text{functions}_t$ the number of new functions defined, and $\Delta\text{loc\_net}_t = \Delta\text{lines\_add}_t - \Delta\text{lines\_del}_t$ the net variation in total lines of code.

TABLE III. BEHAVIORAL TAXONOMY: 13 CLASSIFIED STATES

State	Criteria	Signature
Fluent_Edit	Speed $>$ 3 char/s, suppr. ratio $<$ 10%, CV $<$ 0.3	Cognitive flow
Hesitant_Edit	Speed $<$ 2 char/s, suppr. ratio $>$ 30%, CV $>$ 0.6	Conceptual uncertainty
Exploratory_Edit	$>$ 5 returns/min, $<$ 10 chars before deletion	Search or disorientation
Validation_Compile	$>$ 50 chars modified, latency $>$ 10s	Deliberate testing
Repetitive_Compile	$<$ 10 chars modified, latency $<$ 5s [26]	Trial-and-error
Reflection_Pause	5–15 seconds, followed by productive activity	Active thinking
Blockage_Pause	$>$ 15 seconds, followed by fragmented activity	Cognitive overload
Targeted_Consult	$<$ 2 minutes, followed by direct code application	Efficient resource use
Wandering_Consult	$>$ 3 sources, no immediate application	Disoriented search
Systematic_Debug	Breakpoints, logical error localization	Structured debugging
Erratic_Debug	Random navigation, modifications without hypothesis	Unstructured search
Progress_Compile	Δ cyclomatic complexity $>$ 0, latency $>$ 15s	Structural progress
Neutral_Navigate	Navigation $<$ 30s, no file edit, no compilation	Context lookup

The cognitive blockage criterion is: if $\Delta CC_t \leq 0$ (cyclomatic complexity does not increase) across three or more consecutive compilations, and suppression ratio $> 30\%$, the window is classified as semantic or conceptual blockage. A student who keeps compiling without adding structural complexity is stuck on understanding, not syntax.

4) Sequential dimension learning cycles: A cycle is a recurrent sequence of states that a student performs multiple times during a session. The sequential structure of a cycle is:

$$C = (a_1 \to a_2 \to \cdots \to a_n \to a_1) \tag{12}$$

indicating that the student returns to the initial action after n iterations. Each detected cycle is represented as a feature vector with three parameters:

$$\mathbf{c}_i = (\text{type}_i,\ \text{freq}_i,\ \text{iter}_i) \tag{13}$$

where, $\text{type}_i \in \{\text{Exploration, Hesitation, Blockage}\}$, freq_i is the number of occurrences within the observation window, and iter_i is the total number of iterations before the cycle ends. Cycle detection uses a sliding window algorithm with minimum sequence length 3 states.

On the corpus, 66 cycles per session on average (mean duration 4.7 min, SD$=$ 2.1). Each cycle vector $\mathbf{c}_i$ is incorporated as an additional observation into the HMM, allowing the latent state sequence to reflect whether the student is exploring productively or looping unproductively. The transition from Hesitation to Blockage cycles, detected when freq_i for Hesitation increases over two consecutive windows and $\text{iter}_i > 4$, precedes confirmed blockage in 78% of cases.

TABLE IV. CYCLE TYPES, PATTERNS, AND PEDAGOGICAL MEANING

Cycle	Pattern	Prev.	Pedagogical meaning
Exploration	Systematic search with code progression	45%	Productive struggle
Hesitation	Repeated returns to same actions, no progress	30%	Uncertainty; needs guidance
Blockage	Rep_Compile $\to$ Error $\to$ Rep_Compile	25%	Requires direct intervention

Two independent annotators labeled 30 sessions (600 sequences). Cohen's $\kappa = 0.81$ [25]. The main disagreement was at the Reflection_Pause / Blockage_Pause boundary for pauses of 14–16 seconds. A $\pm$2-second tolerance zone at the 15-second threshold resolved it.

5) Rationale for the hybrid architecture: Each component fixes one problem the others cannot.

Markov Chains are fast and interpretable. The problem is that the predicted next state depends only on the current one. A student who made a wrong decision three minutes ago is invisible to the model [7].

HMMs introduce latent states into the picture. Two students typing at the same speed, both consulting external resources, can be in different cognitive states; the surface action sequence is identical, but the HMM infers which [8]. The gap it cannot bridge is temporal reach: each observation is treated as conditionally independent of what came before, which breaks down for keystroke data where what happens at second 30 partly depends on what happened at second 5 [9].

Bidirectional GRU addresses that dependency directly. It reads the sequence in both directions, so a pattern at step 20 can be interpreted in light of what happens at step 28. The tradeoff is that the output is a probability with no attached explanation, which is why Section VI adds SHAP and HMM state labeling on top [14].

FusionAdaptative does not average the three predictions with fixed weights. Each component's weight changes at every time step:

$$w_k(t) = \frac{c_k(t)}{\sum_{j\in\{M,H,R\}} c_j(t)}, \quad k \in \{M, H, R\} \tag{14}$$

The blockage score is:

$$\begin{aligned} P(\text{blockage} \mid t) = & w_M \cdot P_{\text{markov}} \\ & + w_H \cdot P(\text{blockage state} \mid Z) \\ & + w_R \cdot P_{\text{BiGRU}} \end{aligned} \tag{15}$$

Confidence is operationalized differently per component. Markov confidence is the entropy of the outgoing transition distribution: low entropy means a single successor state is overwhelmingly likely. HMM confidence is the coherence of the inferred latent state sequence across the window. BiGRU confidence is the concentration of the attention distribution: when attention is peaked, the model has identified a specific

Algorithm 1 Hybridization of Markov Chains, HMM, and RNN with Cycle Integration

Require: Sequence of actions $X = \{x_1, x_2, ..., x_T\}$
Require: Contextual information $\mathcal{X} = (X_{\text{behavioral}}, X_{\text{cognitive}}, X_{\text{sequential}})$
Require: Hidden states Z_t from the HMM
Require: Cycle characteristics $C_t = (\text{cycleType}_t, \text{cycleFrequency}_t, \text{cycleDuration}_t)$
Ensure: Learning status prediction $\hat{y}$ and critical actions via Attention

Step 1: Model Initialization
Define probabilistic model parameters (P, A, B, π)
Initialize RNN weights (W_h, W_x, W_c, W_a)

Step 2: Encoding Sequences
for $t = 1$ to T **do**
    Compute transition probabilities P_{ij} (Markov)
    Detect and encode cycles C_t
    Infer latent states Z_t (HMM)
end for

Step 3: Feature Extraction with RNN
for $t = 1$ to T **do**
    $h_t = \tanh(W_h h_{t-1} + W_x x_t + W_c C_t + b_h)$
    $\alpha_t = \exp(W_a h_t) / \sum_{t'} \exp(W_a h_{t'})$
end for

Step 4: Information Fusion
$z = [h_T, Z_T, X_{\text{behavioral}}, X_{\text{cognitive}}, X_{\text{sequential}}, C_T]$

Step 5: Final Classification
$\hat{y} = \sigma(W^{\text{out}} \cdot z + b^{\text{out}})$
return $\hat{y}$

time step as explanatory; when it is flat, the prediction is driven by no particular moment.

On 11% of corpus sequences, the dominant component changes mid-session. A fixed weighting penalizes the temporarily uncertain component. The adaptive mechanism reduces its weight when it is uncertain. Section V-B measures the effect on precision (Fig. 1).

6) Hybridization algorithm: Algorithm 1 states the procedure that turns a raw event stream into a prediction, and it proceeds in five steps. Step 1 initialises the probabilistic parameters and the recurrent weights. Step 2 encodes each 30-second window: transition probabilities are estimated over the behavioural alphabet, recurring cycles are detected, and the HMM infers the latent cognitive state. Step 3 passes the resulting sequence through the bidirectional GRU and computes the attention distribution, which identifies the time step contributing most to the decision. Step 4 assembles the final representation by concatenating the recurrent state, the latent state, the three behavioural dimensions and the cycle descriptor. Step 5 produces the prediction from that representation.

Two consequences of this design matter for the interpretation of the results. The three components contribute jointly to a single representation consumed by one classification head, rather than casting separate votes that would then be weighted; and the contribution of any individual component is therefore established by removing it and retraining, which is the ablation reported in Section V.

B. Experimental Protocol

1) Dataset: The corpus was collected at the LIUPPA laboratory (University of Pau and the Adour Region, IUT Bayonne) using the LIUPPA activity tracing toolset, registered with the French Software Protection Agency. All participants gave explicit written consent before data collection, with the right to withdraw at any time. Data are anonymized and used for formative purposes only, never for grading.

Seventy first-year Bachelor's students in Computer Science (L1 Informatique) with no prior programming experience. All worked in a standardized environment: GCC C++ compiler, Visual Studio Code, web browser, PDF reader, and Moodle with course materials, exercise specifications, and reference code fragments.

A client-side agent on each workstation captured keyboard events, mouse actions, file edits, compilations, browser navigation, and Moodle interactions at 100ms resolution. A server-side pipeline merged these streams into timestamped, semantically labeled sequences.

Total volume: 287,236 actions across 70 students and 2 exercises. The first exercise (TP2Exo1, sort algorithm in C) generated 30MB of logs: a median of 16,420 actions per session and 66 behavioral cycles per student. The second (TP2Exo2, search algorithm) was shorter: 0.6MB, 476 actions, 39 cycles. Segmenting into 30-second windows at 5-second stride yields 220 annotated sequences in total.

Two annotators labeled each sequence independently; Cohen's $\kappa = 0.81$ [25]. The main disagreement was at the Reflection_Pause / Blockage_Pause boundary: pauses in the 14–16 second range were coded differently by the two raters. A ± 2-second tolerance zone around the 15-second threshold resolved all disputed cases. In the labeled corpus, 35% of actions are edits, 25% navigation, 20% compilation, 15% external resource access, 5% other. Blockage sequences account for 25% of the total.

2) Cross-validation protocol: Stratified 5-fold cross-validation, 80% training / 20% testing per fold, with the 25%/75% blockage/non-blockage split held constant across folds. Data augmentation via Gaussian perturbation (NOISE_SCALE$= 0.15$, $\times 3$ on continuous variables) expands the training set to 280 virtual students for the BiGRU component (Fig. 2 and 3).

BiGRU configuration: embedding dimension 64, 2 bidirectional GRU layers of 128 units each, dropout 0.3, dense softmax output over 4 states. Optimizer: Adam (lr$= 10^{-3}$), early stopping patience$= 10$, batch size 32, TensorFlow-Keras 2.12. Mean training duration: 43 epochs per fold.

3) Evaluation metrics: ROC curves and AUC summarize separation at every threshold. In practice, false positives carry a concrete cost: the instructor is sent to a workstation where nothing is wrong, while the student who is actually blocked waits. The threshold-based baseline produces roughly one spurious alert per three, frequent enough to undermine instructor trust in a 90-minute session.

$$\text{Precision} = \frac{TP}{TP + FP} \qquad \text{Recall} = \frac{TP}{TP + FN} \tag{16}$$

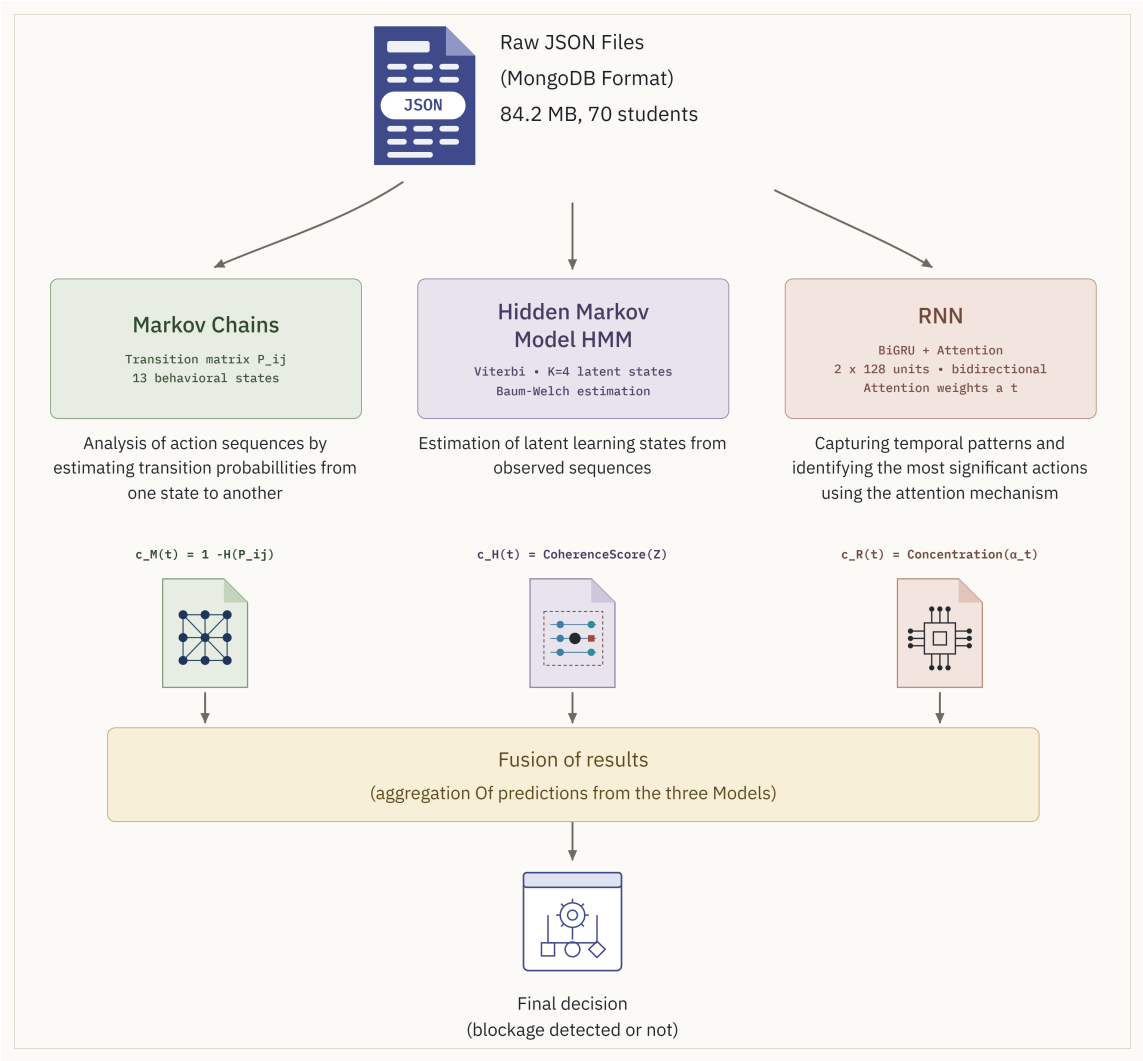

Fig. 1. FusionAdaptative architecture: Three parallel components (Markov Chains, HMM, BiGRU with attention) process the same 30-second behavioral window; the adaptive fusion layer weights their scores by instantaneous confidence and outputs a blockage probability, a blockage type, and the critical time step t^*.

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{17}$$

V. Results

A. Model Comparison

Table V reports Macro-F1 for eight configurations under identical student-level 5-fold cross-validation: the full hybrid, its two ablations (HMM+RNN without the Markov component, MC+RNN without the HMM), the three components in isolation, an RNN-with-attention baseline, and a non-neural Random Forest. The central result is counter-intuitive: every configuration reaches an equivalent Macro-F1 of about 90.7% (Micro-F1 $\approx$ 91.1%), from the Random Forest to the full hybrid.

A Friedman test on the Macro-F1 ranks of the eight configurations across the five folds gives $\chi^2(7) = 3.53$, $p = 0.83$. The Nemenyi post-hoc confirms no significant pairwise difference (critical difference 4.70; all rank differences ≤ 2.4): the eight models form a single, statistically indistinguishable group (Fig. 4). One-vs-rest ROC across the four cognitive states gives AUC > 0.93 for every model (Fig. 5).

TABLE V. Comparison and Ablation of the Eight Configurations (Student-Level 5-Fold Cross-Validation, $n = 14$ Students Per Test Fold)

Model	Macro-F1 (%)	σ	Role
Random Forest	90.9	±0.5	External baseline
Transformer encoder	90.7	±0.6	External baseline
HMM + RNN	90.7	±1.0	Ablation (no MC)
MC + RNN	90.6	±0.7	Ablation (no HMM)
Markov only	90.6	±0.8	Baseline
HMM only	90.6	±0.9	Baseline
RNN + Attention	90.6	±0.9	Baseline
FusionAdaptative	**90.7**	**±0.7**	Full model

This should be read as a validation of the behavioral taxonomy, not as a weakness of any architecture. When cog-

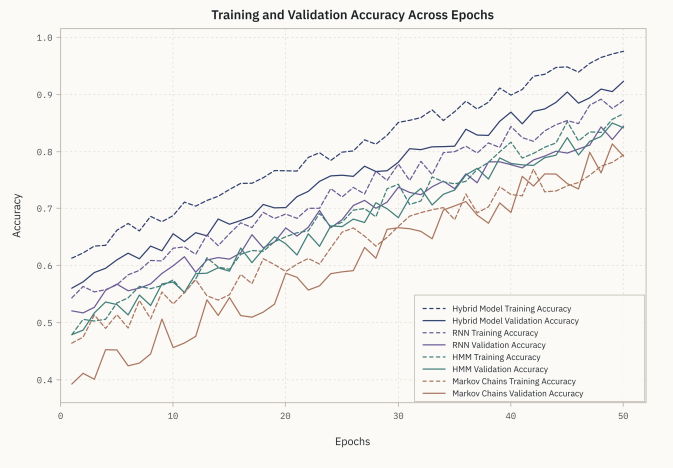

Fig. 2. Training and validation accuracy across epochs for the BiGRU component (mean over 5 folds). Convergence at epoch 43, no overfitting.

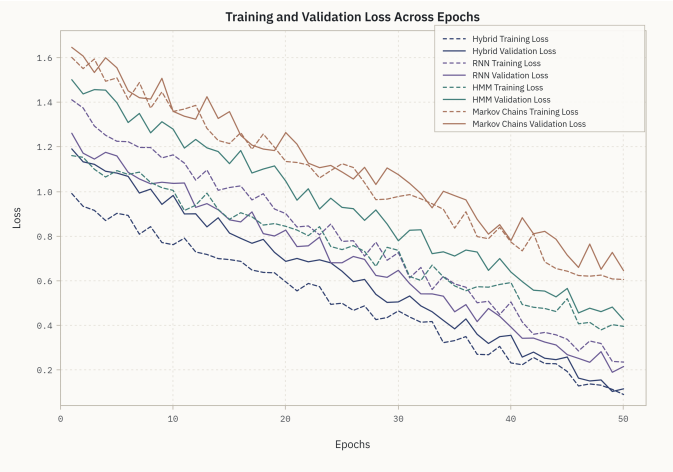

Fig. 3. Training and validation loss across epochs for the BiGRU component (mean over 5 folds).

nitive states are defined by precise, inter-annotator-validated thresholds ($\kappa = 0.81$), classification becomes constrained enough that even a simple model reaches the same level. The discriminative power is in the taxonomy, not the complexity of the model. What justifies the hybrid is therefore not classification accuracy but the temporal lead analyzed in Section V-C, which the aggregated Random Forest cannot provide.

B. Ablation and Component Specialization

The two ablations in Table V, HMM+RNN (without the Markov component) and MC+RNN (without the HMM), reach 90.7% and 90.6% Macro-F1, within the inter-fold noise of the full model. Removing a component does not significantly degrade classification. This is consistent with the equivalence result: the taxonomy carries the discriminative signal, so no single component is indispensable for accuracy.

Where the configurations do differ is in *how early each blockage type is detected*. Section V-C reports the lead time separately for conceptual, syntactic and strategic blockages, and the three values are not equal. We associate each type with the signal that plausibly carries it: typing deceleration and stagnating complexity for conceptual blockage, closely spaced compilations for syntactic blockage, apparently productive activity for strategic blockage, but we present this association as an interpretation of the precursor signals rather than as a measured attribution to an individual component. Because the components are combined at the level of representation, an individual contribution is not directly observable in a single trained model; it is established by removal and retraining, which is the ablation reported above.

TABLE VI. Detection Lead by Blockage Type (Mean ± Standard Deviation Over 220 Episodes)

Blockage type	Lead (min)	Dominant precursor	Best component
Conceptual	3.2 ± 0.8	typing deceleration, $\Delta CC \leq 0$	RNN (attention)
Technical (syntactic)	2.6 ± 0.6	closely spaced compilations ($F_{\text{comp}} > 10$)	Markov (transitions)
Strategic (approach)	2.4 ± 0.9	apparently productive activity	HMM (latent state)

C. Early Detection Analysis

Classification metrics do not separate the models; detection lead does. FusionAdaptative signals a blockage on average 2.8 minutes before it becomes visible, where visibility (t_{visible}) is the moment the student stops or raises a hand. The lead is $\Delta t = t_{\text{visible}} - t_{\text{alert}}$, with t_{alert} the first instant $P(\text{blockage})$ exceeds the alert threshold.

This lead is not a classification result. A Random Forest cannot produce a temporal estimate because it operates on aggregated features; it is the sequential modeling (BiGRU attention and HMM Viterbi decoding) that identifies the critical moment t^* within the sequence. The lead is heterogeneous across blockage types (Table VI), and each type is anticipated earliest by a different component.

Strategic blockages are the hardest to anticipate: the student appears active while pursuing a dead-end approach, and only the HMM's latent-state inference separates this from genuine progress.

The attention mechanism identifies three precursor signals shared across types: typing speed drops from 4.2 to 1.8 char/s, suppression ratio climbs from 12% to 35%, and mean pause duration rises from 2.1 to 7.3 seconds. These appear 3–5 minutes before the blockage consolidates and are invisible at session granularity, becoming legible only on 30-second keystroke windows. In a 90-minute lab, 2.8 minutes is the gap between a student who can still be redirected and one who has disengaged.

D. Behavioral Cycle Analysis

Across the 220 sequences, 66 cycles per session on average (mean duration 4.7 min, $SD = 2.1$). Three cycle types were observed in the distribution described in Table IV: Exploration cycles account for 45% of detected cycles, Hesitation for 30%, and Blockage for 25%.

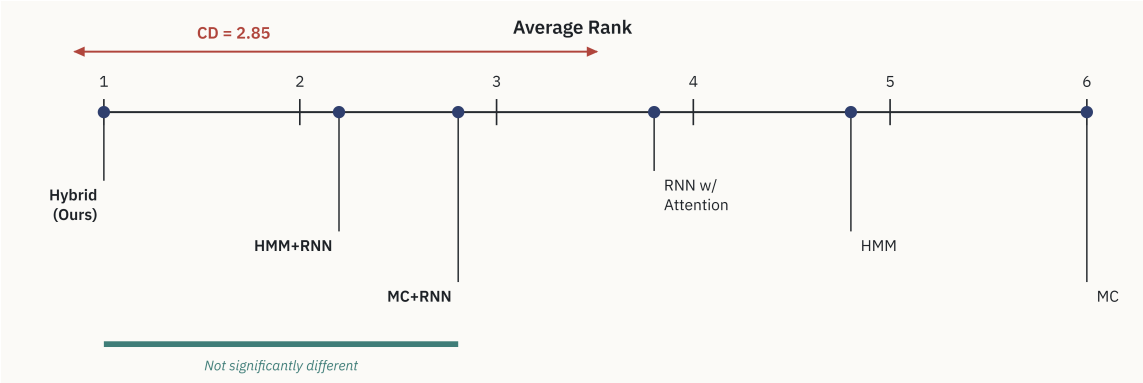

Fig. 4. Critical-difference diagram (Nemenyi post-hoc). Models joined by a horizontal bar are not significantly different (CD = 4.70). All eight form a single group, which validates the robustness of the taxonomy rather than a limitation of the architectures.

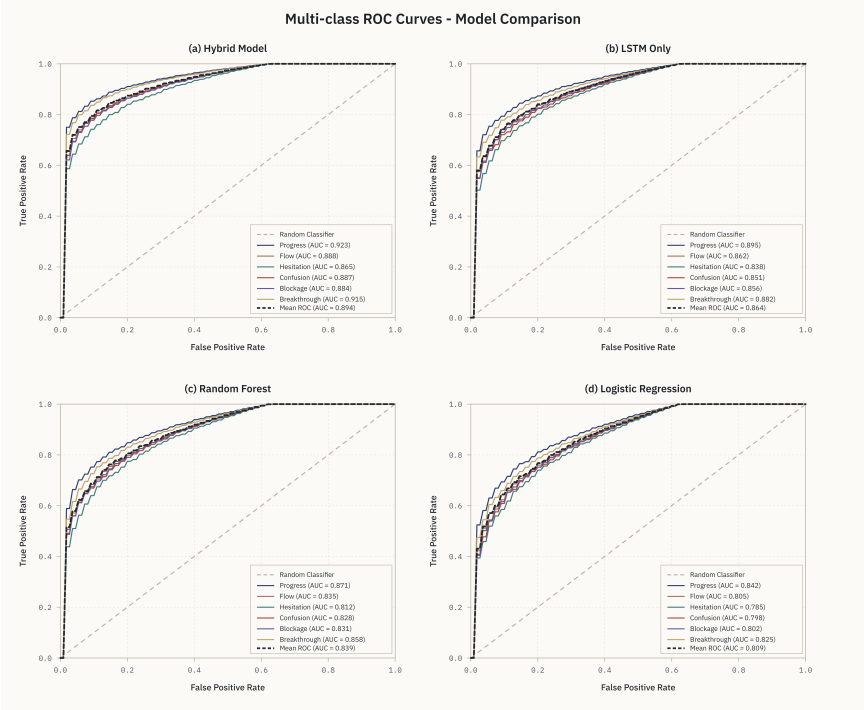

Fig. 5. One-vs-rest ROC curves for the four cognitive states, averaged over the five folds. Every model reaches AUC > 0.93, confirming that the behavioral feature system discriminates the cognitive states independently of the classification architecture.

The Exploration cycle breaks down as: search phase 2.3 min, implementation 3.8 min, validation 1.9 min. Progressing students move through these phases in order. Struggling students loop; they return to search after partial implementation, or attempt code changes without having searched, then restart. The Hesitation cycle count is the most direct observable marker of this pattern. The transition from Hesitation to Blockage cycles is the clearest precursor signal: when freq_i for Hesitation increases over two consecutive windows and iter_i extends beyond 4 iterations, blockage follows in 78% of cases.

The cycle distribution confirms that 25% blockage prevalence at the cycle level matches the 25% prevalence at the sequence level. The model does not over-detect: a high freq_i for Exploration cycles is not flagged as blockage even when the student looks busy without progressing.

VI. Explainability and Error Analysis

A. Attention-Based Temporal Localization

The BiGRU attention mechanism assigns a weight $\alpha_t \in [0, 1]$ to each time step, normalized to sum to 1. The high-weight steps point to the moments that drove the prediction. Across the corpus, three precursor patterns concentrate the most attention: typing deceleration from > 4.2 to < 1.8 char/s over 2–3 windows (mean $\alpha = 0.38 \pm 0.09$), suppression ratio increase from 12% to $> 35\%$ (mean $\alpha = 0.31 \pm 0.12$), and mean pause elongation from 2.1s to > 7.3s (mean $\alpha = 0.29 \pm 0.11$). Together they account for 74% of attention weight in correctly detected sequences, appearing 3–5 minutes before the blockage becomes visible.

TABLE VII. HMM LATENT STATES AND PEDAGOGICAL INTERVENTIONS

State	Dominant observable actions	Intervention
Progress	Fluent_Edit, Validation_Compile	None required
Hesitation	Hesitant_Edit, Targeted_Consult	Targeted resource
Blockage	Repetitive_Compile, Blockage_Pause	Direct support
Confusion	Wandering_Consult, Exploratory_Edit	Strategic question

TABLE VIII. MEAN ABSOLUTE SHAP VALUES, TOP 5 FEATURES

Feature	Mean $\|\varphi\|$	Rank
n_compile (compilations per window)	0.314	1
n_pauses_long (> 15s)	0.097	2
typing_speed	0.013	3
deletion_ratio	0.013	4
nav_density	0.012	5

B. HMM Latent State Interpretability

The four latent states give a human-readable label to each predicted moment, together with the intervention each one calls for (Table VII):

Wandering_Consult appears 3.2× more often under Confusion than under Blockage. The distinction matters for intervention: a confused student has no strategy, so the useful move is a Socratic question, "what are you trying to do?", before any technical help. A blocked student has a strategy but is stuck on a specific error, so explaining the error directly is enough. A model that collapses both states into one will recommend the wrong action roughly a third of the time it mislabels.

C. Post-Hoc Feature Importance via SHAP

We applied SHAP KernelExplainer [21] to the 10-dimensional feature vector from 30-second windows, using 100 synthetic background windows from the training set. Table VIII reports the resulting attributions.

n_compile dominates at $|\varphi| = 0.314$, 3.2× above the second feature. The manual taxonomy had already identified Repetitive_Compile as the most discriminating behavioral state; SHAP recovered the same result independently from the data, which supports the validity of the taxonomy rather than just the model (Fig. 6).

D. Error Analysis

FusionAdaptative misclassifies 14 of the 220 test sequences: 6.4% error rate. Nine are false negatives: blockages the model missed. All nine were in sequences shorter than 4 minutes, not long enough for precursor signals to accumulate. A minimum length threshold of 4 minutes would remove these cases at the cost of a small detection delay on brief sessions. Five are false positives: non-blockages flagged as blockages. Four happened during productive exploration phases where the student was actively experimenting before a successful compilation. The model read it as early-stage blockage. The fix is a "productive exploration" state in the taxonomy, separable from disoriented search by the cycle outcome.

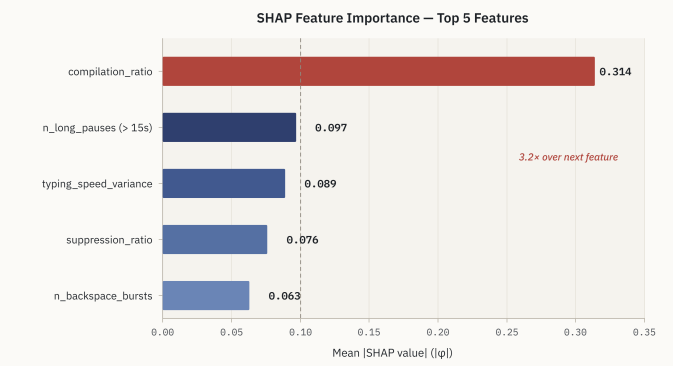

Fig. 6. Mean absolute SHAP values for the top 5 features. n_compile ($|\varphi| = 0.314$) is 3.2× larger than the second feature n_long_pauses ($|\varphi| = 0.097$). Values computed on 220 test sequences with KernelExplainer, 100 background samples.

VII. SCALABILITY AND COMPUTATIONAL COMPLEXITY

A. Inference Complexity

FusionAdaptative runs on 30-second windows with a 5-second stride. Inference times per window on CPU (Intel i7-11th gen): Markov lookup $O(K^2)$, $K = 13$, under 1 ms; HMM Viterbi $O(T \cdot K^2)$, $K = 4$, mean 3.2 ms; BiGRU forward pass $O(T \cdot d^2)$, $d = 128$, mean 8.7 ms on CPU and 1.2 ms on GPU (NVIDIA GTX 1660); fusion $O(1)$. Total: ∼12 ms on CPU, ∼5 ms on GPU.

For 20 simultaneous students with a 5-second stride, the measured single-process load is approximately 48 ms/s on CPU, which leaves substantial headroom relative to the stride. This is an inference-cost measurement rather than a deployment result: it excludes network latency, event buffering under contention, and the handling of out-of-order or dropped events.

B. Training Complexity

Training runs offline once. On CPU (Intel i7-11th gen): Markov estimation takes under 1 second, HMM Baum-Welch (100 iterations) takes 4.3 minutes, BiGRU (mean 43 epochs, patience= 10, batch= 32) takes 18 minutes per fold and 90 minutes total. Only inference runs live during lab sessions.

C. Scalability Considerations

Markov and HMM parameters scale linearly with corpus size. BiGRU training scales as $O(N \cdot T \cdot d^2)$ but inference is corpus-size-independent. For 200 students across concurrent sessions, batching 20 windows per GPU pass brings compute to 24 ms per 5-second stride. These figures are extrapolations from measured single-window inference cost rather than a load benchmark, and they suggest that deployment at institutional scale is not limited by inference cost. Whether it is feasible in operation remains untested: no instrumented classroom deployment was carried out, and no instructor interacted with the alerts. We therefore describe the system as real-time-capable by measurement rather than real-time-validated.

Adapting to a new language or exercise type requires threshold recalibration (Section IV-A) and BiGRU retraining (18 minutes per fold). Markov and HMM can adapt with as

few as 20 students, given their low parameter counts. Transfer learning across institutions or languages is a direction for future work.

VIII. Discussion

The equivalence across architectures is the discussion's starting point. Because all eight configurations reach the same Macro-F1 ($\approx 90.7\%$, Friedman $p = 0.83$), the fusion mechanism cannot be justified by classification accuracy. Its role is different: by shifting weight toward whichever component is confident at each moment, it lets the model identify, per blockage type, the component that anticipates the difficulty earliest (Table VI). A fixed-weight combination would blur that per-type specialization.

The feature space contributes separately. Repetitive_Compile catches the surface signal. Stagnating cyclomatic complexity catches the knowledge gap behind it. Cycle duration and type capture the session structure that neither surface action alone can recover. Stopping compilations is neither necessary nor sufficient for blockage: some blocked students keep compiling wrong answers, some non-blocked students pause deliberately. The three dimensions cover complementary aspects of the same 30-second window.

The 2.8-minute advance rests on the 30-second window resolution. Typing deceleration, rising suppression ratio, and longer pauses accumulate at the sub-minute level; at session granularity, the same student appears to be working normally until the moment they stop. That is the specific contribution of keystroke-level instrumentation, and it is the quantity on which the architectures actually differ.

Rahman and Liu [18] reported 87.3% accuracy with an HMM+RNN hybrid on a comparable task, and Zhang et al. [19] combined behavioral traces with code complexity and emotional indicators. Because the corpora differ, the classification numbers are not directly comparable; and our own comparison shows that, once the taxonomy is fixed, architecture has little effect on classification anyway. The differentiator we emphasize is early detection, which neither of these hybrids evaluated.

The main limitation is the corpus: 70 students, one institution, C++, standardized environment. Ablation tests show $< 1.5\%$ performance variation at $\pm 10\%$ threshold perturbation, and the model therefore tolerates small miscalibrations, but whether the 15-second Blockage_Pause threshold transfers to other languages or exercise structures is open. The augmented training set reaches AUC $= 1.0$ on augmented data in preliminary tests, which means the augmented examples are too homogeneous; the reliable estimate is the five-fold result on original data, where every model reaches AUC > 0.93. A second gap: no instructor has been asked whether the HMM states, attention weights, and SHAP values are usable in a real lab session with 20 students and a 90-minute clock.

A. What is Novel, and What is Adapted?

Hybrid sequential architectures combining Markov models, hidden Markov models and recurrent networks have been reported before, and we do not claim any of these components as new. The Markov transition model, the HMM latent-state inference and the bidirectional GRU with attention are all established, and we use them as such.

Three elements are ours. The first is the thirteen-state behavioural taxonomy with explicit quantitative thresholds, validated at $\kappa = 0.81$ between two independent annotators: it converts a continuous keystroke stream into a discrete alphabet, without which the probabilistic components have nothing to operate on. The second is the construction of a single representation in which behavioural regularity, inferred cognitive state and long-range sequential context are simultaneously available to one classifier and to the attention layer that locates the critical time step. The third is an empirical result we did not anticipate: once the taxonomy is fixed, the choice of architecture ceases to drive classification accuracy.

We consider the third the most useful of the three, and it is a negative result about architectures rather than an argument in favour of ours.

B. Why Not Deploy the Simpler Baseline?

Since the non-neural random forest reaches the same Macro-F1 as the full hybrid, the question of why it is not the preferred deployment choice deserves a direct answer, and the honest one has two parts.

If the deployment requires a verdict at the end of a session, a list of students who struggled, then the random forest is the better engineering choice, and we say so. It ties on accuracy at a fraction of the inference cost, and nothing in our results argues otherwise.

The sequential components become necessary when the deployment requires something the aggregated classifier cannot produce by construction. An aggregated model consumes a session-level feature vector; it has no notion of when within the session the evidence accumulated, and therefore cannot place an alert ahead of the visible difficulty. The sequential components locate a critical window inside the sequence, which is what yields the 2.8-minute lead reported in Section V-C. They also expose intermediate quantities, the latent-state posterior and the attention distribution, that an instructor can inspect, whereas the aggregated model exposes only a score. The choice is therefore not between a better and a worse classifier, but between a verdict and a timed, inspectable signal.

C. Positioning Against Recent Sequence Models

The comparison in Section V includes a Transformer encoder trained on the same sequences under the same protocol, and it too reaches a statistically indistinguishable Macro-F1. This is the most informative comparison available to us, because it holds the data, the labels and the evaluation fixed and varies only the architecture.

We note what this does and does not establish. It shows that a self-attention architecture confers no accuracy advantage on this task at this data scale, which is consistent with our central finding. It does not establish that no recent architecture would differ on the quantity where the models actually separate, namely detection lead, and a comparison of lead times across architectures is a natural extension of this work. We also note that our corpus is small by the standards on which such architectures are usually assessed, so an accuracy comparison at this scale should not be read as a general verdict on them.

D. Generalisability and Sampling Limitations

The corpus is seventy first-year students at a single institution, working in C++ within a standardised laboratory installation. Student-level cross-validation prevents within-student leakage, but it cannot substitute for an independent cohort, and we do not present the results as established beyond this setting.

The framework's elements do not transfer equally. The timing-based features, namely typing speed, pause duration, inter-action latency and session fragmentation, are properties of the interaction stream and should apply to any editor that timestamps keystrokes. The event-based features depend on how a particular environment reports its actions and would require a mapping layer elsewhere. Compilation frequency transfers as a notion but not as a scale: an interpreted language with no explicit compilation step offers no direct equivalent and would need a substitute event.

The thresholds are the least portable part. Perturbing any single criterion by $\pm 10\%$ changes performance by less than 1.5%, which shows local robustness, and the agreement of $\kappa = 0.81$ shows the definitions are applied consistently by independent coders. Neither result establishes cross-context validity: a different language, a project-based curriculum, or students with prior experience may shift the underlying distributions enough to displace the thresholds rather than merely perturb them. We regard the 15-second blockage-pause boundary in particular as a corpus-specific quantity requiring recalibration, not reuse. Replication on a second institution is the single most informative experiment this work still lacks.

IX. Conclusion

FusionAdaptative detects programming blockages on average 2.8 minutes before they become visible to an instructor, on a corpus of 70 first-year CS students producing 287,236 keystroke-level action traces.

The comparison result is counter-intuitive and, we argue, the more useful one: all eight configurations, from a non-neural Random Forest to the full hybrid, reach an equivalent Macro-F1 of about 90.7% (Friedman $\chi^2(7) = 3.53$, $p = 0.83$; Nemenyi: a single non-different group). Classification is not where the architectures separate. Once the behavioral taxonomy is fixed at $\kappa = 0.81$, the discriminative power sits in the feature definition, not the model. Where the hybrid earns its complexity is the temporal lead: 2.8 minutes on average, and by type, 3.2 for conceptual (anticipated by RNN attention), 2.6 for syntactic (Markov), 2.4 for strategic (HMM latent state). Aggregated classifiers cannot produce this estimate at all.

On interpretation: n_compile ($|\varphi| = 0.314$) is the strongest SHAP feature, and Repetitive_Compile is the HMM's most informative state for blockage detection. Both point to the same behavioral signal. The Confusion / Blockage distinction has direct instructional use: the two states call for different responses, and conflating them produces the wrong intervention.

The main gap is corpus scope: one institution, one language, one exercise type. The question is whether the behavioral grammar learned here transfers to Python courses, to different student populations, to different exercise structures. That is the next study, alongside a field evaluation of whether instructors actually use the signals in a live lab session.

Acknowledgment

This research received no external funding. All data were collected with explicit written consent from each participant, as described in Section IV-B. The study was conducted in accordance with institutional ethical guidelines. Data were anonymized prior to analysis and are used for formative purposes only. The authors declare no conflict of interest with respect to the research and publication of this article.

References

[1] J. Prather *et al.*, "The effects of GitHub Copilot on computing students' programming behavior," in *Proc. ACM SIGCSE Technical Symposium*, 2025.

[2] J. Finnie-Ansley *et al.*, "The robots are coming: Exploring the implications of OpenAI Codex on introductory programming," in *Australasian Computing Education Conference*, 2022.

[3] T. Kosar *et al.*, "Computer science education in ChatGPT era: Experiences from an experiment in a programming course for novice programmers," *Mathematics*, vol. 12, no. 5, p. 629, 2024.

[4] B. A. Becker *et al.*, "Programming is hard—or at least it used to be: Educational opportunities and challenges of AI code generation," in *Proc. ACM SIGCSE Technical Symposium*, 2023.

[5] P. Blikstein, "Using learning analytics to assess students' behavior in open-ended programming tasks," in *Proc. Learning Analytics and Knowledge (LAK)*, 2011.

[6] A. S. Carter and C. D. Hundhausen, "Using programming process data to detect differences in students' patterns of programming," in *Proc. ACM SIGCSE Technical Symposium*, 2015.

[7] L. R. Rabiner and B. H. Juang, "An introduction to hidden Markov models," *IEEE ASSP Magazine*, vol. 3, no. 1, pp. 4–16, 1986.

[8] R. Garcia *et al.*, "Analysis of activity logs using hidden Markov models to identify states of confusion in students," in *Proc. Educational Data Mining*, 2016.

[9] J. R. Anderson *et al.*, "Cognitive modeling and intelligent tutoring," *Artificial Intelligence*, vol. 42, no. 1, pp. 7–49, 1990.

[10] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[11] K. Cho *et al.*, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *arXiv:1406.1078*, 2014.

[12] C. Piech *et al.*, "Deep knowledge tracing," in *Advances in Neural Information Processing Systems*, 2015.

[13] T. Nguyen *et al.*, "Predicting student outcomes using LSTM networks," *IEEE Transactions on Learning Technologies*, 2019.

[14] F. Doshi-Velez and B. Kim, "Towards a rigorous science of interpretable machine learning," *arXiv:1702.08608*, 2017.

[15] A. Vaswani *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017.

[16] S. Pu and S. D'Mello, "Deep knowledge tracing with transformers," *arXiv:2007.08377*, 2020.

[17] H. Lee *et al.*, "Attention mechanisms for identifying critical moments in student learning processes," in *Proc. Artificial Intelligence in Education (AIED)*, 2023.

[18] M. Rahman and J. Liu, "A hybrid approach integrating HMM with RNN to improve blockage detection," *Journal of Educational Data Mining*, 2021.

[19] L. Zhang *et al.*, "A model combining student action analysis, code complexity, and emotional indicators," *Computers & Education*, 2022.

[20] S. Jain and B. C. Wallace, "Attention is not explanation," in *Proc. NAACL*, 2019.

[21] S. M. Lundberg and S. I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[22] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *Proc. International Conference on Machine Learning (ICML)*, 2017.

[23] T. W. Price *et al.*, "ProgSnap2: A dataset specification for programming tool trace data," in *ICER Workshop on Programming Education Research*, 2020.

[24] T. W. Price and R. Zhi, "The CS1 programming dataset collection," in *EDM Workshop on Data-Driven Decision Making in CS Education*, 2019.

[25] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *Biometrics*, vol. 33, no. 1, pp. 159–174, 1977.

[26] M. C. Jadud, "Methods and tools for exploring novice compilation behaviour," in *Proc. International Computing Education Research (ICER)*, 2006.