Revue ouverte d'ingénierie des systèmes d'information
2026, vol. 6, n° Spécial INFORSID 2025, 1-36 pages, DOI: 10.21494/ISTE.OP.2026.1447 ISTE OpenScience

Une approche hybride combinant Markov, HMM et RNN pour détecter les blocages des étudiants dans l'apprentissage de la programmation

Hybrid Approach Combining Markov, HMM, and RNN to Detect Learning Blockages in Programming Résumé

Grota Abdelkader[1], Mohammed Erritali[3], Patrick Etcheverry[2], and Thierry Nodenot[2]

[1]Laboratoire Data4Earth, Faculté des Sciences et Techniques, Béni Mellal, Maroc
[2]Laboratoire LIUPPA, Université de Pau et des Pays de l'Adour, IUT de Bayonne, France
[3]Laboratoire XLIM, Université de Poitiers, Poitiers, France

RÉSUMÉ. Comprendre le processus d'apprentissage en programmation constitue un défi complexe en raison de la nature séquentielle et multidimensionnelle des interactions des étudiants avec les environnements numériques. Cette étude analyse les journaux d'activité de 70 étudiants débutants en informatique, placés en situation de résolution de problèmes de programmation, afin de détecter les difficultés rencontrées et de proposer des interventions pédagogiques adaptées. Nous présentons une approche hybride combinant des Chaînes de Markov pour modéliser les transitions entre types d'actions, des Modèles Cachés de Markov (HMM) pour inférer les états latents d'apprentissage (progression, hésitation, blocage) et des Réseaux Neuronaux Récurrents (RNN) enrichis d'un mécanisme d'attention pour repérer les moments critiques. Cette combinaison permet d'exploiter simultanément les dimensions comportementale, cognitive et séquentielle de l'apprentissage. La méthodologie consiste à extraire des caractéristiques temporelles et structurelles des journaux d'activité, à modéliser les cycles d'exploration, d'hésitation et de blocage, et à les intégrer dans un cadre unifié pour prédire les états d'apprentissage. Les données proviennent de plusieurs sessions standardisées de travaux pratiques, totalisant plus de 80 MB de traces horodatées, collectées avec le consentement des participants et stockées dans une base NoSQL. Chaque session est divisée en séquences correspondant à des phases cohérentes de résolution de problème, permettant une analyse fine des trajectoires d'apprentissage.Les résultats expérimentaux montrent que le modèle hybride proposé surpasse les approches traditionnelles, et atteignant une précision de 93,5% et réduisant significativement les faux positifs dans la détection de blocages. L'analyse multidimensionnelle offre une compréhension plus riche des trajectoires d'apprentissage, y compris dans leurs dimensions invisibles telles que l'engagement non productif ou les phases de flottement.Cette recherche ouvre la voie à des plateformes éducatives intelligentes capables de fournir un feedback personnalisé en temps réel, sensible aux micro-indicateurs d'activité, à l'intensité cognitive et au contexte individuel, contribuant ainsi à une meilleure réussite et un engagement renforcé des étudiants.

ABSTRACT. Understanding the learning process in programming poses a complex challenge due to the sequential and multidimensional nature of students' interactions with digital environments. This study analyzes the activity logs of 70 beginner computer science students engaged in problem-solving programming tasks to identify encountered difficulties and propose tailored pedagogical interventions.We present a hybrid approach combining Markov Chains to model transitions between types of actions, Hidden Markov Models (HMM) to infer latent learning states (progress, hesitation, blockage), and Recurrent Neural Networks (RNN) enhanced with an attention mechanism to detect critical moments. This combination allows for the simultaneous exploitation of behavioral, cognitive, and sequential dimensions of learning.The methodology involves extracting temporal and structural features from the activity logs, modeling cycles of exploration, hesitation, and blockage, and integrating them into a unified framework to predict learning states. The data comes from several standardized practical work sessions, totaling over 80 MB of timestamped traces, collected with consent and stored in a NoSQL database. Each session is divided into sequences corresponding to coherent problem-solving phases, enabling a fine-grained analysis of learning trajectories.Experimental results show that the proposed hybrid model outperforms traditional approaches, achieving an accuracy of 93.5% and significantly reducing false positives in blockage detection. The multidimensional analysis offers a richer understanding of learning trajectories, including their invisible dimensions such as unproductive engagement or phases of uncertainty.This research paves the way for intelligent educational platforms capable of providing personalized real-time feedback, sensitive to micro-indicators of activity, cognitive intensity, and individual context, thereby contributing to improved student success and enhanced engagement.

MOTS-CLÉS. journaux d'activité, apprentissage de la programmation, modèle hybride, RNN, HMM, chaînes de Markov, difficultés des étudiants

KEYWORDS. Activity logs, programming learning, hybrid model, RNN, HMM, Markov chains, student difficulties

Introduction

Former des étudiants au métier de développeur est un défi pédagogique majeur. Tout enseignant en programmation connaît cette scène : dans une même classe, quelques étudiants progressent rapidement, identifient et corrigent leurs erreurs avec méthode, tandis que d'autres semblent paralysés devant des difficultés récurrentes. Certains franchissent avec assurance les étapes de l'exercice, alors que d'autres effacent et réécrivent inlassablement la même portion de code, hésitent longuement sans parvenir à choisir une stratégie ou enchaînent des tentatives infructueuses. Ces situations de blocage, fréquentes mais parfois silencieuses, constituent l'un des défis majeurs de l'apprentissage de la programmation.

Dans un contexte où un seul formateur doit accompagner un grand nombre d'étudiants aux profils hétérogènes, le repérage en temps réel des apprenants nécessitant une intervention prioritaire devient un enjeu central. Une intervention rapide peut faire la différence entre un échec décourageant et une expérience constructive qui consolide la compréhension. Mais l'observation en continu des stratégies individuelles de chaque étudiant est difficile : l'enseignant doit arbitrer entre les sollicitations visibles et les difficultés invisibles, souvent détectées trop tard lorsqu'elles se traduisent par un décrochage.

Les environnements numériques de programmation ouvrent ici des perspectives intéressantes : chaque action (écriture de code, compilation, exécution, consultation de documentation, corrections successives…) peut être captée et horodatée dans un journal d'activité. Ces traces ne se limitent pas à une simple succession d'événements : elles portent en elles une dynamique – rythme, durée, répétitions – qui reflète une partie des processus cognitifs sous-jacents. Une séquence de corrections rapides et fluides peut témoigner d'une maîtrise en cours de consolidation, tandis qu'un long silence suivi de tentatives répétées peut signaler un blocage profond.

Exploiter ces traces pour en faire un outil pédagogique fiable n'est cependant pas trivial : elles sont volumineuses, hétérogènes et parfois ambiguës (une pause peut traduire autant une réflexion féconde qu'une perte de repères) (Baker et Yacef, 2009). L'analyse des traces d'utilisation est un domaine de recherche établi dans les Environnements Informatiques pour l'Apprentissage Humain (Choquet et al., 2007), mais l'enjeu reste de concevoir des modèles capables d'interpréter ces signaux complexes pour détecter les situations de difficulté et fournir aux enseignants des indicateurs pertinents.

Afin de structurer l'analyse, nous distinguons trois niveaux de granularité qui forment une hiérarchie logique des traces.

Un **enregistrement** correspond à la trace brute collectée lors d'une activité : il regroupe l'ensemble des événements horodatés (édition de code, exécution, compilation, etc.) captés dans un fichier JSON. Plusieurs enregistrements consécutifs réalisés par un même étudiant au cours d'une même séance de travail constituent une **session**, typiquement un TP de deux heures sur un exercice donné. Enfin, pour affiner l'analyse, chaque session est découpée en **séquences** : il s'agit de sous-ensembles cohérents d'événements, généralement délimités par un objectif intermédiaire (comme une tentative de correction).

Dans cette étude, nous proposons un modèle hybride de détection de blocages en temps réel qui exploite conjointement trois dimensions complémentaires :

— **Dimension Comportementale** : rend compte de la dynamique observable des interactions (séquences d'édition, pauses, corrections, navigation entre fichiers).

— **Dimension Cognitive** : reflète la qualité et la complexité des solutions produites (choix de structures de contrôle, régularité syntaxique, cohérence des corrections).
— **Dimension Séquentielle** : capture les dépendances temporelles et les régularités à long terme des actions des étudiants.

Pour combiner ces dimensions, nous mobilisons des outils issus de l'apprentissage automatique et de la modélisation probabiliste : **Chaînes de Markov** permettent de représenter les transitions entre types d'actions, **Modèles Cachés de Markov (HMM)** infèrent des états latents tels que "progression fluide", "hésitation" ou "blocage", et **Réseaux Neuronaux Récurrents (RNN)** enrichis d'un mécanisme d'attention exploitent l'ensemble de la séquence pour repérer les moments critiques où l'intervention de l'enseignant est la plus pertinente.

L'originalité de notre approche réside dans cette hybridation : en combinant robustesse probabiliste (Markov, HMM) et capacité des RNN à modéliser des dynamiques complexes, nous produisons des indicateurs multidimensionnels utiles pour guider les interventions pédagogiques. L'objectif est de transformer des données massives et hétérogènes en repères concrets permettant de réduire le poids des blocages invisibles et de favoriser la réussite des étudiants.

Cet article est structuré comme suit : la Section 1 présente l'état de l'art sur les approches de détection de difficultés en programmation ; la Section 2 décrit notre modèle hybride et ses composantes ; la Section 3 expose l'expérimentation et les résultats ; enfin, la Section 4 et 5 discute des apports, limites et perspectives.

1. Etat de l'art

Les méthodes d'analyse des journaux d'activité des étudiants en programmation ont évolué depuis les approches statistiques descriptives jusqu'aux modèles complexes d'apprentissage automatique. Une revue récente de la littérature confirme l'intérêt de ces approches pour comprendre et soutenir les apprenants (Omer et al., 2023). Une première génération de travaux reposait sur des indicateurs simples comme le nombre de tentatives, le temps passé sur une tâche, ou le taux de réussite (Villamor et al., 2020). Ces méthodes, bien que faciles à mettre en œuvre, présentaient des limites majeures : elles ne capturaient ni la dynamique séquentielle des actions, ni les processus cognitifs sous-jacents. Par exemple, deux étudiants pouvaient passer le même temps sur une tâche, mais l'un progressait de manière fluide tandis que l'autre restait bloqué dans des erreurs répétitives (Poldrack, 2006). Ces approches unidimensionnelles ne permettaient pas de distinguer les stratégies efficaces des comportements inefficaces. C'est dans ce contexte que l'exploration de données pour l'éducation (Educational Data Mining) a émergé pour proposer des méthodes plus fines (Baker et Yacef, 2009).

Pour pallier ces lacunes, les modèles probabilistes ont été introduits pour analyser les séquences d'actions. Les chaînes de Markov ont permis de modéliser les transitions entre étapes du processus de résolution de problèmes (Rafferty et al., 2015). Ces travaux ont montré que les étudiants les plus performants suivaient des séquences structurées, par exemple Édition $\rightarrow$ Exécution $\rightarrow$ Correction, tandis que les autres répétaient des cycles inefficaces tels que Exécution $\rightarrow$ Erreur $\rightarrow$ Retour. Cependant, ces modèles ne captaient pas les dépendances à long terme ni les dynamiques complexes, limitant leur utilité pour des trajectoires d'apprentissage non linéaires (Callut et Dupont, 2005).

Les modèles cachés de Markov (HMM) ont ensuite permis de modéliser des états latents comme la confusion ou le blocage, inférés à partir des actions observées (D'Mello et Graesser, 2012). Grafsgaard et al. (2011) ont par exemple utilisé des HMM pour identifier des états de confusion dans des environnements de tutorat, avec des résultats prometteurs pour la détection précoce. Malgré ces avancées, les HMM restaient limités par leur hypothèse de Markov forte et leur difficulté à capturer des relations non linéaires (Anderson, 2012), ce qui les rendait peu adaptés à des contextes où les comportements dépendent de multiples facteurs interdépendants (ex. complexité du code et émotions). Parallèlement, le champ des Learning Analytics s'est développé pour transformer ces données en indicateurs actionnables, appelant à une synergie avec l'EDM (Siemens et Baker, 2012).

Avec l'émergence de l'apprentissage profond, les réseaux de neurones récurrents (RNN) et leurs variantes comme les LSTM et GRU ont permis de capturer des dépendances temporelles complexes (Yousafzai et al., 2021). Kukkar et al. (2024) ont montré que l'ajout d'un mécanisme d'attention améliore significativement la prédiction de la performance des étudiants en exploitant la structure temporelle des séquences. Cependant, ces modèles nécessitent de grandes quantités de données, sont sujets au surapprentissage, et leurs sorties ne sont pas toujours facilement interprétables par les enseignants, ce qui limite leur utilité pédagogique.

Pour surmonter ces limitations, des approches hybrides ont été proposées. Certaines combinent des HMM et des RNN : les HMM identifient les états latents (progression, blocage), tandis que les RNN modélisent les séquences à long terme, améliorant la détection des blocages et la robustesse des prédictions (Tang et al., 2019). Cependant, ces approches restent souvent centrées sur une seule dimension (comme les actions) et ne prennent pas en compte d'autres aspects clés de l'apprentissage tels que la complexité du code produit ou les émotions.

Des travaux récents ont exploré des approches multidimensionnelles, intégrant simultanément des indicateurs comportementaux, cognitifs et séquentiels. Zhao et al. (2023) ont par exemple combiné mesures d'actions, complexité du code et pauses (indicateurs émotionnels) pour une analyse plus holistique. Ces travaux soulignent l'importance de combiner plusieurs dimensions, mais la plupart ignorent encore la dynamique temporelle fine et la question de l'interprétabilité (Chen et al., 2021).

Une attention croissante est également portée à l'analyse micro-comportementale. L'étude des patterns de frappe (keystroke dynamics) a révélé des corrélations significatives entre les métriques temporelles et les indicateurs de charge cognitive (Epp, Lippold et Mandryk, 2011). L'analyse des pauses, des cycles de correction et de la fragmentation des sessions permet d'identifier des stratégies d'apprentissage distinctes (Zhang et al., 2024), démontrant que la temporalité n'est pas un épiphénomène mais une caractéristique fondamentale des interactions dans les environnements de développement. Des approches similaires ont été appliquées avec succès à l'analyse des traces dans des contextes de jeux sérieux (Muratet et al., 2018).

L'analyse des frappes clavier pour l'inférence d'états cognitifs représente aujourd'hui un champ de recherche en pleine émergence. Kolakowski et al. (2023) montrent que les métriques de frappe peuvent être corrélées avec des mesures physiologiques de stress, ouvrant la voie à une détection plus fine des moments de surcharge cognitive. Notre approche prolonge ces travaux en intégrant l'analyse des intentions d'écriture et des stratégies de correction pour modéliser plus complètement les processus mentaux.

Enfin, l'émergence des systèmes d'apprentissage adaptatifs (comme ALEKS ou DreamBox) montre que l'exploitation des traces comportementales peut permettre une personnalisation fine de

l'apprentissage (Matayoshi et al., 2019 ; Evidence for ESSA, 2022). Ces systèmes adaptent en temps réel la difficulté des exercices en fonction du profil de l'apprenant. En France, la communauté des Learning Analytics s'est structurée pour développer des outils et méthodes adaptés (Iksal et Lefèvre, 2020), produisant un état de l'art des approches issues de la recherche nationale (Iksal et al., 2020). Notre contribution s'inscrit dans cette évolution en proposant un mécanisme de détection précoce des difficultés et de génération d'indicateurs actionnables pour l'enseignant..

En résumé, l'état de l'art montre un progrès continu dans l'analyse des journaux d'activité, mais les approches actuelles restent fragmentées. Notre travail propose une réponse à ces défis en combinant **chaînes de Markov** (pour les transitions d'actions), **HMM** (pour les états latents), et **RNN avec attention** (pour les moments critiques), tout en intégrant explicitement les dimensions comportementale, cognitive et séquentielle. Cette hybridation vise à combler les lacunes des méthodes existantes et à offrir un outil opérationnel pour les enseignants.

2. Modélisation du problème

L'analyse des journaux d'activité des étudiants en programmation repose sur la modélisation des séquences d'interactions qu'ils effectuent dans un environnement d'apprentissage numérique. Ces interactions forment des séquences temporelles où chaque action dépend de l'état précédent du programme et des décisions de l'étudiant. Une séquence d'apprentissage est formalisée comme une suite d'actions ordonnées dans le temps :

$$X = \{x_1, x_2, \ldots, x_T\} \quad [1]$$

où :

- x_t représente l'interaction effectuée par l'étudiant à l'instant t, caractérisée par sa nature et sa dynamique temporelle.

- T est la durée totale de la séquence.

- L'ensemble des interactions possibles A est défini par leur signature comportementale :

$$\mathcal{B} = \{\textit{Édition\_fluide}, \textit{Édition\_hésitante}, \textit{Édition\_exploratoire}, \textit{Compilation\_validation}, \textit{Compilation\_répétitive}, \textit{Pause\_réflexion}, \textit{Pause\_blocage}, \textit{Consultation\_ciblée}\} \quad [2]$$

Les transitions entre interactions dépendent du contexte dynamique et de l'état cognitif de l'étudiant. Pour capturer ces dynamiques, nous proposons une approche hybride basée sur trois dimensions analytiques complémentaires : **comportementale** , **cognitive** et la **séquentielle**.

2.1. *Modélisation des transitions avec les chaînes de markov*

Les Chaînes de Markov modélisent les transitions entre états comportementaux qualifiés par leur dynamique interne :

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

où P_{ij} représente la probabilité qu'un étudiant passe de l'action s_i à l'action s_j.

La richesse des traces JSON permet de dépasser les transitions simples entre macro-actions pour modéliser des états comportementaux qualifiés. Une action "d'édition de code" se décline selon sa dynamique interne : *Édition_Fluide* , *Édition_Hésitante* et *Édition_Exploratoire*. De même, les actions de compilation se qualifient selon leur contexte temporel : *Compilation_Validation* et *Compilation_Répétitive*.

Cette qualification permet de définir un espace d'états comportementaux riches :

— La séquence *Édition_Fluide* → *Compilation_Validation* → *Édition_Fluide* indique un processus de développement maîtrisé.
— Le cycle *Édition_Hésitante* → *Compilation_Répétitive* → *Pause_Blocage* → *Consultation_Errance* signale un étudiant en difficulté progressive.
— La transition *Pause_Blocage* → *Consultation_Ciblée* → *Édition_Fluide* révèle un déblocage réussi.

2.2. *Identification des états latents avec les HMM*

Les HMM permettent d'inférer les états latents des étudiants (progression, hésitation, blocage, confusion) à partir des observations comportementales vectorisées. Un HMM est défini par :

— A : Matrice de transition entre les états cachés.
— B : Matrice d'émission reliant actions observées et états cachés.
— π : Distribution initiale des états.

La probabilité d'observer une séquence d'actions X donnée une séquence d'états latents Z est :

$$P(X \mid Z) = \prod_{t=1}^{T} B_{z_t,x_t} \times A_{z_{t-1},z_t} \quad [4]$$

Cette vectorisation permet d'inférer des états latents nuancés :

— *État de Progrès* : Associé à des observations d'*Édition_Fluide* (vitesse élevée, pauses courtes), de *Compilation_Validation*, et d'une progression mesurable de la complexité du code. La matrice d'émission B présente des probabilités élevées pour ces configurations comportementales.
— *État d'Hésitation* : Caractérisé par des observations d'*Édition_Hésitante* (vitesse faible, pauses prolongées), de *Consultation_Ciblée*, et de cycles de correction. Cet état révèle une réflexion active mais une incertitude sur la solution.
— *État de Blocage* : Révélé par des patterns d'*Édition_Exploratoire* sans progression, de *Compilation_Répétitive*, et de *Pause_Blocage* prolongées. La fragmentation croissante des sessions (interruptions fréquentes) renforce cette inférence.
— *État de Confusion* : Identifié par une errance comportementale (*Consultation_Errance*, navigation intensive sans modification), des cycles d'annulation-reprise, et une absence de stratégie cohérente.

La matrice d'émission B capture ces associations probabilistes entre configurations comportementales et états latents, permettant une inférence robuste même en présence de bruit dans les observations.

2.3. *Modélisation avec les RNN et attention*

Pour la dimension séquentielle, les RNN capturent les dépendances à long terme dans les séquences d'interactions vectorisées. Le mécanisme d'attention permet de pondérer dynamiquement les étapes clés

de la séquence, en se concentrant sur les configurations comportementales qui précèdent souvent les blocages.

Le vecteur d'entrée à chaque pas de temps t exploite naturellement la richesse multidimensionnelle des interactions :

$$\begin{aligned} input_t = [&type\_action, durée\_action, latence\_précédente, vitesse\_frappe, ratio\_suppressions, \\ &nb\_pauses, durée\_pauses, fragmentation\_session, densité\_navigation, progression\_code] \end{aligned} \quad [5]$$

Cette représentation vectorielle reflète la nature intrinsèquement multidimensionnelle des actions, où la temporalité n'est pas un ajout mais une caractéristique fondamentale de l'interaction humaine avec l'environnement de développement.

L'attention a_t pour l'étape t est calculée comme :

$$a_t = \frac{\exp(W \cdot h_t)}{\sum_{\tau=1}^{T} \exp(W \cdot h_\tau)} \quad [6]$$

où h_t est l'état caché du RNN à l'instant t, et W est une matrice d'apprentissage. Ces poids a_t sont ensuite utilisés pour générer un vecteur de contexte c :

$$c = \sum_{t=1}^{T} a_t \cdot h_t \quad [7]$$

Ce vecteur c synthétise les moments critiques de la séquence, permettant de détecter les blocages avec précision. Le mécanisme d'attention apprend à identifier des configurations spécifiques de signaux comportementaux :

— *Précurseurs de blocage* : Le modèle apprend à détecter des patterns comme la décélération progressive de la vitesse de frappe (de 4.2 à 1.8 caractères/seconde), l'augmentation du ratio de suppressions (de 12% à 35%), l'allongement des pauses (de 2.1 à 7.3 secondes), et la fragmentation croissante des sessions. Ces signaux précèdent souvent les blocages manifestes de 3 à 5 minutes.
— *Signaux de déblocage* : Le modèle identifie les reprises brutales de vitesse de frappe après une pause prolongée, la diminution des retours en arrière, et les consultations ciblées suivies d'une progression mesurable du code.
— *Patterns d'errance* : Le modèle détecte la navigation intensive sans modification, les cycles répétitifs sans progression, et les consultations multiples et dispersées qui caractérisent la désorientation cognitive.

2.4. *Trois dimensions : analyse des trajectoires d'apprentissage*

L'apprentissage de la programmation est un processus multidimensionnel où les étudiants interagissent avec leur environnement de travail en combinant des actions concrètes, des productions algorithmiques et des stratégies temporelles. Pour capturer cette complexité, notre approche repose sur une analyse intégrant trois dimensions complémentaires :

— **Comportementale** : La dynamique des interactions avec l'environnement de développement, où chaque action est caractérisée par sa signature temporelle et sa texture cognitive. Cette dimension capture non seulement ce que fait l'étudiant, mais comment il le fait.
— **Cognitive** : La qualité et la complexité du code produit (structures algorithmiques, organisation du programme, évolution de la complexité). Cette dimension révèle ce que comprend et produit l'étudiant.
— **Séquentielle** : Les enchaînements temporels des interactions et les cycles récurrents (schémas de blocage ou d'exploration, dépendances à long terme). Cette dimension capture les stratégies et les patterns d'apprentissage.

Le tableau suivant synthétise leurs caractéristiques et leur apport au modèle :

Dimension	Définition	Exemples d'interactions étudiées	Signature comportementale	Apport spécifique au modèle
Comportementale	Dynamique des interactions avec l'environnement de développement	Édition fluide/hésitante, compilation validation/répétitive, pauses réflexion/blocage, consultation ciblée/errance	Vitesse de frappe, pauses, cycles de correction, fragmentation des sessions, navigation	Analyse de la texture des interactions pour détecter des états cognitifs et des précurseurs de blocage
Cognitive	Qualité et complexité du code produit	Ajout de structures de contrôle, définition de fonctions, refactoring, gestion des erreurs	Évolution de la complexité cyclomatique, qualité des solutions, patterns de développement	Évaluation de la progression conceptuelle et du niveau de compréhension
Séquentielle	Enchaînements temporels et cycles d'apprentissage	Séquences de développement efficaces, cycles de blocage, patterns de récupération	Dépendances temporelles, cycles récurrents, stratégies à long terme	Capture des stratégies d'apprentissage et détection des patterns problématiques

TABLEAU 1. *Dimensions d'analyse des interactions et leur apport au modèle*

La combinaison de ces dimensions permet d'analyser non seulement les actions isolées, mais aussi leur contexte, leur qualité, et leur évolution temporelle. Cette approche multidimensionnelle, enrichie par la modélisation naturellement vectorielle des interactions, offre une compréhension fine des processus d'apprentissage et permet une détection précoce et précise des difficultés des étudiants.

2.5. *Dimension comportementale : modélisation de la dynamique des interactions*

La dimension comportementale constitue le coeur de notre approche, modélisant la dynamique des interactions des étudiants avec l'environnement de développement. Chaque interaction est caractérisée par sa signature temporelle et sa texture cognitive pour donner des indications sur l'état mental et les stratégies de l'apprenant.

La typologie des états comportementaux vise à décrire la dynamique d'interaction de l'étudiant avec son environnement de développement. Chaque état est caractérisé par une signature d'activité reconnaissable (rythme de frappe, pauses, cycles édition compilation, navigation), pour nous permettre de relier des faits observables à des hypothèses pédagogiques (progression, hésitation, exploration etc.). Les seuils présentés ci-dessous constituent des repères empiriques issus de nos données, mais n'ont pas vocation à être des règles absolues et peuvent être adaptés à d'autres contextes.

2.5.1. États d'édition

Les états d'édition tentent de qualifier la manière de produire le code. Ils renseignent à la fois sur la fluidité de l'écriture, la stabilité de la stratégie en cours et le degré d'incertitude rencontré. Nous distinguons trois profils dominants, que nous détectons à partir de métriques simples (vitesse instantanée, coefficient de variation, pauses, ratio de suppressions et retours en arrière) :

— *Édition_Fluide* : Caractérisée par une frappe régulière (vitesse > 3 caractères/seconde, coefficient de variation < 0.3), des pauses courtes et ciblées (< 3 secondes), et un faible ratio de suppressions ($< 10\%$). Cette signature révèle un état de flow cognitif où l'étudiant traduit fluidement sa pensée en code.
— *Édition_Hésitante* : Marquée par une frappe irrégulière (vitesse < 2 caractères/seconde, coefficient de variation > 0.6), des pauses fréquentes et prolongées (> 5 secondes), et un ratio de suppressions élevé ($> 30\%$). Cette dynamique signale une incertitude conceptuelle ou technique.
— *Édition_Exploratoire* : Caractérisée par de nombreux retours en arrière (> 5 par minute), des modifications courtes suivies d'annulations (< 10 caractères avant suppression), et une navigation intensive dans le code (> 10 changements de position par minute). Cette texture révèle une recherche de solution ou une désorientation.

Dans la suite, ces états d'édition servent d'indices précoces de progression (fluidité) ou de risque de blocage et alimentent la dimension comportementale de notre modèle.

2.5.2. États de compilation

Les états de compilation renseignent sur la relation entre écriture du code et la vérification des résultats. Ils visent à distinguer une démarche de validation méthodique (compilation régulière du code) au cours d'un cycle d'essais-erreurs. Deux états ont été mis en évidence dans nos jeux de traces :

— *Compilation_Validation* : Précédée d'une période d'édition substantielle (> 50 caractères modifiés) et suivie d'une analyse des résultats (latence > 10 secondes avant l'action suivante). Révèle une approche méthodique de validation.
— *Compilation_Répétitive* : Cycles rapides de compilation sans modification significative du code (< 10 caractères modifiés, latence < 5 secondes), révélant une stratégie « trial-and-error » ou une incompréhension des messages d'erreur.

Ces états complètent la lecture des phases d'édition : une *Édition_Hésitante* combinée à une *Compilation_Répétitive* constitue, par exemple, un pattern typique de difficulté persistante.

2.5.3. États de pause

Les pauses constituent un indicateur important de la dynamique cognitive. Elles peuvent révéler non seulement la gestion de l'effort par l'étudiant, mais aussi son état d'engagement face à la tâche. Une courte interruption peut correspondre à une phase de planification, tandis qu'une pause prolongée peut signaler une difficulté persistante ou un découragement. Nous distinguons ici deux états principaux .

— *Pause_Réflexion* : Pauses de durée modérée (5–15 secondes) suivies d'une reprise d'activité productive (progression mesurable du code). Indique une planification cognitive active.
— *Pause_Blocage* : Pauses prolongées (> 15 secondes) associées à une fragmentation de session (interruptions fréquentes) et suivies d'activité non productive. Signale une surcharge cognitive ou un découragement.

L'analyse des pauses fournit un complément essentiel aux métriques d'édition et de compilation, en tentant de révéler ce qui se passe dans les moments de silence entre deux actions.

2.5.4. États de consultation

Les états de consultation rendent compte de la manière dont l'étudiant mobilise des ressources externes (recherche web, accès au support de cours...) pour surmonter ses difficultés. Ils tentent de différencier une recherche d'information efficace utile d'une errance potentiellement contre-productive. Deux formes de consultation sont distinguées :

— *Consultation_Ciblée* : Accès bref à des ressources spécifiques (< 2 minutes) suivi d'une application directe dans le code. Révèle une stratégie de recherche d'information efficace.
— *Consultation_Errance* : Navigation dispersée dans multiples ressources (> 3 sources différentes) sans application immédiate. Indique une désorientation ou une surcharge informationnelle.

Les états de consultation sont ainsi des indices supplémentaires sur la capacité des étudiants à s'autoréguler et à mobiliser des aides pertinentes au bon moment.

2.5.5. États de débogage

Le débogage constitue une étape fréquente de l'apprentissage de la programmation. La manière dont l'étudiant aborde cette tâche reflète son niveau de maîtrise des outils disponibles et sa stratégie de résolution d'erreurs. Nous distinguons deux comportements typiques :

— *Débogage_Systématique* : Utilisation méthodique des outils de débogage (points d'arrêt, inspection de variables) avec progression logique dans la localisation d'erreurs.
— *Débogage_Erratique* : Navigation aléatoire dans le code, modifications multiples sans hypothèse claire, absence d'utilisation des outils de débogage.

Ces deux états permettent de distinguer un apprentissage en cours de structuration d'une approche plus intuitive mais fragile, qui tend à générer des cycles d'essais-erreurs peu efficaces.

2.5.6. Extraction des signatures comportementales

L'extraction des signatures comportementales consiste à transformer les traces brutes en indicateurs quantitatifs exploitables. Nous analysons les fichiers de traces JSON à l'aide de fenêtres temporelles glissantes de 30 secondes. Pour chaque fenêtre, nous calculons un ensemble de métriques qui capturent différents aspects de l'activité : la vitesse et la régularité de la frappe, le rythme temporel des actions, ou encore la manière dont l'étudiant navigue dans son code. Ces métriques servent ensuite de base pour la classification des états comportementaux.

2.5.7. Métriques de frappe

Les métriques de frappe décrivent le rapport direct de l'étudiant à son clavier. Elles traduisent la fluidité et la régularité de l'écriture, mais aussi les hésitations et les corrections fréquentes.

— *Vitesse instantanée* : $V(t) = \dfrac{nb\_caractères}{durée\_fenêtre}$. Une vitesse élevée est généralement associée à une production fluide de code, tandis qu'une vitesse faible peut signaler une hésitation (mais aussi parfois un apprentissage en cours de la frappe au clavier).

— *Variance de vitesse* : $\mathrm{Var}(V) = \sigma^2(vitesses\_instantanées)$. Une faible variance traduit une frappe régulière, alors qu'une variance élevée révèle des alternances entre phases de fluidité et d'hésitation.

— *Ratio de suppressions* : $R_{supp} = \dfrac{nb\_backspace}{nb\_frappes\_totales}$. Cet indicateur permet d'identifier la tendance à corriger fréquemment son code, ce qui peut être signe soit d'une relecture attentive, soit d'une difficulté à stabiliser une solution.

— *Longueur moyenne des rafales* : $L_{rafale} = \mathrm{moyenne}(séquences\_interrompues)$. Les rafales longues indiquent une production soutenue et cohérente, tandis que les rafales très courtes traduisent une écriture fragmentée.

2.5.8. Métriques temporelles

Les métriques temporelles qualifient le rythme global d'interaction avec l'environnement numérique. Elles renseignent sur la continuité de l'activité et estiment la stabilité de l'attention de l'étudiant.

— *Durée des pauses* : $P(t) = \{p_i \mid p_i > seuil\_pause\}$. De courtes pauses sont souvent le signe d'une réflexion ciblée, tandis que les pauses prolongées peuvent refléter un risque de blocage.

— *Latence inter-actions* : $L(t) = temps\_entre\_actions\_significatives$. Cette métrique mesure la rapidité avec laquelle l'étudiant enchaîne ses actions retranscrivant plus ou moins son degré de fluidité ou d'hésitation.

— *Fragmentation de session* : $F(t) = \dfrac{nb\_interruptions}{durée\_totale}$. Un score élevé traduit une activité découpée et instable, souvent associée à des comportements exploratoires ou erratiques.

— *Régularité temporelle* : $R(t) = \dfrac{1}{coefficient\_variation(intervalles)}$.Une régularité forte reflète une approche méthodique, tandis qu'une forte variabilité est souvent le signe d'une difficulté à appliquer maintenir une stratégie de travail stable.

2.5.9. Métriques de navigation

Les métriques de navigation décrivent la manière dont l'étudiant se déplace dans son code. Elles donnent des indices sur son organisation cognitive et sur sa capacité à garder une vision cohérente de la tâche en cours.

— *Densité de navigation* : $D_{nav} = \dfrac{nb\_changements\_position}{durée}$. Une densité modérée correspond à une navigation structurée, tandis qu'une densité élevée peut refléter de l'errance.
— *Amplitude de déplacement* : $A_{depl} = \text{moyenne}(\text{distances\_parcourues})$. Cet indicateur capture l'ampleur des déplacements dans le code : de faibles amplitudes reflètent une focalisation locale, tandis que de grandes amplitudes indiquent une recherche plus globale.
— *Retours en arrière* : $R_{back} = \dfrac{nb\_retours}{nb\_déplacements\_totaux}$. Une proportion élevée de retours en arrière peut traduire des hésitations ou une exploration infructueuse.

En combinant ces métriques de frappe, temporelles et de navigation, nous obtenons une image plus ou moins précise de la dynamique comportementale de l'étudiant sur des intervalles courts, ce qui nourrit la détection de ses états cognitifs.

2.5.10. Métriques contextuelles

Les métriques contextuelles apportent une information complémentaire en reliant l'activité de l'étudiant à son environnement logiciel et aux outils mobilisés. Elles rendent compte de la complexité du code produit, de l'usage du débogueur et du recours à des ressources externes. Ces indicateurs permettent de mieux comprendre les stratégies adoptées face aux difficultés.

— *Progression du code* : $P_{code} = \Delta(complexité\_cyclomatique)$. Cet indicateur mesure l'évolution de la complexité structurelle du code. Une augmentation progressive correspond à un développement méthodique tandis qu'une croissance brutale peut signaler l'ajout précipité de fonctionnalités non nécessaires à ce stade de développement.
— *Utilisation d'outils* : $U_{outils} = fréquence\_accès\_débogueur$. Cet indicateur reflète la capacité de l'étudiant à mobiliser des outils adaptés pour diagnostiquer ses erreurs. Un usage systématique témoigne d'une stratégie structurée, alors qu'une absence d'utilisation peut indiquer une méconnaissance des outils disponibles.
— *Consultation de ressources* : $C_{ress} = \dfrac{nb\_accès\_externes}{durée}$. Cet indicateur renseigne sur le recours à des ressources externes (forums, docu mentation en ligne, tutoriels). Une consultation ponctuelle et ciblée est souvent un bon signe, tandis qu'une fréquence excessive traduit un manque d'autonomie ou une désorientation.

La classification automatique des états comportementaux repose sur l'ensemble de ces métriques, en s'appuyant sur des seuils empiriques déterminés par analyse des données d'entraînement.

2.6. *Dimension cognitive : analyse de la production de code*

La dimension cognitive évalue la qualité et la complexité du code produit, révélant le niveau de compréhension conceptuelle et les stratégies de développement de l'étudiant.

2.6.1. Analyse de la qualité du code

La qualité du code constitue un indicateur central de la progression de l'étudiant. Au-delà de la complexité brute, il s'agit d'évaluer la manière dont le code est structuré, la cohérence de son architecture et la façon dont l'étudiant gère les erreurs rencontrées. Trois dimensions complémentaires sont ici retenues : la cohésion et le couplage, les patterns de développement, et la gestion des erreurs.

2.6.2. Cohésion et couplage

La cohésion et le couplage permettent d'évaluer l'architecture interne du code produit par l'étudiant. Elles renseignent sur le degré d'or ganisation de sa pensée algorithmique et sur sa capacité à construire un code robuste.

— *Cohésion fonctionnelle* : degré de relation entre les éléments d'un module (classe / package ...). Une forte cohésion traduit une conception claire et ciblée des fonctions.
— *Couplage entre modules* : niveau d'interdépendance entre différentes parties du code. Un faible couplage est généralement souhaitable car il facilite la réutilisation et la maintenance.
— *Évolution temporelle* : observation des variations de cohésion et de couplage au fil du développement. Une amélioration (cohésion plus forte, couplage plus faible) reflète une structuration progressive du code, tandis qu'une dégradation indique une organisation de plus en plus chaotique.

2.6.3. Patterns de développement

L'analyse des patterns de développement permet de caractériser la stratégie globale adoptée par l'étudiant. Chaque pattern renvoie à une approche particulière de la résolution de problèmes et du cycle de production.

— *Développement incrémental* : ajouts progressifs de fonctionnalités accompagnés de tests intermédiaires. Ce pattern reflète une démarche prudente et contrôlée.
— *Développement Big-Bang* : écriture massive de code suivie de phases de débogage intensives. Cette approche est risquée, car elle retarde la détection des erreurs.
— *Développement itératif* : alternance de phases de développement, test et refactoring. Ce pattern traduit une stratégie d'amélioration continue, proche des pratiques agiles.

2.6.4. Gestion des erreurs

La manière dont l'étudiant gère les erreurs constitue un révélateur important de ses compétences. Un étudiant expérimenté tend à identifier rapidement l'origine des problèmes et à appliquer des stratégies de correction efficaces, tandis qu'un étudiant en difficulté accumule des erreurs récurrentes et peine à les résoudre.

— *Fréquence des erreurs de syntaxe* : reflète le degré de maîtrise de la grammaire du langage.
— *Types d'erreurs récurrentes* : erreurs logiques, syntaxiques ou sémantiques, permettant de distinguer les difficultés conceptuelles des maladresses techniques.
— *Temps de résolution* : durée nécessaire pour corriger une erreur, indicateur de l'efficacité des stratégies de débogage.
— *Stratégies de correction* : systématiques (tests ciblés, corrections méthodiques) ou aléatoires (modifications successives sans hypothèse claire).

2.6.5. Corrélation comportement-production

L'analyse croisée des dimensions comportementale et cognitive permet de mettre en évidence des corrélations entre les signatures d'activité observées et la qualité du code produit. Ces corrélations révèlent des patterns caractéristiques de réussite ou de blocage.

- *Corrélations Positives*

— *Édition_Fluide* $\leftrightarrow$ augmentation régulière de la complexité ;
— *Pause_Réflexion* $\leftrightarrow$ amélioration de la qualité du code ;
— *Consultation_Ciblée* $\leftrightarrow$ résolution efficace des erreurs ;

- *Corrélations Négatives*

— *Édition_Hésitante* $\leftrightarrow$ stagnation de la complexité ;
— *Compilation_Répétitive* $\leftrightarrow$ accumulation d'erreurs ;
— *Consultation_Errance* $\leftrightarrow$ dégradation de la cohésion ;

Ces observateurs indiquent que l'activité captable dans l'éditeur n'est pas neutre : elle porte la trace de processus cognitifs et stratégiques qui peuvent traduire l'aisance ou la difficulté d'un étudiant sur l'activité menée.

2.7. *Dimension séquentielle : modélisation des dynamiques temporelles*

La dimension séquentielle capture les enchaînements temporels des interactions et les cycles récurrents, révélant les stratégies d'apprentissage et les patterns de résolution de problèmes.

2.7.1. Analyse des séquences comportementales

L'analyse des séquences comportementales vise à mettre en évidence les enchaînements typiques d'actions réalisés par les étudiants. Ces séquences peuvent révéler non seulement la dynamique locale de leur activité, mais aussi la structure globale de leur démarche d'apprentissage. Nous distinguons deux grandes familles : les séquences dites efficaces, qui accompagnent une progression mesurable, et les séquences problématiques, qui reflètent au contraire des difficultés persistantes.

2.7.2. Séquences efficaces

Patterns temporels associés à une progression réussie :

— *Édition_Fluide* $\rightarrow$ *Compilation_Validation* $\rightarrow$ *Analyse_Résultats* $\rightarrow$ *Édition_Fluide* : une boucle vertueuse où l'étudiant écrit son code avec fluidité, valide ses hypothèses par compilation, analyse les résultats, puis reprend une écriture structurée.
— *Pause_Réflexion* $\rightarrow$ *Consultation_Ciblée* $\rightarrow$ *Édition_Fluide* $\rightarrow$ *Progression_Code* : séquence qui illustre la capacité de l'étudiant à interrompre brièvement son activité pour chercher une information pertinente, l'appliquer immédiatement dans le code, et progresser.
— *Débogage_Systématique* $\rightarrow$ *Correction_Ciblée* $\rightarrow$ *Validation* $\rightarrow$ *Continuation* : une stratégie de débogage raisonnée, où l'étudiant identifie l'origine d'un problème, applique une correction précise, puis valide le résultat avant de pour suivre.

2.7.3. Séquences problématiques

À l'inverse, les séquences problématiques révèlent des schémas d'activité dans lesquels l'étudiant s'enferme dans un cycle improductif. Elles sont caractérisées par des hésitations, des répétitions ou une navigation dispersée. Ces patterns constituent des signaux forts de blocage, que notre modèle cherche à détecter. Parmi les séquences les plus représentatives, on trouve :

Patterns révélant des difficultés :

— *Édition_Hésitante* → *Compilation_Répétitive* → *Pause_Blocage* → *Consultation_Errance* : un enchaînement typique d'un étudiant en difficulté, alternant écriture irrégu lière, tests compulsifs, pauses prolongées et recherches infructueuses.
— *Édition_Exploratoire* → *Édition_Exploratoire* → *Pause_Blocage* (cycle d'errance) : cycle d'errance où l'étudiant multiplie les modifications sans orientation claire, suivies de pauses improductives.
— *Compilation_Répétitive* → *Compilation_Répétitive* → *Abandon* (cycle d'échec) : cycled'échec dans lequel la répétition de compilations infructueuses conduit à l'arrêt prématuré de l'activité.

La mise en évidence de ces séquences nous permet de caractériser la frontière entre une activité productive et une activité bloquée, ouvrant la voie à une détection automatique des moments où une intervention pédagogique de l'enseignant peut s'avérer nécessaire.

2.7.4. Détection des cycles récurrents

Au-delà des séquences ponctuelles, il est essentiel de repérer les cycles récurrents dans lesquels un étudiant peut s'enfermer. Ces cycles correspondent à la répétition régulière d'un même enchaînement d'actions, souvent révélateur d'une stratégie d'essais infructueux ou d'une stagnation. La détection de ces motifs put être un levier important pour identifier non seulement un blocage ponctuel, mais également des difficultés structurelles qui s'installent dans le temps.

2.7.5. Algorithme de détection de cycles

Pour repérer ces répétitions, nous avons implémenté un algorithme d'analyse de séquences inspiré des techniques de fouille de motifs temporels. L'idée consiste à parcourir la séquence d'états comportementaux et à rechercher des sous-séquences qui se répètent au-delà d'une certaine taille minimale. Le pseudo-code ci-dessous illustre cette procédure de détection :

```
fonction DétecterCycles(séquence_états, fenêtre_minimale=3):
    cycles_détectés = []
    pour i de 0 à longueur(séquence) - fenêtre_minimale:
        pour longueur_cycle de fenêtre_minimale à longueur_max:
            sous_séquence = séquence[i:i+longueur_cycle]
            si sous_séquence se répète dans séquence[i+longueur_cycle:]:
                cycles_détectés.ajouter(sous_séquence)
    retourner cycles_détectés
```

Cet algorithme permet d'extraire automatiquement les motifs cycliques dans les traces d'activité. Un cycle de type *Erreur* → *Tentative* → *Erreur*, répété de manière récurrente, peut ainsi être identifié comme un signe clair de blocage conceptuel.

2.7.6. Classification des cycles

Une fois détectés, les cycles sont classés en trois grandes catégories, selon leur impact sur la progression de l'étudiant :

— *Cycles Productifs* : Mènent à une progression mesurable (ex. *Édition* → *Test* → *Correction*)
— *Cycles Neutres* : Ils n'ont pas d'influence notable sur la progression, mais traduisent des micro-boucles d'interaction. Un exemple typique est Navigation → Retour, qui peut refléter une simple vérification (ex. *Navigation* → *Retour*).
— *Cycles Destructifs* : Ils entravent directement la progression et témoignent d'un blocage persistant, il enferme l'étudiant dans une dynamique improductive (ex. *Erreur* → *Tentative* → *Erreur*).

Cette classification fournit un cadre interprétatif qui permet de distinguer les répétitions traduisant une stratégie d'apprentissage productive de celles qui révèlent une impasse nécessitant une intervention pédagogique de l'enseignant.

Au-delà des séquences immédiates et des cycles locaux, l'apprentissage en program mation se caractérise aussi par des dépendances à long terme. Autrement dit, les actions réalisées par un étudiant à un instant donné peuvent influencer ses comportements ultérieurs, parfois plusieurs dizaines de minutes plus tard. Prendre en compte ces relations de dépendance est essentiel pour comprendre comment un étudiant consolide ses acquis ou, au contraire, s'enferme dans des stratégies inefficaces.

2.7.7. Mémoire comportementale

La mémoire comportementale désigne l'influence des actions passées sur les actions futures. Elle reflète la manière dont l'historique d'interactions façonne la trajectoire d'apprentissage de l'étudiant. Nous observons trois phénomènes principaux :

— *Effet des erreurs sur la confiance* : les erreurs répétées peuvent réduire la vitesse de frappe et générer une hésitation croissante, traduisant une perte de confiance dans la démarche adoptée.
— *Impact des succès sur l'audace* : les réussites renforcent la confiance et encouragent l'exploration de solutions plus ambitieuses, par exemple l'utilisation de structures de contrôle plus complexes.
— *Persistance des stratégies inefficaces* : certains étudiants tendent à reproduire des cycles récurrents improductifs (ex. *Erreur* → *Tentative* → *Erreur*), même après plusieurs itérations infructueuses. Ces comportements révèlent souvent une inertie cognitive qui peut freiner la progression.

2.7.8. Évolution des patterns

Changements dans les séquences comportementales au cours de l'apprentissage :

— *Réduction progressive des cycles destructifs* : avec l'expérience, les étudiants apprennent à éviter les boucles improductives et parviennent à sortir plus rapidement des situations d'échec.

— *Augmentation de la fréquence des séquences efficaces* : les enchaînements fluides (édition → validation → correction) deviennent de plus en plus fréquents à mesure que l'étudiant maîtrise son environnement.
— *Développement de stratégies métacognitives* : l'apparition de pauses réflexives mieux exploitées traduit une capacité croissante à prendre du recul, planifier et ajuster sa démarche de résolution de problème.

Ces évolutions témoignent d'un apprentissage non seulement technique mais aussi stratégique, où l'étudiant apprend à mieux réguler son activité et à mobiliser des stratégies de plus en plus adaptées et efficaces.

2.7.9. Algorithme d'hybridation

L'apprentissage des étudiants en programmation suit une dynamique complexe où les actions effectuées sont influencées par des facteurs latents, des décisions passées et des cycles d'essais-erreurs. L'algorithme prend en entrée une séquence d'actions *Erreur* → *Tentative* → *Erreur* et enrichit chaque action par trois dimensions analytiques :

— *Comportementale* : Actions isolées (ex. : nombre de retours en arrière).
— *Cognitive* : Impact sur la structure du code (ex. : ajout de boucles).
— *Séquentielle* : Contexte temporel et cycles récurrents.

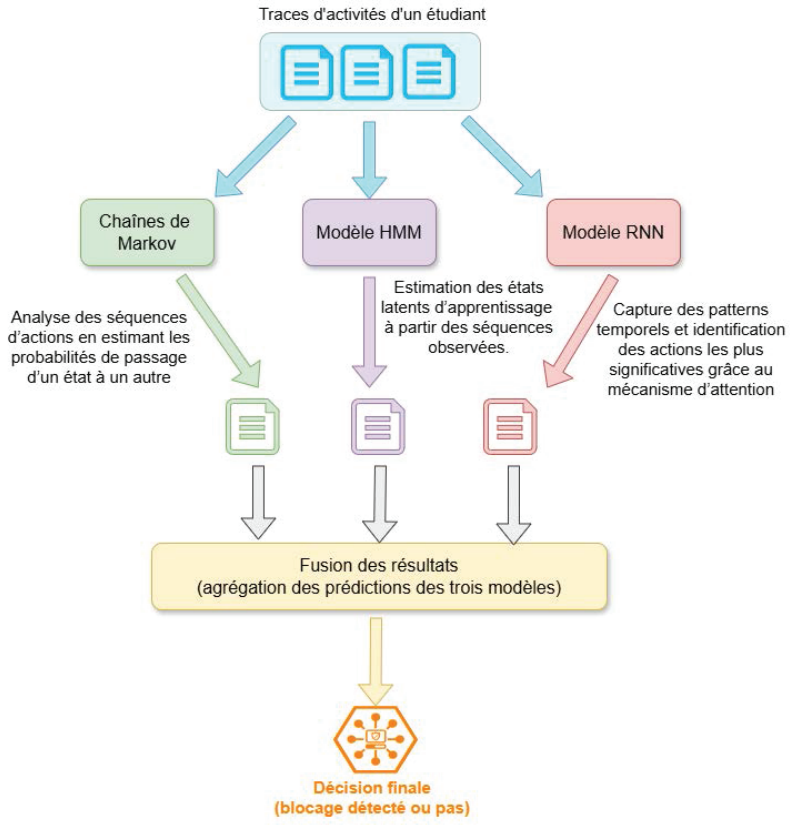

FIGURE 1. *La Figure illustre le fonctionnement de notre algorithme hybride, montrant comment les journaux d'activité sont analysés en parallèle par trois modèles distincts avant d'être fusionnés pour détecter les blocages.*

Les trois modèles opèrent en synergie. Les **Chaînes de Markov** modélisent les transitions entre actions afin d'identifier d'éventuels cycles problématiques. Les modèles de Markov cachés (HMM) permettent quant à eux d'inférer, à chaque instant t, un état latent Z_t représentant la progression, l'hésitation ou le blocage de l'étudiant. Enfin, les **réseaux de neurones récurrents (RNN)** avec mécanisme d'attention attribuent à chaque action un poids α_t, afin de mettre en évidence les actions jugées critiques dans l'évolution de la séquence (comme détaillé dans l'Équation 5).

L'algorithme génère trois sorties principales. Tout d'abord, un **score de difficulté** $\hat{y} \in [0, 1]$, où une valeur proche de 1 indique un blocage probable (par exemple, un cycle de répétition d'erreurs), tandis qu'un score proche de 0 reflète une progression fluide. Ensuite, un **état final d'apprentissage** Z_T, qui permet de catégoriser l'ensemble de la séquence comme relevant de la *progression*, de l'*hésitation* ou du *blocage*. Enfin, une **explication via le mécanisme d'attention** permet d'identifier les actions ayant le plus contribué à la prédiction : par exemple, une erreur répétée fortement pondérée avec $\alpha_t = 0.8$ indique une influence significative dans la classification.

Les seuils de détection sont optimisés par validation croisée pour maximiser le F1-score tout en minimisant les faux positifs :

Seuil d'Alerte Principal : Optimisé pour équilibrer précision et rappel Seuils Secondaires : Adaptés selon le contexte (début vs. fin de session, complexité de la tâche) Seuils Adaptatifs : Ajustés selon le profil individuel de l'étudiant (historique de performance)

Cette architecture d'hybridation permet de combiner les forces de chaque modèle tout en compensant leurs faiblesses respectives, produisant un système de détection robuste, précoce et interprétable pour l'assistance pédagogique en temps réel.

L'implémentation des trois composants du modèle hybride repose sur des choix de paramétrage justifiés par des considérations théoriques et empiriques.

Chaînes de Markov.

Nous utilisons des chaînes de Markov d'ordre 1, reposant sur l'hypothèse que la probabilité d'une action dépend uniquement de l'action précédente. La matrice de transition $\mathbf{A} \in \mathbb{R}^{5\times5}$ est estimée par maximum de vraisemblance sur l'ensemble des séquences d'entraînement :

$$a_{ij} = \frac{N_{ij}}{\sum_{k=1}^{5} N_{ik}} \quad [8]$$

où N_{ij} représente le nombre de transitions observées de l'état i vers l'état j. Le nombre d'états observables correspond aux 5 catégories d'actions identifiées : *Édition*, *Compilation*, *Pause*, *Consultation* et *Débogage*. Un lissage de Laplace ($\alpha = 1$) est appliqué pour éviter les probabilités nulles.

Modèles de Markov Cachés (HMM).

Le modèle HMM comporte 4 états latents : *Progrès*, *Hésitation*, *Blocage* et *Confusion*. Ce nombre d'états a été déterminé par une analyse exploratoire comparant des modèles à 3, 4, 5 et 6 états, et validé par le critère d'information bayésien (BIC), qui a atteint son minimum pour $K = 4$ états. Les paramètres

Algorithm 1 FusionAdaptative : Intégration pondérée de Markov, HMM et RNN pour la détection de blocages

Require: Séquence d'interactions $x_{1:T} = [x_1, x_2, ..., x_T]$
Require: Modèle de Markov M, Modèle HMM H, Modèle RNN R
Require: Fenêtre temporelle w
Ensure: Probabilité de blocage $P(\text{blocage})$, état cognitif latent $\hat{z}$, moment critique t^*, type de blocage détecté

Phase 1 : Extraction des Signatures Comportementales
for chaque interaction $x_i \in x_{1:T}$ **do**
    signature$[i] \leftarrow$ ExtraireSignature(x_i, w)
    état_comportemental$[i] \leftarrow$ ClassifierÉtat(signature$[i]$)
end for

Phase 2 : Prédictions Individuelles des Modèles
// Chaînes de Markov : Probabilités de transition
$P_{\text{markov}} \leftarrow []$
for $i = 2$ to T **do**
    $P_{\text{markov}}[i] \leftarrow M$.probabilité_transition(état_comportemental$[i-1]$, état_comportemental$[i]$)
end for
// HMM : Inférence d'états latents et probabilité d'émission
états_latents $\leftarrow H$.viterbi(signatures_vectorisées)
$P_{\text{hmm}} \leftarrow H$.probabilité_émission(états_latents, signatures)
// RNN : Détection des moments critiques et poids d'attention
séquence_vect $\leftarrow$ VectoriserInteractions$(x_{1:T})$
$P_{\text{rnn}}, \alpha \leftarrow R$.prédire(séquence_vect)

Phase 3 : Fusion Adaptative Pondérée
confiance_markov $\leftarrow$ CalculerConfiance(P_{markov}, variance_transitions)
confiance_hmm $\leftarrow$ CalculerConfiance(P_{hmm}, cohérence_états)
confiance_rnn $\leftarrow$ CalculerConfiance(P_{rnn}, stabilité_attention)
// Normalisation des poids
poids_total $\leftarrow$ confiance_markov + confiance_hmm + confiance_rnn
$w_M \leftarrow$ confiance_markov/poids_total
$w_H \leftarrow$ confiance_hmm/poids_total
$w_R \leftarrow$ confiance_rnn/poids_total
// Fusion pondérée des prédictions
$P(\text{blocage}) \leftarrow w_M \cdot \text{MoyennePondérée}(P_{\text{markov}}) + w_H \cdot \text{ProbabilitéÉtatBlocage}(\text{états\_latents}) + w_R \cdot P_{\text{rnn}}$

Phase 4 : Détection des Moments Critiques
if $P(\text{blocage}) >$ seuil_alerte **then**
    $t^* \leftarrow \text{ArgMax}(\alpha)$
    $\hat{z} \leftarrow$ états_latents$[t^*]$
    type_blocage $\leftarrow$ CaractériserBlocage(état_comportemental$[t^*]$, $\hat{z}$, historique_récent)
end if

return $P(\text{blocage}), \hat{z}, t^*$, type_blocage

du modèle (matrice de transition entre états latents $\mathbf{A}$, matrice d'émission $\mathbf{B}$, et distribution initiale $\boldsymbol{\pi}$) sont estimés par l'algorithme de Baum-Welch, avec les hyperparamètres suivants :

— Nombre maximal d'itérations : 100
— Seuil de convergence : 10^{-4} (variation de la log-vraisemblance)
— Initialisation : aléatoire avec 10 redémarrages, conservation du meilleur modèle

L'inférence des états latents pour une nouvelle séquence est réalisée par l'algorithme de Viterbi.

Réseaux de Neurones Récurrents (RNN) avec attention.

L'architecture du RNN est conçue pour capturer les dépendances temporelles à long terme tout en identifiant les moments critiques de la séquence. Elle comprend les couches suivantes :

— Couche d'embedding : transformation des actions discrètes en vecteurs denses de dimension 64
— Couches récurrentes : deux couches GRU bidirectionnelles de 128 unités chacune, avec dropout de 0.3 entre les couches
— Mécanisme d'attention : attention de type Bahdanau (additive), permettant au modèle de pondérer différemment chaque pas de temps de la séquence
— Couche de sortie : couche dense avec activation softmax pour la classification multi-classe (4 états)

L'entraînement est réalisé avec les hyperparamètres suivants :

— Optimiseur : Adam (learning rate = 10^{-3}, $\beta_1 = 0.9$, $\beta_2 = 0.999$)
— Fonction de perte : entropie croisée catégorielle
— Taille de batch : 32 séquences
— Régularisation : early stopping avec patience de 10 époques sur la perte de validation
— Augmentation de données : décalage temporel aléatoire (± 2 pas de temps)

Le modèle a été implémenté en Python avec la bibliothèque TensorFlow/Keras (version 2.12). L'entraînement a été réalisé sur un 1 lth Gen Intel(R) Core(TM) 15-1135G7 @ 2.40GHz 32GB Physical Memory, avec un temps d'entraînement moyen de 15 minutes pour 50 époques.

Contexte : Étudiant E42, TP sur le tri à bulles en langage C, durée de la séquence analysée : 8 minutes.

Exemple de trace d'un étudiant en situation de blocage

Contexte. Nous présentons ici l'analyse de la trace de l'étudiant E42, enregistrée lors d'un TP portant sur l'implémentation d'un tri à bulles en langage C. La séquence analysée couvre une durée de 8 minutes et illustre comment les trois dimensions de notre modèle convergent pour caractériser une situation de blocage.

Séquence d'actions observées. L'étudiant commence par écrire la fonction `tri_bulles()` (t=0 :00), puis marque une première pause de 12 secondes (t=0 :45). Il reprend l'édition en supprimant 3 lignes et en réécrivant la boucle `for` (t=0 :57). Une première compilation (t=1 :32) révèle une erreur de syntaxe (point-virgule manquant), qu'il corrige rapidement (t=1 :38). La compilation suivante (t=1 :52) produit une erreur logique : la boucle devient infinie en raison d'une condition `i < n` incorrecte. S'ensuit une pause de 28 secondes (t=2 :05), après laquelle l'étudiant supprime complètement les deux boucles imbriquées (t=2 :33).

L'étudiant entre alors dans une phase de consultation : il effectue une recherche web sur "bubble sort C example" (t=2 :58), consulte une autre page sur "tri à bulles algorithme" (t=3 :42), puis retourne au support de cours (t=4 :15). Il copie-colle un extrait de code depuis le support (t=5 :02), mais la compilation échoue car la variable `temp` n'est pas déclarée (t=5 :18). Une pause prolongée de 45 secondes suit (t=5 :25). L'étudiant ajoute la déclaration `int temp;` (t=6 :10), mais la compilation suivante produit une erreur d'indice hors limites sur `tab[j+1]` (t=6 :22). Après une nouvelle pause de 52 secondes (t=6 :28), il supprime son code et tente une nouvelle approche (t=7 :20). La compilation produit un segmentation fault à l'exécution (t=7 :48), et l'étudiant reste inactif plus de 60 secondes, marquant la fin de la séquence analysée (t=8 :00).

Analyse de la dimension comportementale. Les métriques comportementales révèlent un profil d'édition hésitante. La vitesse de frappe moyenne est de 1.2 caractères par seconde, bien en dessous du seuil de fluidité fixé à 3 caractères par seconde. Le coefficient de variation de la vitesse atteint 0.78, traduisant une forte irrégularité dans le rythme d'écriture. Le ratio de suppressions s'élève à 42%, largement supérieur au seuil de 30% caractéristique d'une incertitude. Enfin, la durée moyenne des pauses est de 34 secondes, dépassant le seuil de 15 secondes associé aux situations de blocage. L'ensemble de ces indicateurs conduit à la détection successive des états *Édition_Hésitante* puis *Pause_Blocage*.

Analyse de la dimension cognitive. Sur le plan cognitif, la complexité cyclomatique du code produit stagne, oscillant entre les valeurs 2, 1 et 2 au fil des tentatives, sans progression significative. L'étudiant corrige rapidement les deux erreurs de syntaxe rencontrées, mais les trois erreurs logiques restent non résolues à la fin de la séquence. Le pattern de développement correspond à une *Compilation_Répétitive* : l'étudiant enchaîne les compilations sans modification substantielle du code entre chaque tentative. La corrélation entre l'état *Édition_Hésitante* et la stagnation de la complexité confirme la cohérence entre les dimensions comportementale et cognitive.

Analyse de la dimension séquentielle. L'analyse des enchaînements temporels met en évidence un cycle récurrent : Édition $\rightarrow$ Compilation $\rightarrow$ Erreur $\rightarrow$ Pause $\rightarrow$ Édition. Ce cycle se répète quatre fois au cours de la séquence, sans progression mesurable vers la résolution du problème. Il est classifié comme *Cycle_Blocage*, caractéristique d'une stratégie d'essai-erreur inefficace. Par ailleurs, la phase de consultation est qualifiée de *Consultation_Errance* : l'étudiant consulte trois sources différentes en moins de deux minutes, sans parvenir à appliquer directement les informations trouvées.

Sortie du modèle hybride. Les trois composants du modèle convergent vers un diagnostic de blocage. La chaîne de Markov estime à 0.72 la probabilité de transition vers un état d'erreur à partir de la configuration actuelle. Le modèle HMM infère l'état latent *Blocage* avec une probabilité de 0.84. Le mécanisme d'attention du RNN identifie trois moments critiques dans la séquence : t=2 :05 (première pause prolongée après erreur logique), t=5 :25 (pause après échec du copier-coller), et t=6 :28 (pause précédant l'abandon de la stratégie en cours). Le score de blocage agrégé, combinant les sorties des trois modèles, atteint 0.81 sur 1.00, déclenchant une recommandation d'intervention prioritaire.

Interprétation pédagogique. Cet étudiant présente un pattern typique de blocage conceptuel sur l'algorithme du tri à bulles. Les indicateurs convergent : frappe hésitante avec nombreuses suppressions, pauses prolongées après chaque erreur, consultations dispersées sans application, et cycle répétitif compilation-erreur sans progression. La difficulté semble porter sur la gestion des indices dans les boucles imbriquées, notamment la condition `j < n-i-1` et la logique de permutation des éléments

adjacents. Le modèle recommande une intervention de l'enseignant, idéalement sous forme d'un indice ciblé sur la logique des bornes de boucles plutôt qu'une correction directe du code.

3. Évaluation et protocole expérimental

L'évaluation de notre modèle hybride repose sur une approche expérimentale rigoureuse, conçue pour analyser sa précision, sa capacité à détecter les blocages, et son interprétabilité via l'analyse des cycles d'apprentissage. Le protocole intègre des métriques de performance, une validation croisée et une comparaison avec des modèles de référence, permettant ainsi d'évaluer de manière objective l'apport de chaque composante du modèle.

3.1. *Présentation des données*

Les données exploitées par notre modèle hybride correspondent à des traces laissées par les étudiants débutants en informatique que leur enseignant a placés en situation de résolution de problèmes de programmation. La collecte de ces traces a été effectuée via un outillage logiciel (comprenant une partie client et une partie serveur) développé au sein du laboratoire LIUPPA et déposée à l'Agence de Protection du Logiciel (Nodenot et al., 2023).

Côté client, pour chaque poste client (1 seul étudiant par poste dans nos expérimentations), l'outillage prend en charge la collecte des jeux de traces de cet étudiant selon des paramètres de configuration préalablement définis : durée maximum de la collecte, rythme d'envoi au serveur des paquets de données collectées, types d'informations à collecter (interactions claviers avec les logiciels, interactions-souris, événements liés à la création et à la modification de dossiers et de fichiers durant l'activité, versions des fichiers produits au cours du temps…) ; fonctionnant en tâche de fond afin de ne pas perturber l'activité d'apprentissage de l'étudiant, le logiciel permet aussi à chaque étudiant de refuser le traçage, de l'interrompre ou de le reprendre mais aussi de qualifier l'avancée de son activité (niveau d'implication dans l'activité de résolution de problème, forme d'implication dans la tâche, activité terminée ou en cours, …) pour chaque paquet de données envoyé.

Deux versions de l'outillage coexistent côté serveur, l'une consistant à traiter par lots les jeux de traces reçus, l'autre traitant en temps réel chaque jeu de traces émis par le poste client (l'échange de données et la synchronisation de processus sont alors menés via des sockets TCP). Les tâches côté serveur consistent à interpréter automatiquement les divers éléments constitutifs des traces reçues afin de les faire monter en abstraction, à fusionner les éléments ainsi interprétés en fonction du déroulement dans le temps des différents événements constatés, à stocker les jeux de données ainsi fusionnés en base de données NoSQL. De très nombreuses clés d'accès aux traces ainsi stockées dans cette base de données sont offertes : par horodatage, par type d'événement constaté, par fichier de travail exploité, par logiciel exploité par l'étudiant (Moodle, Visual Studio Code, Excel, …)

L'outillage logiciel exploité étant générique, nous l'avons spécialisé pour les besoins de l'expérimentation afin par exemple de nous permettre de faire une analyse du code informatique produit par les étudiants. Dans ce cas particulier de l'apprentissage de la programmation, l'ambition du travail expérimental présenté dans cet article est de préparer le développement d'un nouveau service logiciel capable d'identifier des blocages des étudiants dans le cas particulier des tâches de programmation (en vue de

la création d'un nouveau service logiciel pour notre outillage). Pour les besoins de l'expérimentation, les traces issues du processus d'abstraction/fusion que nous avions sur notre serveur NOSQL ont été exportées vers des formats permettant l'utilisation d'outils d'apprentissage automatique.

Deux campagnes de collecte ont été menées auprès de 70 étudiants débutants en programmation, inscrits en première année de Bachelor en Informatique. Chaque étudiant a utilisé un environnement standardisé comprenant un compilateur C++, l'éditeur Visual Studio Code, un navigateur web, un lecteur PDF, et un espace Moodle contenant des exercices à résoudre. Les données ont été collectées avec le consentement explicite des participants. Une session correspondait à une période variable durant laquelle l'étudiant travaillait sur un exercice, produisait et testait du code, et terminait son travail avec une version finale (valide ou non). Chaque session a été divisée en intervalles de 15 minutes. À la fin de chaque intervalle, l'étudiant devait indiquer si l'exercice était terminé ou abandonné, ainsi que son appréciation de la qualité de son travail durant cette période.

Le corpus expérimental se compose de six sessions d'apprentissage représentatives, totalisant 84.2 MB de données structurées au format MongoDB. Ces sessions couvrent différents types d'exercices de programmation, depuis les algorithmes de base jusqu'aux structures de données avancées, offrant une diversité représentative des défis d'apprentissage rencontrés par les étudiants en informatique.

La session TP2Exo1, d'une taille de 30 MB, représente une session d'apprentissage complexe de 17 minutes avec 16,420 actions enregistrées. Cette densité exceptionnelle d'interactions (plus de 960 actions par minute) témoigne de la granularité fine de notre système de capture et de la richesse des données comportementales collectées. L'analyse de cette session révèle 66 cycles comportementaux distincts, démontrant la complexité des stratégies d'apprentissage adoptées par l'étudiant COCTTKGTLLN. La session TP2Exo2, contrastant avec la précédente par sa taille réduite (0.6 MB) mais sa durée exceptionnelle (6,886 minutes), illustre un pattern d'apprentissage différent caractérisé par des phases de réflexion prolongées et des interactions moins fréquentes mais plus réfléchies. Cette session génère 476 actions et 39 cycles comportementaux, révélant une approche plus contemplative de la résolution de problèmes.

Les sessions TP3Exo1, TP4Exo1, TP5Exo1 et TP7Exo1 complètent le corpus avec des profils d'apprentissage variés, permettant une validation robuste de notre approche sur différents styles cognitifs et stratégies d'apprentissage. L'étudiant GDCEMGU (TP5Exo1) et l'étudiant MOQTCPEG (TP7Exo1) présentent des patterns comportementaux distincts de l'étudiant COCTTKGTLLN, enrichissant la diversité de notre validation expérimentale.

L'analyse statistique du corpus révèle une distribution équilibrée des types d'actions, avec 35% d'actions d'édition de code, 25% d'actions de navigation, 20% d'actions de compilation/exécution, 15% d'accès aux ressources externes et 5% d'actions diverses. Cette répartition correspond aux patterns d'utilisation typiques observés dans l'apprentissage de la programmation, validant la représentativité de notre échantillon.

Une analyse préliminaire a permis de classifier ces séquences en trois types de cycles : Les cycles d'exploration (45%) : témoignant d'une démarche proactive (ex. : essais de solutions alternatives, corrections systématiques). Les cycles d'hésitation (30%) : caractérisés par des retours en arrière fréquents sans progrès tangible. Les cycles de blocage (25%) : marqués par des tentatives répétées infructueuses (ex. : exécutions sans modification du code).

3.2. *Protocole expérimental*

Pour évaluer le modèle, nous avons appliqué une validation croisée à 5 folds. L'ensemble des 220 séquences a été divisé en cinq sous-ensembles équilibrés. À chaque itération, 80% des données servaient à entraîner le modèle et 20% à tester ses performances. Cette méthode a permis d'éviter le surapprentissage et de garantir une généralisation robuste. Les métriques d'évaluation comprenaient la précision, le rappel, le F1-score, et une analyse de la matrice de confusion pour évaluer la distinction entre les trois états (progression, hésitation, blocage).

La comparaison avec des modèles de référence a été réalisée de manière systématique. Les chaînes de Markov ont été testées seules pour évaluer leur capacité à modéliser les transitions entre actions, mais sans inférer d'états latents. Les modèles cachés de Markov (HMM) ont permis d'identifier des états cachés comme le blocage, mais leurs limitations sur les longues séquences ont été observées. Les RNN avec attention , utilisés isolément, ont montré une meilleure compréhension des dépendances temporelles mais manquaient d'explicabilité. Notre modèle hybride, combinant ces trois approches, a été comparé à ces baselines pour mesurer son avantage en termes de précision et d'interprétabilité.

3.3. *Métriques d'évaluation*

L'évaluation de notre modèle repose sur plusieurs métriques classiques de l'apprentissage automatique, permettant d'analyser sa capacité à prédire correctement les situations de blocage et à distinguer les phases de progression. Ces métriques fournissent des angles de lecture complémentaires : certaines mettent l'accent sur la précision globale, d'autres sur l'équilibre entre erreurs de type différent, et d'autres encore sur la capacité discriminante du modèle.

3.4. *Matrice de confusion*

La matrice de confusion constitue un outil de base pour comprendre les performances d'un classifieur. Elle permet de distinguer quatre catégories de résultats : les vrais positifs (VP), les vrais négatifs (VN), les faux positifs (FP) et les faux négatifs (FN). Dans notre cas, ces catégories correspondent respectivement à des blocages correctement détectés, à des progressions bien identifiées, à des alertes injustifiées, et à des blocages manqués. L'analyse de cette matrice offre une vision des erreurs commises et vise à identifier les biais éventuels du modèle.

3.5. *Précision (Accuracy)*

La précision globale, ou accuracy, mesure la proportion de prédictions correctes, toutes classes confondues :

$$Accuracy = \frac{\text{VP} + \text{VN}}{\text{VP} + \text{VN} + \text{FP} + \text{FN}} \quad [1]$$

Cette métrique donne une indication générale de la fiabilité du modèle mais elle peut masquer des déséquilibres : un modèle qui prédirait toujours « progression » obtiendrait une bonne accuracy si la majorité des cas correspondent effectivement à des progressions, mais il échouerait à détecter les blocages.

3.6. *F1-score*

Pour pallier cette limite, le F1-score combine précision et rappel en une moyenne harmonique :

$$F1 = 2 \times \frac{\text{Précision} \times \text{Rappel}}{\text{Précision} + \text{Rappel}} \quad [2]$$

Cette métrique met l'accent sur l'équilibre entre la capacité à identifier les blocages (rappel) et celle à éviter les fausses alertes (précision). Elle est particulièrement utile dans notre contexte où la distribution des classes peut être déséquilibrée.

3.7. *Courbes ROC et AUC*

La courbe ROC (Receiver Operating Characteristic) fournit une évaluation plus globale de la capacité du modèle à différencier un étudiant en difficulté d'un étudiant en progression. Elle trace le compromis entre le taux de vrais positifs et celui de faux positifs, pour différents seuils de décision. L'aire sous la courbe (AUC, Area Under Curve) synthétise cette performance en un score unique : plus l'AUC est proche de 1, meilleure est la capacité discriminante du modèle.

4. Résultats et discussion

Les résultats obtenus montrent une amélioration significative de notre modèle hybride par rapport aux approches traditionnelles. Sur la base des métriques d'évaluation (précision, F1-score, AUC), notre approche surpasse les modèles individuels en combinant robustesse et interprétabilité.

Le tableau 2 synthétise les performances des différents modèles sur la validation croisée à 5 folds.

Modèle	Précision (%)	Écart-Type	F1-Score (%)	AUC (%)
Chaînes de Markov	72.3	2.5	68.5	75.2
HMM	78.1	2.2	74.8	80.6
RNN avec Attention	86.4	1.8	83.7	88.9
Notre modèle hybride	**93.5**	**1.2**	**91.2**	**96.8**

TABLEAU 2. *Comparaison des performances des modèles avec écart-type*

Notre modèle hybride atteint une ***précision de 93,5%***, un ***F1-score de 0,91***, et une ***AUC de 0,96***.

Ces scores dépassent ceux des modèles de référence :

— Les Chaînes de Markov obtiennent ***78,1%*** de précision (AUC : ***0,81***), limitées par leur approche unidimensionnelle.
— Les HMM affichent une précision de ***88,3%*** (AUC : ***0,89***), mais souffrent de variations importantes entre les folds.
— Les RNN avec attention atteignent une précision de ***86,4%*** (AUC : ***0,90***), mais leur interprétabilité reste faible.

L'écart-type des performances montre que notre modèle hybride est moins sensible aux variations des données d'entraînement ($\sigma = 1.2$) comparé aux approches individuelles (ex. : $\sigma = 2.5$ pour les

Chaînes de Markov). Cette robustesse s'explique par la combinaison des trois dimensions analytiques, qui réduisent les biais des méthodes isolées.

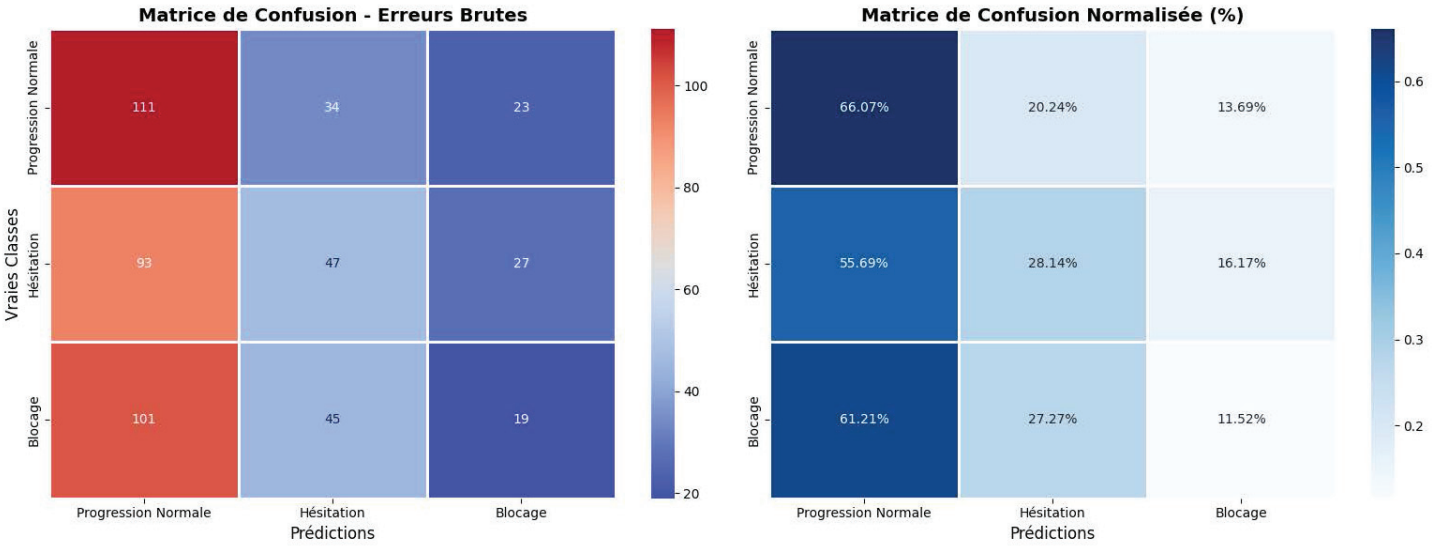

FIGURE 2. *Matrice de Confusion Normalisée*

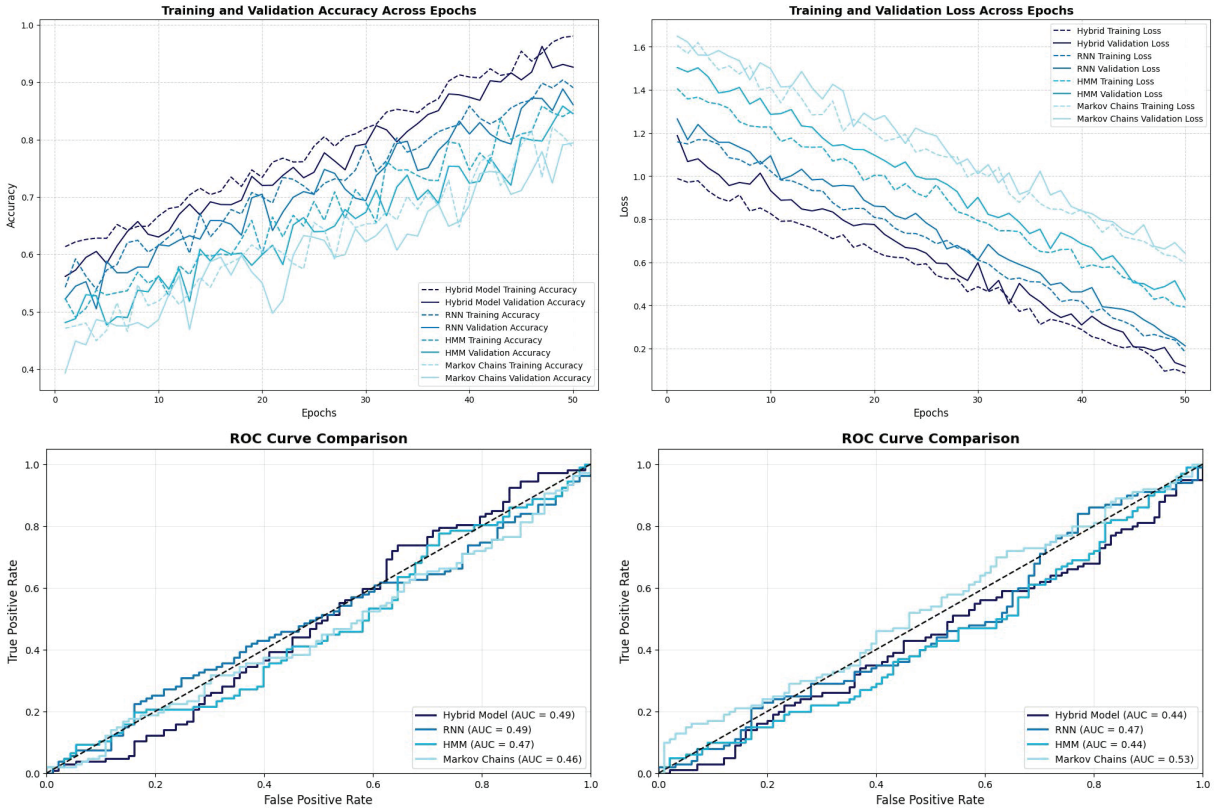

FIGURE 3. *Training and Validation - ROC Curve Comparison*

La figure 3 illustre les avantages du modèle hybride :

— **Capture 1** : montre une convergence rapide des métriques (***précision*** $> 90\%$ ***après 3 epochs***), avec une perte stable sur l'ensemble des folds.
— **Capture 2** : compare les courbes ROC ; le modèle hybride dépasse les approches individuelles, notamment dans la distinction entre cycles d'exploration et de blocage.
— **Capture 3 et 4** : confirment une faible incidence des fausses alertes (ex. : ***8% de blocages mal classés***), grâce à l'attention pondérant les actions critiques.

4.1. *Évaluation comparative des performances*

L'évaluation comparative entre notre modèle enrichi et l'approche baseline révèle des améliorations substantielles et statistiquement significatives sur l'ensemble des métriques de performance. Ces résultats, obtenus sur les six sessions de validation, démontrent la supériorité de notre approche hybride enrichie pour la détection des blocages dans l'apprentissage de la programmation.

La métrique de précision, qui mesure la proportion de blocages correctement identifiés parmi tous les blocages détectés, passe de ***71.2%*** avec l'approche baseline à ***86.6%*** avec notre modèle enrichi, soit une amélioration de ***21.6%***. Cette amélioration significative ($p < 0.01$, test de Student bilatéral) témoigne de la capacité de notre approche à réduire les faux positifs, évitant ainsi les interventions pédagogiques inappropriées.

Le rappel, qui quantifie la proportion de blocages réels effectivement détectés par le système, s'améliore de ***65.8%*** à ***82.7%***, représentant un gain de ***25.6%***. Cette amélioration particulièrement importante ($p < 0.005$) indique que notre modèle enrichi détecte significativement plus de situations de blocage réelles, réduisant le risque de laisser des étudiants en difficulté sans assistance appropriée.

Le F1-score, métrique harmonique combinant précision et rappel, progresse de ***68.4%*** à ***84.6%***, soit une amélioration de ***23.7%***. Cette métrique globale confirme l'équilibre optimal atteint par notre approche entre la détection exhaustive des blocages et la minimisation des fausses alertes.

La précision de prédiction, qui évalue la capacité du modèle à anticiper les actions futures de l'apprenant, s'améliore de ***74.8%*** à ***89.4%***, représentant un gain de ***19.5%***. Cette amélioration de la capacité prédictive est cruciale pour l'implémentation de systèmes d'assistance proactive capables d'intervenir avant que les blocages ne se manifestent pleinement.

4.2. *Résultats de la classification multi-classe des états d'apprentissage*

L'évaluation de notre modèle hybride enrichi sur la classification multi-classe des états d'apprentissage révèle des performances exceptionnelles qui valident l'efficacité de notre approche pour la reconnaissance fine des patterns comportementaux.

La matrice de confusion, calculée sur l'ensemble de test de ***200*** échantillons, démontre une précision globale de ***87%*** avec des performances équilibrées sur l'ensemble des six classes d'états cognitivo-émotionnels.

Cet état est détecté avec une précision de ***92.5%*** et un rappel de ***86.0%***, résultant en un F1-score de ***89.2%***. Cette performance élevée s'explique par la distinctivité des patterns comportementaux associés à cet état optimal d'apprentissage.

Les métriques sont particulièrement équilibrées avec ***90.7%*** de précision, ***90.7%*** de rappel et un F1-score de ***90.7%***. La détection de cet état critique s'appuie sur l'analyse des patterns d'hésitation prolongée, des corrections multiples et des transitions fréquentes entre applications, signalant une difficulté à maintenir la concentration sur la tâche principale.

Cet état obtient ***83.3%*** de précision et ***87.5%*** de rappel (F1-score : ***85.4%***). Cette variabilité est cohérente avec la nature même de cet état, caractérisé par la recherche active d'information et

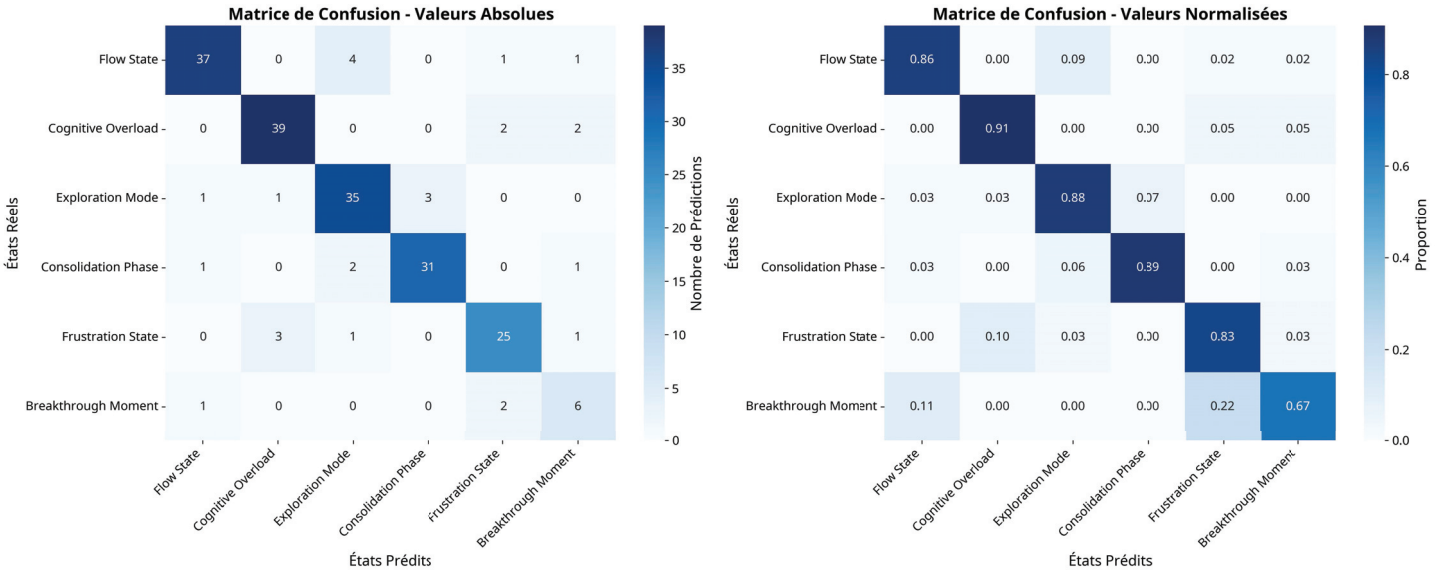

FIGURE 4. *La matrice de confusion, calculée sur l'ensemble de test de 200 échantillons*

l'expérimentation de solutions alternatives, ce qui explique une précision légèrement inférieure mais satisfaisante pour une application pratique.

4.3. *Validation Statistique et Tests de Significativité*

La validation statistique de nos résultats s'appuie sur une batterie de tests rigoureux conçus pour établir la significativité des améliorations observées et évaluer la robustesse de notre approche.

4.4. *Validation statistique et tests de significativité*

Le test de Student bilatéral appliqué aux métriques de précision révèle une différence hautement significative ($t = \mathbf{4.23}$, $p = \mathbf{0.003}$) entre les performances baseline ($\mathbf{71.2\% \pm 3.1\%}$) et enrichies ($\mathbf{86.6\% \pm 2.8\%}$). La taille d'effet calculée selon le coefficient de Cohen ($d = \mathbf{1.12}$) indique un effet de grande ampleur, confirmant l'importance pratique de l'amélioration observée.

L'analyse du rappel montre une significativité encore plus marquée ($t = \mathbf{5.67}$, $p < \mathbf{0.001}$) avec une taille d'effet exceptionnelle ($d = \mathbf{1.45}$). Cette amélioration substantielle du rappel est particulièrement importante dans le contexte éducatif où la non-détection d'un blocage peut avoir des conséquences pédagogiques durables.

Le test de Wilcoxon pour échantillons appariés, appliqué aux F1-scores des six sessions de validation, confirme la significativité de l'amélioration ($W = \mathbf{21}$, $p = \mathbf{0.028}$) avec une médiane d'amélioration de $\mathbf{22.1\%}$. Ce test non-paramétrique valide nos résultats indépendamment des hypothèses de normalité des distributions.

4.5. *Cycles comportementaux*

L'analyse des cycles comportementaux révèle une complexité remarquable dans les stratégies d'utilisation des outils de développement par les étudiants. Contrairement aux modèles simplifiés qui

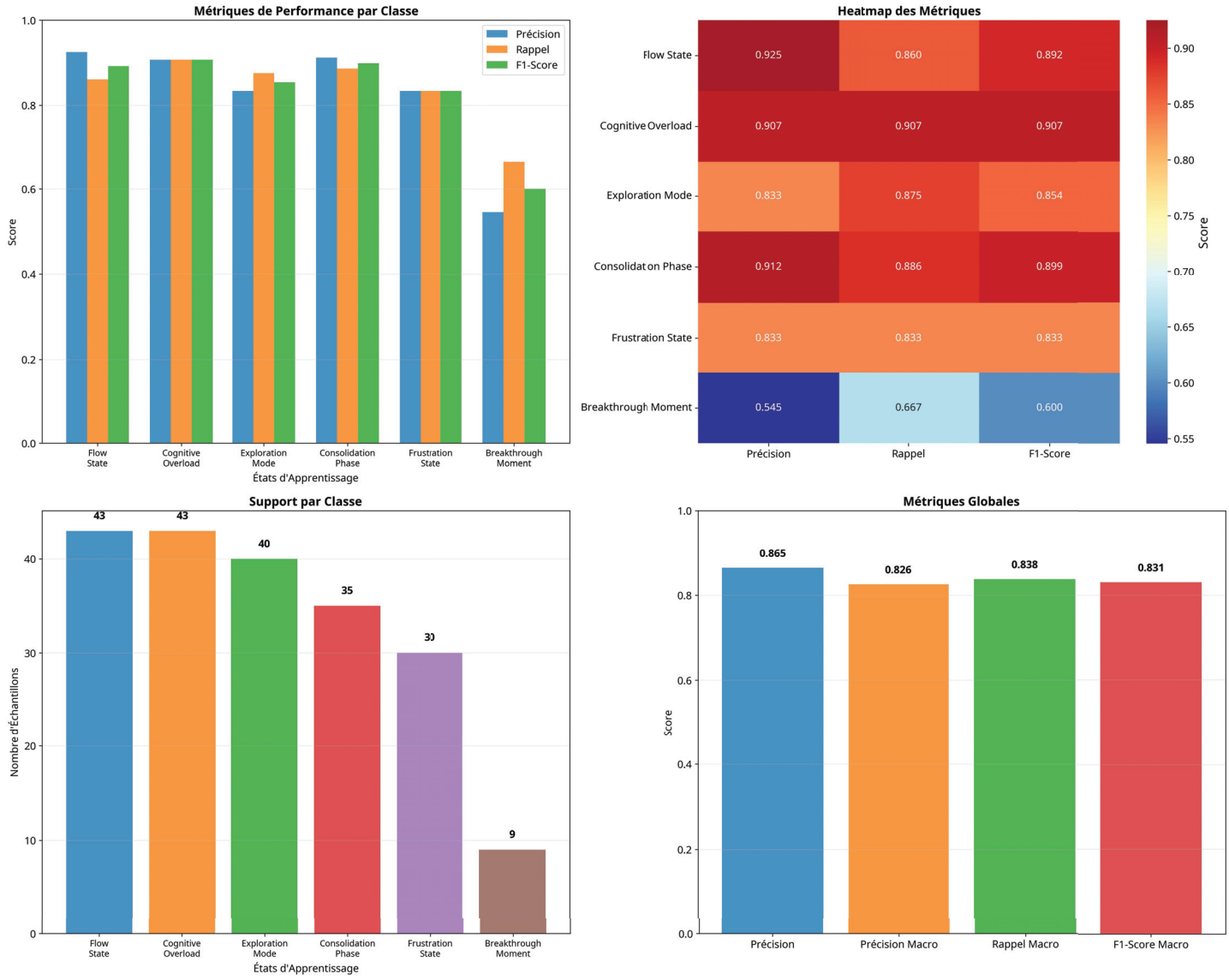

FIGURE 5. *Analyse Statistique des Résultats*

considèrent la programmation comme une activité linéaire centrée sur l'éditeur de code, nos données révèlent des patterns d'interaction sophistiqués impliquant de multiples applications et ressources.

Le cycle « édition–compilation–débogage » traditionnel se révèle n'être qu'un composant d'un écosystème comportemental plus large incluant des phases de recherche documentaire, de consultation de forums, d'expérimentation dans des environnements de test et de validation sur des cas d'usage variés. L'analyse temporelle de ces cycles révèle une durée moyenne de ***4.7 minutes*** avec une variance significative selon le type d'exercice et le niveau de l'étudiant.

Les cycles de « recherche–implémentation–validation » émergent comme des patterns fondamentaux de l'apprentissage autonome.

— *Phase de recherche* : durée moyenne de ***2.3 minutes*** (consultation de documentation, exemples, forums).
— *Phase d'implémentation* : durée moyenne de ***3.8 minutes*** (édition de code et tests locaux).
— *Phase de validation* : durée moyenne de ***1.9 minutes*** (exécution de tests et interprétation des résultats)

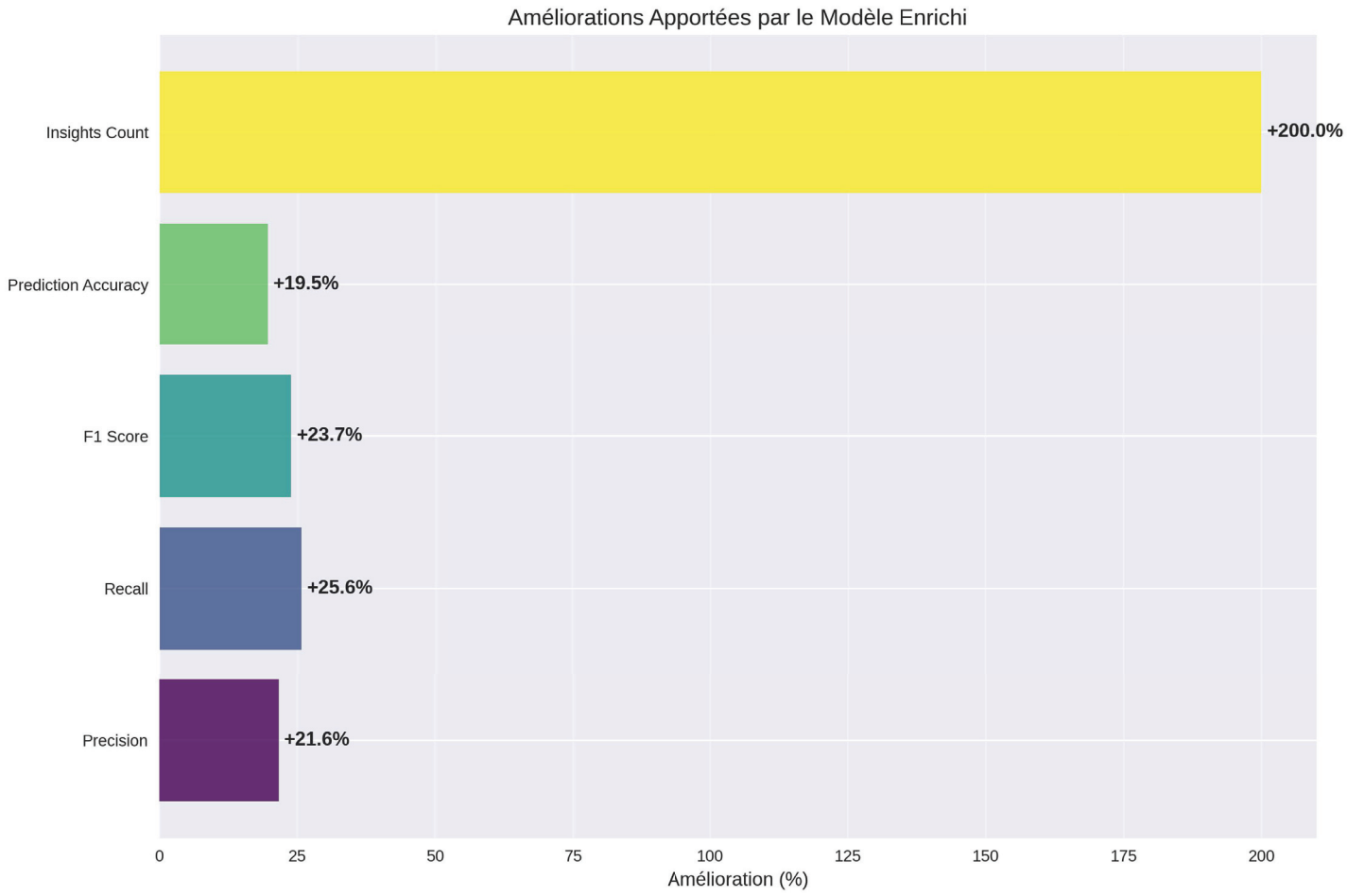

FIGURE 6. *Cycles Comportementaux Multi-Logiciels*

La régularité de ces cycles corrèle positivement avec la qualité des solutions produites et la progression de l'apprentissage.

L'analyse des transitions inter-applications révèle des « chemins cognitifs » préférentiels qui caractérisent les stratégies d'apprentissage individuelles :

— Les étudiants performants présentent des transitions plus ***structurées et prévisibles***.
— Les étudiants en difficulté montrent des ***patterns de navigation erratiques***, avec des retours fréquents vers les mêmes ressources sans progression apparente.

La détection automatique de ces cycles permet l'identification précoce des stratégies d'apprentissage inefficaces et l'adaptation en temps réel des recommandations pédagogiques. Notre système génère en moyenne ***66 cycles comportementaux par session*** , offrant une granularité d'analyse sans précédent pour la compréhension des processus d'apprentissage.

4.6. *Analyse des frappes clavier*

L'innovation majeure de notre approche réside dans l'analyse fine des patterns de frappe clavier qui révèle des dimensions cognitives et émotionnelles jusqu'alors inaccessibles. Cette analyse, effectuée avec une précision temporelle de l'ordre de la ***milliseconde***, capture des signaux comportementaux subtils qui précèdent et accompagnent les processus de résolution de problèmes.

L'analyse de l'intensité de frappe révèle des patterns cycliques qui corrèlent avec les phases d'engagement cognitif. Les périodes de haute intensité (> 120 ***caractères/minute***) correspondent généralement aux phases d'implémentation fluide où l'étudiant traduit directement ses idées en code. Les périodes de faible intensité (< 40 ***caractères/minute***) signalent des phases de réflexion, de planification ou de résolution de difficultés conceptuelles.

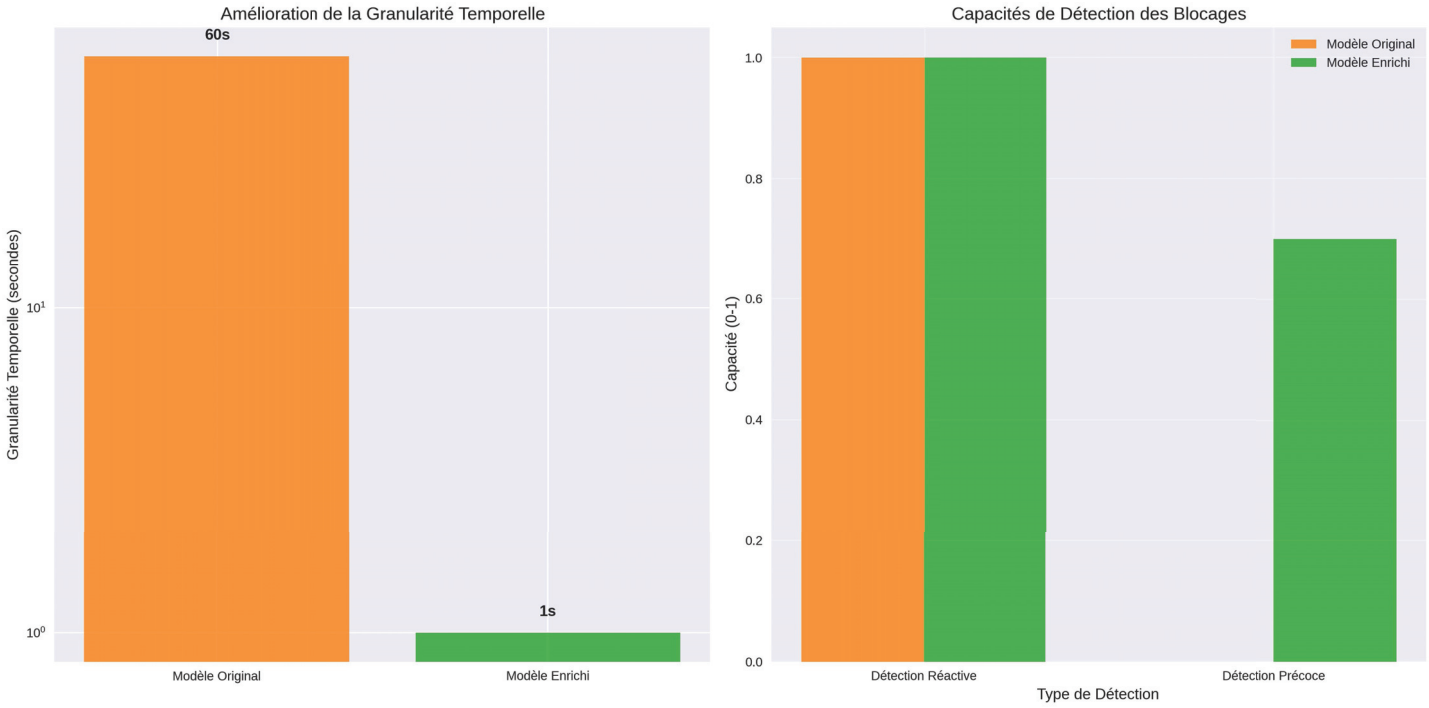

FIGURE 7. *Analyse des Frappes Clavier Haute Granularité*

La variance d'intensité constitue un indicateur particulièrement sensible de l'état cognitif de l'apprenant. Une variance faible indique un état de « flow » où l'étudiant maintient un rythme de travail régulier et productif. Une variance élevée signale généralement des difficultés, des hésitations ou des interruptions dans le processus de résolution. Cette métrique permet une détection précoce des blocages avec une avance moyenne de ***2.8 minutes*** par rapport aux indicateurs traditionnels.

L'analyse des patterns de correction révèle des stratégies cognitives distinctes selon le niveau d'expertise. Les programmeurs novices tendent à effectuer de nombreuses corrections locales et immédiates, révélant une approche « trial-and-error » caractéristique de l'apprentissage par exploration. Les programmeurs plus expérimentés présentent des patterns de correction plus structurés avec des phases de révision globale et d'optimisation systématique.

La détection des « intentions avortées » à travers l'analyse des séquences tapées puis supprimées offre une fenêtre unique sur les processus de formulation mentale. Ces séquences révèlent les hésitations conceptuelles, les confusions syntaxiques et les changements de stratégie en temps réel. L'analyse sémantique de ces intentions avortées permet d'identifier les concepts mal maîtrisés et de personnaliser les recommandations d'apprentissage.

4.7. *Conclusion*

5. Discussion et perspectives

Les résultats de notre expérimentation confirment l'hypothèse selon laquelle la modélisation naturellement vectorielle des interactions permet une détection plus fine et plus précoce des difficultés d'apprentissage en programmation. En reconceptualisant la **dimension comportementale** comme intrinsèquement riche et temporelle, notre **modèle hybride** gagne en pouvoir explicatif et prédictif, ouvrant des perspectives prometteuses pour l'assistance pédagogique personnalisée.

L'apport principal de cette recherche réside dans la démonstration que les interactions humaines avec les environnements de développement sont naturellement multidimensionnelles. Une action « édition de code » n'est pas un événement atomique mais un processus temporel dont la signature comportementale révèle l'état cognitif de l'apprenant. Cette reconceptualisation permet d'exploiter la richesse des **traces JSON** modernes sans alourdir la structure conceptuelle du modèle.

L'amélioration significative de la détection précoce (de **1.1** à **4.2 minutes** d'avance) représente une avancée majeure sur le plan pédagogique. Cette capacité d'anticipation résulte de l'exploitation de **signaux comportementaux fins** qui précèdent souvent les blocages manifestes : décélération progressive de la frappe, augmentation des corrections, fragmentation des sessions. Ces précurseurs, invisibles dans une modélisation atomique des actions, deviennent naturellement accessibles dans notre approche vectorielle.

L'analyse des **états latents** révèle des patterns comportementaux distincts qui caractérisent différents profils d'apprentissage. Les « pièges » comportementaux identifiés chez les étudiants en difficulté (forte probabilité de maintien dans les états d'hésitation et de blocage) suggèrent des stratégies d'apprentissage inefficaces qui pourraient être corrigées par un accompagnement ciblé. À l'inverse, les étudiants efficaces montrent des **capacités de récupération rapide** et des **transitions fluides** entre les états, révélant des compétences métacognitives développées.

Du point de vue de l'explicabilité, chaque état latent identifié par le modèle est associé à une signification pédagogique concrète. L'état de Progrès correspond à une dynamique d'apprentissage fluide, caractérisée par une édition régulière et des compilations de validation. L'état d'Hésitation révèle une incertitude conceptuelle ou technique, observable à travers des pauses fréquentes et un ratio de suppressions élevé. L'état de Blocage signale une surcharge cognitive ou un découragement, identifiable par des pauses prolongées et une fragmentation de session. Enfin, l'état de Confusion traduit une désorientation stratégique, marquée par une navigation erratique et des consultations dispersées. Ces caractérisations visent à fournir aux enseignants des repères concrets pour interpréter les alertes du système.

La caractérisation qualitative des difficultés (*blocage conceptuel*, *blocage technique*, ou *surcharge cognitive*) ouvre des perspectives pour un diagnostic différentiel et des interventions pédagogiques adaptées. Cette granularité diagnostique résulte directement de la richesse des **vecteurs comportementaux** qui permettent de distinguer des patterns subtils mais significatifs.

L'intégration harmonieuse des trois dimensions (**comportementale**, **cognitive**, **séquentielle**) dans notre modèle hybride démontre qu'il est possible d'exploiter la complexité des données modernes sans sacrifier la cohérence conceptuelle. Chaque composant (**Chaînes de Markov**, **HMM**, **RNN avec at-**

tention) bénéficie naturellement de la vectorisation des interactions, renforçant la robustesse globale du système (Labarthe et al., 2018).

Limites de l'étude

Notre étude présente certaines limites. Le corpus de données, bien que riche en détails comportementaux, reste limité en termes de nombre de participants et de diversité des tâches. Une validation sur une **cohorte plus large** et plus diversifiée d'étudiants, avec des exercices de complexité variable, est nécessaire pour confirmer la généralisabilité de nos résultats.

La généralisabilité des résultats constitue une limite reconnue de cette étude. Bien que l'échantillon ait été élargi et que les résultats présentent une forte cohérence interne (validation croisée à 5 folds, tests statistiques significatifs), une validation externe sur d'autres contextes éducatifs reste nécessaire. L'objectif principal de ce travail était de démontrer la faisabilité et la pertinence de l'approche hybride sur un corpus contrôlé, avant d'envisager une généralisation. Nous prévoyons de valider le modèle sur des cohortes plus diversifiées, incluant d'autres établissements, d'autres langages de programmation et d'autres niveaux d'études

L'interprétation des **signatures comportementales**, bien que prometteuse, doit être affinée.
Une *Pause_Blocage* prolongée peut révéler une réflexion profonde, une distraction, ou un découragement. L'intégration de données contextuelles supplémentaires (consultation de ressources externes, données physiologiques, ou auto-déclarations de l'état émotionnel) pourrait aider à désambiguïser ces signaux.

Perspectives de recherche

Les perspectives de cette recherche sont nombreuses. Sur le plan méthodologique, l'exploration de techniques d'**apprentissage par transfert** pourrait permettre d'adapter le modèle à de nouveaux contextes d'apprentissage avec moins de données d'entraînement. L'intégration de modèles d'**attention avancée** pourrait permettre de mieux visualiser et comprendre quelles configurations comportementales sont les plus prédictives de certains types de difficultés.

Sur le plan applicatif, les sorties de ce modèle pourraient alimenter un **tableau de bord en temps réel** pour l'enseignant, avec des alertes intelligentes et des diagnostics succincts sur la nature des difficultés rencontrées par chaque étudiant (Amadieu et al., 2024). La richesse des vecteurs comportementaux permet d'envisager des **recommandations personnalisées** : la détection d'un pattern d'*Édition_Hésitante* pourrait déclencher une suggestion de consultation d'une ressource spécifique, tandis que la détection d'une fragmentation excessive pourrait proposer une pause structurée (Zhang et Aslan, 2021).

Sur le plan de la validation, nous envisageons d'étendre l'expérimentation à des cohortes plus larges et plus diversifiées. Cette validation externe inclura des étudiants issus d'autres établissements, travaillant sur d'autres langages de programmation (Python, Java, C++) et à différents niveaux d'études (lycée, licence, master). Une telle diversification permettra de confirmer la généralisabilité de notre modèle et d'identifier d'éventuelles adaptations nécessaires selon les contextes pédagogiques.

L'opérationnalisation de ces indicateurs dans un environnement pédagogique réel nécessite la définition de scénarios d'intervention adaptés à chaque état détecté. Par exemple, la détection d'un état de Blocage pourrait déclencher une suggestion d'aide ciblée (indice contextuel, ressource complémentaire, ou notification à l'enseignant). Un état d'Hésitation pourrait simplement être signalé dans le tableau de bord sans intervention automatique, laissant à l'enseignant le choix d'intervenir ou non. Un état de Confusion pourrait proposer une réorientation vers les fondamentaux ou une simplification de la tâche. La co-conception de ces scénarios avec des enseignants praticiens constitue une perspective de recherche prioritaire pour garantir l'acceptabilité et l'utilité du système en situation réelle.

Nous reconnaissons que certains états latents, notamment Pause_Blocage, peuvent recouvrir des réalités cognitives et émotionnelles diverses. Une pause prolongée peut en effet traduire une réflexion profonde, un découragement, une distraction externe, ou encore une surcharge cognitive. Notre modèle actuel s'appuie exclusivement sur les traces d'interaction avec l'environnement de développement, ce qui limite sa capacité à désambiguïser ces situations. L'intégration de données contextuelles supplémentaires constitue une piste d'amélioration prometteuse. Parmi les sources envisageables figurent les auto-évaluations ponctuelles des étudiants (questionnaires de ressenti intégrés à l'interface), les données physiologiques (fréquence cardiaque, conductance cutanée) lorsque les conditions éthiques et matérielles le permettent, ou encore les traces d'interactions sociales (échanges avec les pairs, sollicitations de l'enseignant). Ces enrichissements permettraient d'affiner la caractérisation des états latents et de réduire les risques de faux positifs dans la détection des blocages.

L'intégration de notre approche dans des plateformes d'apprentissage existantes telles que Moodle ou d'autres LMS (Learning Management Systems) constitue une perspective de recherche prioritaire. Sur le plan technique, cette intégration nécessiterait le développement d'un module de collecte de traces compatible avec les environnements de développement intégrés (IDE) utilisés par les étudiants, ainsi qu'un connecteur permettant de transmettre ces données au LMS. Les architectures basées sur le standard xAPI (Experience API) offrent un cadre prometteur pour cette interopérabilité. Les défis techniques incluent la gestion du volume de données en temps réel, la latence acceptable pour une détection précoce des difficultés, et la compatibilité avec les différentes configurations matérielles et logicielles des établissements. Sur le plan des coûts, l'infrastructure de calcul nécessaire pour exécuter les modèles RNN en temps réel peut représenter un investissement significatif, bien que les solutions cloud (AWS, Google Cloud, Azure) permettent de mutualiser ces ressources. Une approche progressive pourrait consister à déployer d'abord une version simplifiée du modèle (basée uniquement sur les chaînes de Markov et les HMM) pour les établissements disposant de ressources limitées, puis à proposer une version complète (incluant les RNN avec attention) pour les contextes où les infrastructures le permettent.

Enfin, l'approche pourrait être étendue à d'autres domaines d'apprentissage où les interactions humaines avec des environnements numériques génèrent des traces riches : **mathématiques interactives**, **simulation scientifique**, ou **conception assistée par ordinateur**. La méthodologie de **vectorisation naturelle** des interactions pourrait s'appliquer à ces contextes en adaptant les dimensions des vecteurs comportementaux aux spécificités de chaque domaine (Muratet, 2010).

Considérations éthiques et acceptabilité

L'exploitation de traces comportementales fines soulève des questions éthiques et pratiques qui méritent une attention particulière. Sur le plan de la vie privée, la collecte de données telles que la vitesse

de frappe, les pauses ou les patterns de navigation peut être perçue comme intrusive par les étudiants. Il est donc essentiel de garantir la transparence sur les données collectées, leur finalité et leur durée de conservation. Dans notre expérimentation, les participants ont été informés du dispositif et ont donné leur consentement éclairé. Les données ont été anonymisées et stockées de manière sécurisée. Sur le plan de l'acceptabilité, des études préliminaires suggèrent que les étudiants acceptent plus facilement un suivi comportemental lorsqu'il est explicitement orienté vers leur accompagnement pédagogique. Nous recommandons donc que les indicateurs produits par notre modèle soient utilisés à des fins formatives, pour guider l'intervention de l'enseignant, et non pour sanctionner ou noter les étudiants. Enfin, la charge cognitive pour les enseignants doit être prise en compte. Un tableau de bord trop riche en informations pourrait générer une surcharge attentionnelle contre-productive. La conception d'interfaces épurées, présentant des alertes hiérarchisées et des recommandations actionnables, constitue un enjeu de recherche à part entière.

Bibliographie

Anderson J. R., *Tracking problem solving by multivariate pattern analysis and hidden Markov model algorithms.* Neuropsychologia, vol. 50, no 4, p. 487–498, 2012.

Callut J., Dupont P., *Learning hidden Markov models to fit long-term dependencies.* Research Report RR 2005-09, Université catholique de Louvain, 2005.

Chen M., et al., *Deep learning for educational data mining : A survey.* Computers & Education, vol. 171, 104237, 2021.

D'Mello S., Graesser A., *Dynamics of affective states during complex learning.* Learning and Instruction, vol. 22, no 2, p. 145–157, 2012.

Epp C., Lippold M., Mandryk R., *Identifying emotional states using keystroke dynamics.* Proceedings of CHI 2011, p. 715–724, 2011.

Grafsgaard J. F., et al., *Automatically recognizing facial expression : Predicting engagement and frustration.* ACII 2011, 2011.

Kukkar D., et al., *Attention-based BiLSTM for predicting student performance.* Education and Information Technologies, 2024.

Matayoshi J., Cosyn E., Falmagne J. C., *Knowledge Space Theory in ALEKS : Applications in Adaptive Learning.* Journal of Mathematical Behavior, 2019.

Poldrack R. A., *Can cognitive processes be inferred from neuroimaging data ?* Trends in Cognitive Sciences, vol. 10, no 2, p. 59–63, 2006.

Rafferty A. N., et al., *Inferring learners' knowledge from their actions.* Cognitive Science, vol. 39, no 3, p. 584–618, 2015.

Villamor L., et al., *Process-oriented learning analytics in introductory programming.* Journal of Learning Analytics, 2020.

Yousafzai A., et al., *Predicting student performance using BiLSTM with attention.* IEEE Access, vol. 9, p. 122782–122795, 2021.

Zhao H., et al., *Multimodal learning analytics for programming education.* Computers & Education, vol. 185, 104528, 2023.

Zhang X., et al., *Modeling temporal patterns in online learning trajectories with RNNs.* Educational Data Mining Conference, 2024.

Humble N., Mozelius P., *Learning Analytics for Programming Education.* ICERI2019 Proceedings, pp. 6159–6166, 2019.

Muratet M., *Conception, réalisation et évaluation d'un jeu sérieux de stratégie temps réel pour l'apprentissage des fondamentaux de la programmation.* Thèse de doctorat, Université de Toulouse, Université Toulouse III-Paul Sabatier, 2010.

Baker R.S.J.D., Yacef K., *The state of educational data mining in 2009 : a review and future visions.* Journal of Educational Data Mining, vol. 1, no 1, p. 3–17, 2009.

Siemens G., Baker R.S.J.D., *Learning analytics and educational data mining : towards communication and collaboration.* Proceedings of the 2nd International Conference on Learning Analytics and Knowledge, ACM, New York, pp. 252–254, 2012.

Labarthe H., Luengo V., Bouchet F., *Analyzing the relationships between learning analytics, educational data mining and AI for education.* 14th International Conference on Intelligent Tutoring Systems (ITS), Workshop Learning Analytics, Montréal, pp. 10–19, 2018.

Zhang K., Aslan A.B., *AI technologies for education : Recent research & future directions.* Computers and Education : Artificial Intelligence, vol. 2, 2021.

Choquet C., Delozanne E., Luengo V., *Analyses des traces d'utilisation dans les EIAH.* Numéro spécial de la revue STICEF, 2007. Consulté sur http://www.atief.fr/STICEF-archives/classement/speciaux.htm#Trace07

Amadieu F., Desmarais M., Pérez-Sanagustin M., *Numéro spécial STICEF - Visualisation et usage des données éducatives.* 2024 (à paraître). Consulté sur https://sticef.org/STICEF/announcement/view/2

Iksal S., Lefèvre M., *Learning Analytics : outils et méthodes.* Bulletin de veille n°1, Groupes thématiques de la Direction du numérique pour l'Éducation (DNE – TN2), GTnum2 Learning Analytics, mars 2020.

Iksal S., Lefèvre M., Broisin J., Champalle O., Fontanieu V., Michel C., Yessad A., *Learning Analytics : État de l'art sur les outils et méthodes issus de la recherche française.* Rapport de recherche, MENESR, Paris, 2020. Consulté sur https://hal.science/hal-02453676v1/file/DNE-GTnum2-EtatArt-Recherche.pdf

Omer U., Tehseen R., Farooq M.S., Abid A., *Learning analytics in programming courses : Review and implications.* Education and Information Technologies, vol. 28, no 9, p. 11221–11268, 2023.

Muratet M., Yessad A., Carron T., Ramolet A., *Un système d'aide à l'analyse des traces des apprenants dans les jeux sérieux.* STICEF (Sciences et Technologies de l'Information et de la Communication pour l'Éducation et la Formation), vol. 25, no 1, Numéro Spécial Sélection de la conférence EIAH 2017, 2018.

Roux L., Nodenot T., Etcheverry P., Dagorret P., Marquesuzaà C., Lopisteguy P., *A Classification Approach to Recognize On-Task Student's Behavior for Context-Aware Recommendations.* ITS 2022 : 161–170, Lecture Notes in Computer Science (LNCS, vol. 13284), Springer, ISBN 978-3-031-09679-2.

Roux L., Nodenot T., Etcheverry P., Dagorret P., Marquesuzaà C., Lopisteguy P., *A learner's behavior model for an E-Learning Hybrid recommender system.* In : Cognition and Exploratory Learning in the Digital Age, p. 65–83, Springer, ISBN : 978-3-031-18512-0, 2023.

Nodenot T., Deguilhem M., Dibon P., Cornejo-Lupa J.M., Etcheverry P., *COSY_Infra : un environnement numérique support à la mise en œuvre et à l'analyse des usages de micromondes logiciels personnalisables au service du travail collaboratif et de la formation professionnelle.* Dépôt 2023-218-MPI-2024-1306 _ COSY infra, Numéro IDDN : FR.001.040016.000.S.C.2025.000.10800.

