The Digital Proctor: A Comprehensive Analysis of Behavioral Learning Analytics and Interaction Modules in Programming Education

Executive Summary

The field of educational technology for computer science is on the cusp of a paradigm shift, moving from static, outcome-based assessment to dynamic, process-oriented behavioral analytics. Traditional learning analytics, largely confined to Learning Management Systems (LMS), offer a sparse and retrospective view of student engagement, tracking metrics such as login frequency and assignment submissions. This report outlines a new technological frontier: the integration of high-fidelity interaction modules directly within the programming Integrated Development Environment (IDE). By capturing and analyzing the full spectrum of a learner's real-time behavior—every keystroke, mouse movement, paste event, and debugging cycle—we can unlock unprecedented opportunities for personalized pedagogical support, robust academic integrity, and a deeper understanding of the cognitive processes underlying programming education.

This report provides a comprehensive analysis of the architecture, application, and implications of such a system. Part I details the data acquisition layer, exploring the technical methodologies for capturing granular user interactions, including keystroke logging and advanced session replay technology. Part II examines the dual use of this data stream through the lens of behavioral biometrics, demonstrating how typing and mouse patterns can provide continuous, passive authentication for academic integrity while also serving as a proxy for the learner's cognitive and affective state. Part III delves into the application of artificial intelligence, particularly Long Short-Term Memory (LSTM) networks, to model this sequential data and predict moments of student struggle, differentiating between productive cognitive effort and unproductive frustration.

Part IV translates these analytical capabilities into transformative applications, including a new forensic paradigm for academic integrity that transcends the limitations of

similarity-based plagiarism detectors, and the creation of a closed-loop pedagogical system where intelligent tutoring and adaptive scaffolding are triggered in real-time by detected student needs. Finally, Part V confronts the profound ethical and legal challenges inherent in this level of student monitoring. It addresses the "surveillance dilemma," the potential for algorithmic bias, and the critical need for a robust governance framework built on principles of transparency, student agency, and data privacy. This report concludes that while the technological potential is immense, the successful and responsible implementation of behavioral learning analytics hinges on an unwavering commitment to an ethical, "human-in-the-loop" framework that prioritizes the pedagogical mission of supporting the learner above all else.

Part I: The Data Acquisition Layer: Capturing the Full Spectrum of Learner Interaction

The foundation of any advanced behavioral analytics system is the quality and granularity of its input data. In the context of programming education, this requires moving data collection from the periphery of the LMS to the very heart of the learning experience: the Integrated Development Environment (IDE). This section details the technical architecture and methodologies for capturing a high-fidelity, continuous stream of interaction data, transforming the IDE from a simple coding tool into a sophisticated sensor for the learning process.

Section 1.1: High-Fidelity Logging in Educational IDEs

Capturing the nuanced process of software development requires a suite of logging techniques capable of recording not just the final code, but the sequence of actions that produced it. For modern, browser-based educational IDEs, this is achieved primarily through client-side technologies that monitor user interactions with the application's interface.

Technical Deep-Dive into Logging Techniques

The selection of a logging technique involves a trade-off between data fidelity, implementation complexity, performance impact, and intrusiveness. For educational

applications, the optimal approach must be both effective and ethically sound.

- **API-based & JavaScript-based Logging:** This is the most prevalent and practical method for web-based IDEs.[1] It leverages the browser's native capabilities to listen for user-generated events. By attaching JavaScript event listeners such as onKeyUp(), onPaste(), and onMouseMove() to the IDE's text editor and other UI elements, a rich stream of interaction data can be captured.[1] This method hooks directly into the application's programming interfaces (APIs), allowing it to record specific data points with high precision, including the timestamp of a keypress, its duration (dwell time), and the name of the key used.[2] This approach is relatively non-intrusive from a system perspective, as it operates within the sandboxed environment of the browser and does not require special permissions or software installation on the user's machine.
- **Kernel-based Logging:** This technique operates at a much deeper level of the operating system. A kernel-based logger intercepts keystrokes as they pass through the OS kernel, granting it access to everything typed on the system, regardless of the application.[1] While exceptionally powerful, this method is difficult to implement, requires root-level access, and is frequently associated with malicious software like rootkits.[1] For these reasons, kernel-based logging is both technically impractical and ethically indefensible for an educational platform. Its discussion is relevant only to provide a complete taxonomy of logging methods and to highlight the profound difference in intrusiveness compared to browser-based techniques.
- **Clipboard Event Monitoring:** A critical subset of API-based logging is the specific monitoring of clipboard events, which is foundational for modern academic integrity tools.[1] By programmatically accessing the system's ClipboardManager, an application can be notified when a paste event occurs.[3] The system can then inspect the pasted content, checking its MIME type to ensure it is plain text (MIMETYPE_TEXT_PLAIN) and logging the size and content of the pasted data.[3] This mechanism provides a direct, unambiguous signal of code being introduced from an external source, forming the basis for forensic tools like PasteTrace, which analyzes these events to track code provenance.[4]

Session Replay Technology

Session replay represents the state-of-the-art in interaction logging, moving beyond discrete event logs to create a holistic, reconstructable record of the user's session. This technology, originally developed for commercial software debugging and user experience (UX) research, is uniquely suited for the challenges of understanding the programming process.[6]

- **Mechanism:** Contrary to what its name suggests, session replay does not typically

involve video recording. Instead, it logs every mutation to the web page's Document Object Model (DOM)—the underlying structure of the page—along with all user inputs like mouse movements, clicks, scrolls, and keyboard entries.[7] This stream of events is then sent to a server where it can be used to reconstruct a pixel-perfect, video-like playback of the user's session, showing exactly what the user saw and did.[7]

- **Advantages for Education:** The primary advantage of this approach is the unparalleled qualitative context it provides. While a simple log might show a student produced 50 compiler errors, a session replay can show the *exact* sequence of edits, hesitations, and debugging attempts that led to those errors.[6] For an instructor, this is the difference between seeing a failing grade and understanding the misconception that caused it. It allows for the precise reproduction of bugs and a deep understanding of student workflows and pain points, making it an invaluable tool for both pedagogical assessment and debugging support.[6] This convergence of technology from the commercial software development world into education is a significant enabler for advanced learning analytics. The tools that help a professional developer understand a customer's bug report are precisely the tools an instructor needs to understand a student's cognitive bug. This implies that EdTech platforms can leverage mature, commercially-proven technologies as a foundation, rather than building this complex data-capture infrastructure from scratch.
- **Privacy-by-Design:** A critical feature of DOM-based session replay is its inherent support for privacy. Because the session is a reconstruction, not a recording, sensitive data (such as passwords in a form field or personal information in a comment) can be programmatically identified and obscured during the reconstruction process.[7] This is a fundamental advantage over simple screen recording and is essential for compliance with data privacy regulations like the General Data Protection Regulation (GDPR).

Section 1.2: Architectural and Security Considerations for Educational IDEs

The environment in which data is captured is as important as the capture method itself. The architecture of the educational IDE has profound implications for functionality, security, and scalability.

- **Client-side vs. Server-side Execution:** Online IDEs generally follow one of two architectural models.[8] Client-side environments execute all code within the learner's browser, using technologies like JavaScript or transpilers. This approach is simple and scalable but is severely limited in the complexity of the programming tasks it can support. In contrast, server-based environments, such as those provided by Codio, execute the learner's code on a remote server.[8] This allows the platform to spin up complex,

containerized development environments on demand—from a simple Python stack to a GPU-powered Jupyter notebook—providing a much more authentic and powerful programming experience.[9]

- **The Untrusted Code Problem:** The power of server-side execution comes with a significant security challenge: the system must execute untrusted, and potentially malicious, code submitted by thousands of learners.[8] A robust security posture is therefore non-negotiable. The standard solution is the use of sandboxing and containerization technologies (like Docker) to isolate each student's execution environment. This ensures that any code, whether buggy or intentionally malicious, cannot affect the host system or the environments of other students. This security requirement is a primary driver for the adoption of the Software-as-a-Service (SaaS) model in this space, as it centralizes the complex task of secure code execution.
- **Platform Integration:** To be truly effective, the rich behavioral data generated within the IDE must not exist in a silo. It must be integrated with the institution's broader data ecosystem, most notably the LMS.[10] Modern educational platforms must provide robust APIs to allow for seamless integration with systems like Canvas.[11] This enables the correlation of fine-grained behavioral data (e.g., debugging patterns) with macro-level student data (e.g., demographics, prior course performance, grades), creating a holistic view of the learner.

The move from the static, sparse data of traditional educational systems to the continuous, high-resolution data stream captured within the IDE represents a fundamental change in the nature of learning analytics. It necessitates a complete rethinking of data infrastructure, away from simple relational databases and toward time-series databases and stream-processing architectures capable of ingesting and analyzing thousands of events per student per session. This is a shift from analyzing the *product* of learning to analyzing the *process* of learning, opening up an entirely new set of questions about how students think, struggle, and ultimately succeed.

The following table provides a comparative analysis of the primary interaction logging techniques discussed, evaluated for their suitability in an educational context.

Technique	Data Fidelity	Implementation Complexity (Web IDE)	Performance Impact	Intrusiveness / Privacy Risk	Suitability for Education
API/JS-based Logging	High	Low	Low	Low-Medium	**High**. The standard, most practical method for

					capturing specific events like keystrokes and pastes.
DOM-based Session Replay	Very High	Medium	Low-Medium	Medium (Mitigated by data masking)	**Very High**. Provides the richest qualitative context for understanding the learning process.
Kernel-based Logging	Very High	N/A (Impractical for Web)	High	Very High	**None**. Ethically and technically unsuitable for educational use. Associated with malware.
Hardware-based Logging	Very High	N/A (Impractical for Web)	None	Very High	**None**. Requires physical access to the user's machine, making it irrelevant for online education.

Part II: Behavioral Biometrics: From Authentication to Cognitive Profiling

The high-fidelity data stream captured from the IDE serves a dual purpose. On one hand, it contains unique, stable patterns of user behavior that can be used for identity verification, forming the basis of a sophisticated academic integrity system. On the other hand, deviations from these stable patterns can provide a rich, implicit signal of the user's changing cognitive and affective state. This section explores this duality, demonstrating how the same behavioral data can function as both a digital proctor and an empathetic tutor.

Section 2.1: Keystroke Dynamics for Continuous Identity Verification

Keystroke dynamics, or typing biometrics, is a behavioral biometric that identifies individuals based on their unique manner and rhythm of typing, rather than the content of what they type.[12] The concept has a surprisingly long history, originating with military intelligence efforts in the era of the telegraph, where experienced operators could identify each other by their distinctive keying rhythm, a signature known as "The Fist of the Sender".[12] In the digital age, this principle is applied by analyzing the precise timing of key-press and key-release events.

Feature Extraction

To build a biometric profile, a system measures a variety of timing and frequency features that are statistically unique to an individual's typing pattern.[12] These features can be broadly categorized as:

- **Static Features:** These are timing characteristics associated with individual keys or short sequences. Key features include **dwell time** (the duration a single key is held down) and **flight time** (the time elapsed between releasing one key and pressing the next).[12] The timing of specific, common n-grams (sequences of characters) like `for`, `ing`, or `the` in comments can also be highly characteristic and stable for an individual.[12]
- **Dynamic Features:** These features capture broader characteristics of the user's typing style over time. They include overall typing speed, frequency of errors, and the characteristic use of navigational and editing keys like backspace, delete, and the arrow

keys.[15] These features are statistical in nature and are often modeled using machine learning techniques like artificial neural networks to create a robust biometric template.[12]

Application: Continuous Passive Authentication (CPA)

The most powerful application of keystroke dynamics in an educational context is Continuous Passive Authentication (CPA), a security method that verifies a user's identity seamlessly and constantly throughout a session.[16]

- **Principle:** Unlike active authentication, which requires a discrete user action like entering a password or responding to a two-factor authentication (2FA) prompt, CPA operates invisibly in the background.[16] It provides real-time, continuous verification rather than a single check at the point of login.
- **Mechanism:** The process begins with an enrollment period, during which the system builds a baseline biometric template of the user's normal typing rhythm as they complete initial assignments.[14] Once this profile is established, the system continuously monitors their real-time typing patterns and compares them against the template. This comparison generates a dynamic risk score. If the live typing pattern deviates significantly from the established profile—for instance, if another person with a different typing rhythm takes over the keyboard—the risk score will exceed a predefined threshold.[16] This can trigger an automated response, such as flagging the session for review, requiring active re-authentication, or even locking the assignment or exam.
- **Educational Relevance:** CPA is a potent tool for ensuring academic integrity in high-stakes online assessments. It directly addresses the risk of "ringer" scenarios, where one student logs into an exam and a more proficient individual completes the work on their behalf. By continuously verifying that the person typing is the enrolled student, it provides a level of security that is impossible to achieve with login-based authentication alone.

Section 2.2: Multimodal Behavioral Signatures

While powerful, authentication models based solely on keystroke dynamics can be brittle. A user's typing rhythm can be affected by factors like fatigue, posture, or the keyboard itself. To create a more robust and multifaceted behavioral profile, the data stream can be enriched with other interaction modalities, most notably mouse dynamics.

Integrating Mouse Dynamics

Just as individuals have unique typing rhythms, they also have characteristic patterns of mouse usage. By integrating mouse interaction data, the system can build a more resilient and accurate user profile.[18]

- **Mouse Features:** A wide range of features can be extracted from mouse data, including average movement speed, total travel distance for a given action, trajectory curvature and straightness, the duration of pauses and hovers over specific UI elements, patterns of left and right clicks, and scrolling behavior (speed and amplitude).[18]
- **Combined Models:** Research has demonstrated that models combining both keystroke and mouse dynamics achieve significantly higher authentication accuracy.[19] The two modalities are complementary; a user might be typing very little while navigating code with the mouse, or vice-versa. A multimodal system can maintain a continuous stream of biometric data across these different phases of the programming process.

Beyond Authentication: Profiling Cognitive and Affective States

Herein lies the crucial pivot from a security application to a pedagogical one. The same behavioral signatures used to verify identity can also provide a window into the user's internal state. Deviations from a user's baseline behavior are not always indicative of an imposter; more often, in an educational context, they are signs of cognitive or emotional distress.

- **Cognitive Load:** Studies suggest a strong correlation between mouse movement patterns and cognitive load. When faced with a difficult problem or decision uncertainty, users tend to exhibit more random, less direct mouse movements and longer hover times.[18] These patterns can serve as a behavioral indicator that a student is experiencing a high cognitive load, struggling to process the information required to solve a problem.
- **Stress and Frustration:** Time pressure and task difficulty are known stressors for programmers. Research has shown that these stressors manifest in observable changes to interaction patterns. Under stress, individuals tend to increase their use of the backspace and delete keys, exhibit changes in the length and frequency of pauses between typing bursts, and, on pressure-sensitive hardware, apply more physical force to the keys.[15] These signals can be automatically detected as indicators of student frustration.
- **Emotional State:** The link between interaction patterns and emotion extends even to more general states. Studies have successfully used keystroke dynamics to differentiate

between text written with a positive versus a negative sentiment, suggesting that our emotional valence subtly influences our typing rhythm.[20]

This dual-use nature of behavioral biometrics is the central architectural principle of a truly intelligent educational IDE. A single, rich data stream can feed two distinct analytical modules: a "proctor" module focused on identity and a "tutor" module focused on cognitive state. The system's challenge, and its ultimate value, lies in its ability to disambiguate these signals through context. A sudden change in typing rhythm during a final exam might trigger a security alert. That exact same change during a formative homework assignment should instead be interpreted as a potential "struggle" signal, triggering a pedagogical intervention.

Furthermore, this approach offers a practical and scalable path toward affective computing in education. Many attempts to make educational software "emotion-aware" have relied on highly intrusive and often impractical technologies like webcam-based facial expression analysis or wearable physiological sensors.[21] These methods face significant privacy objections from students and are difficult to deploy at scale. Keystroke and mouse dynamics, by contrast, provide a non-intrusive, hardware-agnostic proxy for detecting key affective states like stress and frustration.[15] This allows a system to become sensitive to a student's emotional state without requiring them to turn on a camera or wear a sensor, overcoming one of the largest barriers to the adoption of affective computing in mainstream educational tools.[24]

Part III: Modeling the Learner: Applying AI to Detect Cognitive and Affective States

With a high-fidelity data stream established (Part I) and a theoretical framework for interpreting behavioral signatures (Part II), the next step is to apply advanced machine learning techniques to automatically and predictively model the learner's state. The goal is to move beyond simple metrics and develop a system that can identify, in real-time, when a student is experiencing cognitive struggle. This requires a deep understanding of both the psychology of novice programmers and the capabilities of modern AI architectures.

Section 3.1: Identifying Behavioral Correlates of Cognitive Struggle

To build a model that detects struggle, we must first define what struggle looks like in terms of

observable behavior. This involves synthesizing insights from cognitive psychology, computer science education research, and cognitive load theory.

The Psychology of the Novice Programmer

Programming is an exceptionally demanding cognitive activity. It requires abstract reasoning, symbolic interpretation, and places a heavy load on both working memory (for tracking variables and logic flow) and long-term memory (for recalling syntax and design patterns).[25] Novice programmers, in particular, often lack the well-developed metacognitive strategies that experts use to manage this complexity.[27] They struggle with planning, monitoring their own understanding, and systematically debugging their code. This often leads to a reliance on inefficient and frustrating "trial-and-error" methods, where they become preoccupied with fixing superficial syntax errors without addressing the underlying logical flaws in their approach.[27]

Observable Behavioral Patterns of Struggle

This internal cognitive and metacognitive difficulty manifests in a set of observable behavioral patterns within the IDE. An analytics system can be trained to recognize these patterns as indicators of struggle:

- **Debugging Patterns:** Novice debugging is often unsystematic. Instead of forming a hypothesis and testing it, they may resort to reading the code repeatedly, making random changes in the hope of a fix, and inadvertently introducing new bugs in the process.[29] They also have difficulty distinguishing between different error types—syntax errors from the compiler, runtime errors during execution, and logic errors where the program runs but produces the wrong output—and applying the correct strategy for each.[31]
- **Code Edit Patterns:** The editing process itself contains strong signals of struggle. These include spending an excessive amount of time or making an excessive number of attempts on a single problem with little to no progress, a state often described as being "stuck".[34] Other indicators are a high frequency of compilation events that consistently result in errors [32], a pattern of rapid, small edits followed immediately by a compile-fail cycle (sometimes called "flailing"), and high volatility in their interaction with learning materials, which can be measured as high Shannon entropy in their navigation patterns.[35]

Cognitive Load Theory (CLT) in Programming

Cognitive Load Theory provides the theoretical lens through which to interpret these behaviors. CLT posits that human working memory is severely limited, and learning is hampered when the mental effort required to process information—the cognitive load—exceeds this capacity.[36] CLT distinguishes between three types of load:

1. **Intrinsic Load:** The inherent complexity of the material itself. Programming has a very high intrinsic load due to its high "element interactivity"—many different concepts (variables, loops, conditionals, data structures) must be held and manipulated in working memory simultaneously to be understood.[36]
2. **Extraneous Load:** The load imposed by the way information is presented. Poorly designed instruction, confusing interfaces, or unclear problem descriptions increase extraneous load, consuming precious working memory without contributing to learning.[39]
3. **Germane Load:** The effort dedicated to processing information and constructing long-term knowledge schemas. This is the "good" cognitive load associated with deep learning.[39]

From this perspective, the behavioral patterns of struggle identified above are the external manifestations of cognitive overload. They are the actions a student takes when the combined intrinsic and extraneous load of a programming task has overwhelmed their working memory capacity, leaving insufficient resources for the germane load required for learning.[36] The primary goal of a predictive analytics module is therefore to detect these behavioral signals of cognitive overload in real-time.

Section 3.2: Predictive Modeling with Sequential Interaction Data

The raw data stream of every keystroke and mouse movement is too granular and noisy to be used directly in a predictive model. The crucial first step is feature engineering, followed by the application of a machine learning model capable of understanding temporal sequences.

Feature Engineering and Selection

Feature engineering is the process of transforming raw data into a set of informative features (or variables) that a machine learning model can use to make predictions.[40] For IDE interaction data, this involves aggregating low-level events into higher-level metrics over

specific time windows.

- **Feature Creation:** Instead of individual keystroke timestamps, we can create features like: typing speed and backspace frequency over the last 30 seconds; the number of compilation events and resulting errors in the last 2 minutes; the average time between edit, compile, and run events; the size and frequency of code chunks being deleted or pasted; and metrics of navigation like the entropy of switching between different files in a project.[15]
- **Feature Selection:** Not all engineered features will be equally predictive. To improve model performance, reduce computational complexity, and prevent overfitting, feature selection techniques are applied. These can be **filter methods** that use statistical tests (e.g., correlation) to rank features independently, or more complex **wrapper** and **embedded methods** that select features based on their contribution to the performance of a specific machine learning model.[45]

Long Short-Term Memory (LSTM) Networks

The programming process is inherently sequential; what a student does now is heavily dependent on what they did minutes ago. Traditional machine learning models struggle to capture these temporal dependencies. This is where Recurrent Neural Networks (RNNs), and specifically Long Short-Term Memory (LSTM) networks, become essential.

- **Why LSTMs?** LSTMs are a special type of RNN designed explicitly to learn long-term dependencies in time-series data.[47] Unlike standard neural networks, LSTMs have internal memory cells and gating mechanisms that allow them to selectively remember information from earlier in a sequence and use that context to inform predictions about later events.[47] This makes them perfectly suited for modeling the sequence of actions that constitute the programming process.
- **Application to Struggle Detection:** In this context, a sequence of the engineered features described above (e.g., a vector of typing speed, error rate, compile frequency, etc., for each 10-second interval over a 5-minute window) is fed into an LSTM model.[49] The model is trained on a large dataset of student sessions that have been labeled with ground-truth states (e.g., "struggling," "making progress," "idle"). The LSTM learns the temporal patterns of behavior that typically precede moments of struggle or breakthrough.[51] Once trained, the model can take a student's recent interaction history as input and output a real-time probability that they are currently in a state of struggle. Hybrid models that combine Convolutional Neural Networks (CNNs) to extract "spatial" features from snapshots of the code itself, with LSTMs to model the temporal evolution of those features, have shown particularly high predictive accuracy.[48]

In a very real sense, the goal of this modeling effort is to create the "digital equivalent of a furrowed brow." A skilled human teacher in a classroom can detect non-verbal cues of struggle—a student staring blankly, sighing in frustration, or anxiously fidgeting.[22] In an online IDE, these physical cues are absent. The behavioral patterns captured by the logger—erratic mouse movements, frantic typing followed by large deletions, rapid and unsuccessful compilation attempts—are the digital body language of cognitive distress. The LSTM model acts as an automated observer, trained to recognize these subtle signals. This reframes the purpose of the analytics system: it is not merely about predicting grades, but about restoring the nuanced observational capability of an expert human tutor in a scalable, online environment, enabling timely and empathetic intervention.

However, a sophisticated system must go one step further. Not all struggle is detrimental to learning. The cognitive effort involved in grappling with a difficult concept is essential for building robust mental models (i.e., germane cognitive load).[39] The critical challenge is to differentiate this

productive struggle from **unproductive struggle**, or "flailing".[34] A student systematically using

print statements to trace a variable's value is engaged in productive debugging. A student randomly changing operators and recompiling is flailing. Simple metrics like "time on task" cannot distinguish between these two states. A sequential model like an LSTM, however, can learn to recognize the distinct temporal signatures of these different behaviors. The success of any automated intervention system hinges on this distinction. Intervening during productive struggle can disrupt a student's concentration and undermine their learning process. Therefore, the model must be trained not just to detect failure, but to recognize the specific patterns of behavior that lead to unproductive roadblocks, allowing it to target interventions with precision and pedagogical intelligence.

The following table provides a conceptual mapping between inferred cognitive/affective states and the behavioral indicators that a machine learning model would use to detect them.

Inferred State	Keystroke Dynamics Features	Code Edit Features	Mouse Dynamics Features	Session Features
Flow / Engagement	Consistent, fast typing speed; low error rate; rhythmic	Long, focused coding sessions between compiles; successful	Efficient, direct mouse movements; minimal unnecessary	Linear navigation through assignment files; focused on relevant

	pauses.	compilations; systematic progression.	hovering.	resources.
Productive Struggle	Slower typing; longer pauses for thought; increased use of navigation keys.	Systematic edit-compile-debug cycles; use of print statements or debugger tools; incremental progress.	Deliberate movements; hovering over specific code sections or error messages.	Focused switching between code, documentation, and error outputs.
Unproductive Struggle / Flailing	Erratic typing speed; high frequency of backspace/delete; frantic bursts of typing.	Very short, rapid edit-compile cycles with persistent errors; random code changes; large deletions of recent work.	Erratic, high-velocity mouse movements; excessive, non-purposeful scrolling.	Rapid, unfocused switching between many files; repeated viewing of same help resource without progress.
Disengagement	Long periods of no keyboard activity.	No compilation or execution events for extended periods.	No mouse activity or minor, repetitive movements.	Idle on a single page; navigation to off-task websites (if tracked).
Plagiarism Event	Abrupt cessation of typing followed by a large paste event; sudden, dramatic change in typing speed	A large block of complex, error-free code appears instantly.	Minimal mouse movement during the paste event.	Navigation to external websites like Stack Overflow or Chegg immediately preceding a paste.

	and accuracy.			

Part IV: Pedagogical and Institutional Applications

The ability to accurately model a learner's cognitive and affective state in real-time is not an end in itself. Its true value lies in enabling a new generation of pedagogical tools and institutional practices that are more responsive, effective, and secure. This section explores the concrete applications of behavioral learning analytics, from revolutionizing academic integrity to powering truly personalized tutoring and adaptive learning environments.

Section 4.1: A New Paradigm for Academic Integrity

Traditional approaches to academic integrity in programming have focused on analyzing the final submitted product. Tools like MOSS and JPlag perform static analysis to detect similarities between student submissions.[10] While useful, these methods have significant limitations. They can be defeated by simple obfuscation techniques (e.g., renaming variables, reordering functions) and are increasingly ineffective against code generated by Large Language Models (LLMs), which can produce functionally similar but textually unique solutions for each student.[10] Behavioral analytics enables a paradigm shift from product analysis to process forensics.

- **Forensic Process Analysis:** Instead of asking "Is this code similar to another submission?", this new approach asks "How was this code created?". The high-fidelity interaction logs provide a complete, auditable record of the development process.
    - **Code Playback:** Platforms like Codio offer a "Code Playback" feature that allows an instructor to visually replay the entire history of a code file, keystroke by keystroke.[10] An instructor can watch the student's solution evolve over time. A large, complex block of code appearing instantaneously in the editor is an unmistakable sign of a paste event, providing a powerful and intuitive red flag for potential misconduct.
    - **Paste and Authorship Tracking:** More advanced systems, such as the research tool PasteTrace, implement sophisticated authorship tracking.[4] When code is copied from within the specialized IDE, the system injects invisible, zero-width space characters into the copied text. These characters encode metadata, including a unique ID for the user and the project (InstallID and ProjectID). When this text is pasted into another project, the IDE detects and decodes this metadata, logging the precise origin of the pasted code. This creates an irrefutable, forensic link between submissions, allowing

an instructor to build a complete graph that traces the flow of code from one student to another. This method is fundamentally immune to code similarity analysis because it tracks the
act of copying, not the content of the code itself.[4]

- **Behavioral Signature Analysis:** The continuous biometric profiles discussed in Part II provide another layer of evidence. A sudden and dramatic shift in typing proficiency—for example, from a slow, error-prone style to a rapid, expert style within the same file—is a strong behavioral indicator that a different person has taken over the keyboard.[14]

This process-oriented approach represents a potential end to the perpetual cat-and-mouse game of plagiarism detection. For decades, academic integrity has been an arms race: students develop new methods of cheating, and institutions develop new methods of detection. Forensic analysis of the creation process is a disruptive shift because it is largely agnostic to the *content* of the plagiarized material. It does not matter if the code was copied from a classmate, a contract cheating service, or a generative AI. If the code was not physically typed into the IDE by the authenticated student, the system flags the anomaly. This has profound implications for institutional policy, potentially rendering traditional similarity checkers obsolete and driving the adoption of monitored, authenticated development environments for all graded programming work.

Section 4.2: Powering Intelligent Tutoring and Adaptive Scaffolding

The most significant pedagogical application of struggle detection is its ability to serve as a trigger for proactive, personalized student support. Instead of a reactive model where students must recognize their own confusion and actively seek help, the system can identify the moment of need and offer assistance in real-time.[54]

- **Intelligent Tutoring Systems (ITS):** An ITS is a system that uses AI to provide customized instruction and feedback, mimicking a human tutor.[56] When the struggle detection model flags a student, the ITS can initiate an intervention.
  - **Personalized, Real-Time Feedback:** The nature of the feedback can be tailored to the context. If the system detects a common syntax error, it might provide a direct hint or a link to relevant documentation.[58] If it detects a pattern indicative of a logic error, it might engage the student in a Socratic dialogue, asking questions that guide them to discover the flaw in their own reasoning, a process known to be more effective for deep learning.[57] This immediate feedback is crucial for maintaining student motivation and preventing the frustration that leads to disengagement.[58]
  - **Automated Program Repair:** In some cases, the ITS can go beyond providing hints and leverage automated program repair techniques.[62] For certain classes of

well-understood bugs, the system can automatically generate a correct code patch and suggest it to the student. This can be invaluable for helping novices overcome frustrating implementation hurdles, allowing them to focus on higher-level problem-solving concepts.[62]

- **Adaptive Scaffolding:** Scaffolding in education refers to temporary support structures that allow a learner to accomplish a task that would otherwise be beyond their capability.[65] An adaptive system uses the real-time learner model to dynamically adjust the level of scaffolding provided.
  - **Dynamic Support and Fading:** When the system detects that a student is struggling, it can increase the level of support—for example, by providing more explicit hints, breaking the problem down into smaller sub-tasks, or offering worked examples.[67] Crucially, as the system observes the student's proficiency increasing (e.g., fewer errors, more systematic debugging patterns), it begins to "fade" the scaffolding, gradually withdrawing support and giving the learner more autonomy.[66] This dynamic adjustment ensures that each student receives the precise level of support they need at each moment, creating a truly individualized learning path.[70]

These components—struggle detection, intelligent tutoring, and adaptive scaffolding—can be combined to create a complete, automated pedagogical feedback loop. The system (1) **senses** a behavioral signature of unproductive struggle using its LSTM model; (2) **responds** with a targeted intervention, such as a hint from the ITS; (3) continues to **monitor** the student's subsequent behavior to assess the intervention's effectiveness; and (4) **adapts** its next action based on that assessment, either offering more help or beginning to fade the scaffolds. This sense-respond-adapt cycle mimics the core process of expert one-on-one human tutoring and represents the ultimate goal of AI in education: providing personalized, scalable support to every learner.

Section 4.3: Advanced Learning Analytics Dashboards (LADs)

The vast amount of behavioral data collected can be visualized to provide powerful insights for both instructors and students, moving far beyond the simple gradebooks and activity logs of current LADs.[71]

- **For Instructors:** A dashboard could provide an aggregated, class-level view of the learning process. An instructor could see "heat maps" showing which parts of an assignment are causing the most difficulty for the most students (i.e., "struggle hotspots").[72] They could identify the most common misconceptions or error patterns across the entire class, allowing them to adjust their next lecture to address these specific issues. The system could also allow them to compare the problem-solving trajectories of students who succeeded with those who struggled, revealing key

differences in strategy that can inform their teaching.[73]

- **For Students (Self-Regulated Learning):** Dashboards can also be powerful tools for metacognition, helping students to reflect on and improve their own learning processes.[71] A student could view a personalized dashboard that visualizes their own work patterns. For example, a timeline could show them how much time they spent in a state of "productive struggle" versus "unproductive flailing," encouraging them to adopt more systematic debugging strategies.[27] By making their own learning process visible, these tools can empower students to become more effective, self-regulated learners.

Part V: The Ethical Imperative: A Framework for Responsible Implementation

The technologies described in this report hold immense promise for transforming programming education. However, their power is matched by their potential for misuse. The act of continuously monitoring and analyzing every detail of a student's interaction with their learning environment is a form of surveillance, and without a robust ethical and legal framework, such systems risk doing more harm than good. This final section provides a critical analysis of these challenges and proposes a framework for responsible implementation.

Section 5.1: Navigating the Surveillance Dilemma

The deployment of fine-grained monitoring tools in education creates a fundamental tension between the goal of providing support and the risk of creating an oppressive learning environment.

- **The "Chilling Effect" on Learning:** Constant, granular monitoring can have a "chilling effect" on student behavior.[74] Learning to program is an iterative process that inherently involves making mistakes, experimenting with different approaches, and exploring inefficient or "wrong" paths. When students know that their every action is being logged and analyzed, they may become risk-averse, sticking only to safe, prescribed methods and avoiding the creative exploration that leads to deep understanding.[75] The surveillance can thus undermine the very intellectual curiosity and resilience that programming education aims to foster.[77]
- **Erosion of Trust:** A healthy learning environment is built on a foundation of trust between students, instructors, and the institution. Pervasive surveillance can erode this trust, making students feel that they are presumed to be cheating or incompetent.[74] This

can lead to disengagement and an adversarial relationship with the learning platform, which comes to be seen as a tool of judgment rather than support. Research shows that while schools often justify surveillance on the grounds of safety and security, the evidence for its effectiveness is thin, and it often leads to unintended negative consequences, including a *decreased* sense of safety and trust among students.[75]

- **The Problem of Data Misinterpretation:** Automated systems are inherently brittle and lack the nuanced understanding of a human expert. An algorithm might flag a student as "disengaged" when they are, in fact, deeply engaged in off-screen thinking, sketching out a solution on paper. A long pause may not be a sign of confusion, but a moment of reflective thought.[73] An automated intervention triggered at the wrong moment can be disruptive and counterproductive, breaking a student's state of "flow" and causing frustration.[25] The pedagogical clumsiness of a system that cannot distinguish deep thought from disengagement is a significant risk.

This creates the central **paradox of proctoring**: the very mechanism used to detect struggle (surveillance) can create an environment of anxiety and distrust that inhibits learning. The "proctor" function and the "tutor" function are in direct philosophical conflict. Resolving this paradox is the single most important challenge for the successful implementation of this technology. The system's design and policies must be overwhelmingly biased toward the "tutor" role. The "proctor" function, such as flagging for academic dishonesty, must be a rare exception triggered only by a high threshold of evidence, not the default mode of operation.

Section 5.2: A Blueprint for Ethical Data Governance

A commitment to ethical practice cannot be an afterthought; it must be designed into the system's architecture and the institution's policies from the very beginning. This requires navigating a complex legal landscape and adhering to a set of core principles for data governance.

Legal Frameworks

In the United States, several federal laws govern the handling of student data. Any platform implementing behavioral analytics must ensure strict compliance.

- **FERPA (Family Educational Rights and Privacy Act):** The detailed interaction data collected by the IDE unequivocally constitutes an "education record" under FERPA and is

subject to its protections.[81] This means institutions must obtain consent for its collection and disclosure. Third-party vendors typically operate under FERPA's "school official" exception, which allows them to handle student data but strictly limits them to using it only for the educational purposes for which they were contracted and prohibits any secondary use, such as for advertising.[82]

- **COPPA (Children's Online Privacy Protection Act):** If the platform is used in K-12 settings with students under the age of 13, COPPA imposes even stricter requirements, including verifiable parental consent before any personal information is collected.[82]
- **State and International Laws:** Many states have enacted their own student data privacy laws, such as California's Student Online Personal Information Protection Act (SOPIPA), which are often more stringent than federal law.[82] Furthermore, for institutions with a global student body, regulations like Europe's GDPR must be considered, which grant students extensive rights over their data.

Best Practices for Implementation

Legal compliance is the floor, not the ceiling. An ethically sound implementation must be built on a foundation of best practices designed to protect students and build trust.[8]

- **Radical Transparency:** Students and their families must be clearly, explicitly, and proactively informed about what data is being collected, how it is being analyzed, and for what specific pedagogical purposes.[21] This information must be presented in an easily understandable format, not buried within a lengthy terms of service document that students are known to ignore.[2]
- **Student Agency and Control:** Wherever feasible, students should be given meaningful control over their data. This includes the right to access their own data and analytics, to understand how the system has classified their behavior, and, where appropriate, to opt-out of certain types of data collection or analysis.[21] Providing students with their own dashboards is a key part of this, turning the system from a tool of surveillance into a tool for self-reflection.[85]
- **Privacy-Preserving Technologies:** Institutions should invest in and demand privacy-enhancing technologies (PETs). For example, research into homomorphic encryption could allow for the analysis of keystroke dynamics on encrypted data, enabling the authentication server to verify a user's typing rhythm without ever having access to the unencrypted timing patterns themselves.[17] By default, data should be anonymized and aggregated whenever possible to protect individual student identities.[86]
- **Purpose Limitation and Human-in-the-Loop:** There must be strict technical and policy firewalls preventing data collected for pedagogical support from being used for unrelated purposes, such as marketing or non-academic disciplinary actions.[21]

Furthermore, high-stakes decisions, especially those related to academic integrity, should never be fully automated. The system should flag potential issues for review, but the final judgment must be made by a human instructor who can apply context and nuance.

Finally, it is imperative to address the risk of **algorithmic bias**. The machine learning models that detect struggle are trained on historical student data.[51] If that data reflects existing societal or educational biases, the model will learn and amplify them. For example, if students from under-resourced high schools tend to have less prior programming experience, they may exhibit behavioral patterns that the model learns to associate with "struggle." The system could then disproportionately flag these students, leading to stigmatization or a self-fulfilling prophecy. Similarly, the model might fail to recognize the valid but atypical problem-solving strategies of neurodivergent students, misclassifying their behavior as "flailing." Therefore, regular, rigorous audits of the models for fairness and bias across all demographic, socioeconomic, and neurodiverse groups are not an optional extra; they are an absolute ethical necessity to ensure the system supports equity rather than undermining it.

The following table provides a framework for addressing the key ethical risks associated with behavioral learning analytics through a combination of technical, policy, and pedagogical mitigation strategies.

Identified Risk	Technical Mitigation	Policy Mitigation	Pedagogical Mitigation
Student Privacy Violation	Data anonymization/aggregation by default; Encryption in transit and at rest; Privacy-Enhancing Technologies (e.g., homomorphic encryption).	Strict adherence to FERPA/GDPR; Clear, accessible data privacy policies; Purpose limitation clauses in vendor contracts.	"Human-in-the-loop" for all sensitive data access; Training for instructors on data privacy best practices.
Algorithmic Bias	Use of fairness-aware machine learning algorithms; Regular bias audits of models and training data across demographic	Diverse and representative data collection protocols; Establishment of an independent ethics review board.	Supplement algorithmic flags with qualitative instructor judgment; Solicit student feedback on model fairness

	groups.		and accuracy.
"Chilling Effect" on Creativity	Focus models on detecting unproductive "flailing" rather than all forms of struggle; Anonymize data before analysis.	Policy of not using data for grading effort or penalizing experimentation; Explicitly communicate that exploration and mistakes are encouraged.	Frame system as a supportive, non-judgmental "coding companion"; Use interventions to encourage creative problem-solving.
Erosion of Trust	Provide students with a personal dashboard to view and understand their own data; Implement opt-out options where possible.	Radical transparency about all data collection and analysis; Co-design of policies with student and faculty representatives.	Emphasize the "tutor" function over the "proctor" function; Ensure interventions are helpful and empathetic, not punitive.
Data Misinterpretation	Design dashboards that provide context and avoid simplistic labels; Integrate confidence scores with predictions.	Mandate a "human-in-the-loop" for all high-stakes decisions; Establish clear protocols for interpreting and acting on data.	Train instructors to treat analytics as one data point among many, not as an absolute truth; Encourage dialogue with students about flagged events.

Conclusion and Future Directions

The integration of behavioral learning analytics into programming IDEs represents a genuine technological and pedagogical leap forward. By moving from a post-hoc analysis of student artifacts to a real-time understanding of the learning process itself, we can build educational systems that are more supportive, adaptive, and effective than ever before. The ability to provide personalized, just-in-time feedback, to dynamically scaffold learning for individual

needs, and to ensure academic integrity with a new level of forensic rigor has the potential to address some of the most persistent challenges in computer science education.

However, this report has also demonstrated that this potential is inextricably linked to profound ethical responsibilities. The power to monitor is the power to control, and the line between a supportive tutor and an oppressive proctor is dangerously thin. The success of this entire paradigm is not a technical question; it is an ethical one. It depends on our collective ability to design and deploy these systems within a framework of radical transparency, student agency, and unwavering respect for privacy. The technology must always serve the pedagogy, and the pedagogy must always serve the student.

Looking forward, the evolution of this field will likely focus on several key areas. The integration of more data modalities, such as voice commands or gaze-tracking within the IDE, could provide an even richer picture of the learner's cognitive state. The analytical models will become more sophisticated, moving beyond simple struggle detection to provide feedback on higher-order skills like problem decomposition, strategic planning, and algorithmic thinking. Finally, the ongoing co-evolution of these systems with generative AI code assistants will present new challenges and opportunities, requiring analytics to distinguish between effective use of an AI partner and over-reliance on it. Navigating this future will require a continuous dialogue between technologists, educators, policymakers, and students to ensure that as our tools become more powerful, they also become more wise.

Works cited

1. Keystroke logging - Wikipedia, accessed August 25, 2025, https://en.wikipedia.org/wiki/Keystroke_logging
2. What is Keystroke Logging and Keyloggers? - Kaspersky, accessed August 25, 2025, https://www.kaspersky.com/resource-center/definitions/keylogger
3. Copy and paste | Views - Android Developers, accessed August 25, 2025, https://developer.android.com/develop/ui/views/touch-and-input/copy-paste
4. PasteTrace: A Single Source Plagiarism Detection Tool For ... - arXiv, accessed August 25, 2025, https://arxiv.org/pdf/2506.17355
5. Solved: Copy/Paste Event Strucutre - NI Community - National Instruments, accessed August 25, 2025, https://forums.ni.com/t5/LabVIEW/Copy-Paste-Event-Strucutre/td-p/1424466
6. The Best 7 Session Replay Tools in 2025 - Statsig, accessed August 25, 2025, https://www.statsig.com/comparison/best-session-replay-tools
7. Session Replay: How It Works & Use Cases | Datadog, accessed August 25, 2025, https://www.datadoghq.com/knowledge-center/session-replay/
8. An interactive Web-based IDE towards teaching and learning in ..., accessed August 25, 2025, https://www.researchgate.net/publication/261156315_An_interactive_Web-based_IDE_towards_teaching_and_learning_in_programming_courses
9. Codio | Hands-On Learning Experience Platform | Data, AI, Cyber, Dev, accessed August 25, 2025, https://www.codio.com/

10. Proactive Cheating Prevention | Go Beyond Plagiarism Detection ..., accessed August 25, 2025, https://www.codio.com/solutions/cheating-prevention
11. Canvas LMS: #1 in North America for Accessible, Flexible Learning | Instructure, accessed August 25, 2025, https://www.instructure.com/canvas
12. Keystroke dynamics - Wikipedia, accessed August 25, 2025, https://en.wikipedia.org/wiki/Keystroke_dynamics
13. Keystroke Dynamics for User Authentication and Identification by using Typing Rhythm, accessed August 25, 2025, https://www.researchgate.net/publication/304066436_Keystroke_Dynamics_for_User_Authentication_and_Identification_by_using_Typing_Rhythm
14. Keystroke Dynamics: Concepts, Techniques, and Applications - arXiv, accessed August 25, 2025, https://arxiv.org/html/2303.04605v2
15. Towards detecting programmers' stress on the basis of keystroke dynamics - Annals of Computer Science and Information Systems, accessed August 25, 2025, https://annals-csis.org/Volume_8/pliks/263.pdf
16. Continuous Passive Authentication explained - Corbado, accessed August 25, 2025, https://www.corbado.com/blog/continuous-passive-authentication
17. [2209.06557] A Generic Privacy-Preserving Protocol For Keystroke Dynamics-Based Continuous Authentication - arXiv, accessed August 25, 2025, https://arxiv.org/abs/2209.06557
18. Mouse Behavioral Patterns and Keystroke Dynamics in End-User Development: what can they tell us about users' behavioral attributes? | Request PDF - ResearchGate, accessed August 25, 2025, https://www.researchgate.net/publication/323167479_Mouse_Behavioral_Patterns_and_Keystroke_Dynamics_in_End-User_Development_what_can_they_tell_us_about_users'_behavioral_attributes
19. User Authentication Method Based on Keystroke Dynamics and Mouse Dynamics with Scene-Irrelated Features in Hybrid Scenes - PMC, accessed August 25, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC9460698/
20. Keystroke Dynamics Patterns While Writing Positive and Negative Opinions - PMC, accessed August 25, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC8434638/
21. Ethical use of student engagement data in online education, accessed August 25, 2025, https://swisscyberinstitute.com/blog/ethical-use-student-engagement-data-online-education/
22. Students' Ethical, Privacy, Design, and Cultural Perspectives on Visualizing Cognitive-Affective States in Online Learning - ERIC, accessed August 25, 2025, https://files.eric.ed.gov/fulltext/EJ1456255.pdf
23. Automatic modeling of student characteristics with interaction and physiological data using machine learning: A review - Frontiers, accessed August 25, 2025, https://www.frontiersin.org/journals/artificial-intelligence/articles/10.3389/frai.2022.1015660/full
24. Affective Computing for Learning in Education: A Systematic Review and Bibliometric Analysis - MDPI, accessed August 25, 2025,

https://www.mdpi.com/2227-7102/15/1/65
25. The Psychology Behind Being a Computer Programmer - Philip Zimbardo, accessed August 25, 2025, https://www.zimbardo.com/the-psychology-behind-being-a-computer-programmer/
26. The Mind Behind the Code: Exploring the Psychology of Programming - Technorely, accessed August 25, 2025, https://technorely.com/insights/the-mind-behind-the-code-exploring-the-psychology-of-programming
27. Examining Metacognitive Difficulties in Learning Programming: Analysis of Student Behavior and Strategy - ResearchGate, accessed August 25, 2025, https://www.researchgate.net/publication/394050481_Examining_Metacognitive_Difficulties_in_Learning_Programming_Analysis_of_Student_Behavior_and_Strategy
28. (PDF) Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review - ResearchGate, accessed August 25, 2025, https://www.researchgate.net/publication/320679287_Students'_Misconceptions_and_Other_Difficulties_in_Introductory_Programming_A_Literature_Review
29. Software Debugging Patterns for Novice Programmers - The Hillside Group, accessed August 25, 2025, https://hillside.net/asianplop/proceedings/AsianPLoP2017/papers/15.pdf
30. A Video-Based Approach to Learning Debugging Techniques - CEUR-WS, accessed August 25, 2025, https://ceur-ws.org/Vol-3806/S_18_Shynkarenko_Zhevaho.pdf
31. An Empirical Study of Debugging Patterns Among Novices Programmers - ResearchGate, accessed August 25, 2025, https://www.researchgate.net/publication/314294332_An_Empirical_Study_of_Debugging_Patterns_Among_Novices_Programmers
32. An analysis of patterns of debugging among novice computer science students, accessed August 25, 2025, https://www.researchgate.net/publication/220807559_An_analysis_of_patterns_of_debugging_among_novice_computer_science_students
33. [2410.21282] Logic Error Localization in Student Programming Assignments Using Pseudocode and Graph Neural Networks - arXiv, accessed August 25, 2025, https://arxiv.org/abs/2410.21282
34. Detecting Students of Concern in Introductory Programming Classes: Techniques to Indicate Potential Struggle or Cheating - eScholarship, accessed August 25, 2025, https://escholarship.org/content/qt4xr6k0d4/qt4xr6k0d4_noSplash_3caf5af4895ea3f41db55d60733dc31c.pdf
35. Students' Learning Behaviour in Programming Education Analysis: Insights from Entropy and Community Detection - PMC, accessed August 25, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC10453761/
36. Cognitive Load Theory in Computing Education Research: A Review - ResearchGate, accessed August 25, 2025,

https://www.researchgate.net/publication/360278282_Cognitive_Load_Theory_in_Computing_Education_Research_A_Review
37. pmc.ncbi.nlm.nih.gov, accessed August 25, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC11852728/#:~:text=Cognitive%20Load%20Theory%20(CLT)%20was,working%20memory%20has%20limited%20capacity.
38. (PDF) Applying Cognitive Load Theory to Computer Science Education - ResearchGate, accessed August 25, 2025, https://www.researchgate.net/publication/250790986_Applying_Cognitive_Load_Theory_to_Computer_Science_Education
39. Challenging Cognitive Load Theory: The Role of Educational Neuroscience and Artificial Intelligence in Redefining Learning Efficacy - PMC - PubMed Central, accessed August 25, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC11852728/
40. Feature Extraction in Machine Learning: A Complete Guide - DataCamp, accessed August 25, 2025, https://www.datacamp.com/tutorial/feature-extraction-machine-learning
41. What is Feature Engineering? - Machine Learning - GeeksforGeeks, accessed August 25, 2025, https://www.geeksforgeeks.org/machine-learning/what-is-feature-engineering/
42. Role of convolutional features and machine learning for predicting student academic performance from MOODLE data, accessed August 25, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC10631652/
43. Kiro AI: A Guide With Practical Examples - DataCamp, accessed August 25, 2025, https://www.datacamp.com/tutorial/kiro-ai
44. Empirical Evaluation of Prompting Strategies for Python Syntax Error Detection with LLMs, accessed August 25, 2025, https://www.mdpi.com/2076-3417/15/16/9223
45. Leveraging Feature Extraction to Perform Time-Efficient Selection for Machine Learning Applications - MDPI, accessed August 25, 2025, https://www.mdpi.com/2076-3417/15/15/8196
46. Feature Selection Techniques in Machine Learning - GeeksforGeeks, accessed August 25, 2025, https://www.geeksforgeeks.org/machine-learning/feature-selection-techniques-in-machine-learning/
47. Top 5 Must-Know LSTM Techniques: Memory Networks Explained - Number Analytics, accessed August 25, 2025, https://www.numberanalytics.com/blog/top-must-know-lstm-techniques
48. iCNN-LSTM: A batch-based incremental ransomware detection system using Sysmon, accessed August 25, 2025, https://arxiv.org/html/2501.01083v1
49. A Deep Learning Approach Towards Student Performance Prediction in Online Courses: Challenges Based on a Global Perspective - arXiv, accessed August 25, 2025, https://arxiv.org/html/2402.01655v1
50. A Deep Learning Approach Towards Student Performance Prediction in Online Courses: Challenges Based on a Global Perspective - arXiv, accessed August 25, 2025, https://arxiv.org/pdf/2402.01655

51. ATTENTION-ENHANCED LSTM MODEL FOR INTRUSION DETECTION IN IMBALANCED NETWORK TRAFFIC DATA - jatit, accessed August 25, 2025, https://www.jatit.org/volumes/Vol103No14/16Vol103No14.pdf
52. Predicting student academic performance using Bi-LSTM: a deep learning framework with SHAP-based interpretability and statistical validation - Frontiers, accessed August 25, 2025, https://www.frontiersin.org/journals/education/articles/10.3389/feduc.2025.1581247/full
53. A Robust Hybrid CNN–LSTM Model for Predicting Student Academic Performance | Request PDF - ResearchGate, accessed August 25, 2025, https://www.researchgate.net/publication/391949395_A_Robust_Hybrid_CNN-LSTM_Model_for_Predicting_Student_Academic_Performance
54. 5 Ways to Save Struggling Students | Element451, accessed August 25, 2025, https://element451.com/blog/5-ways-to-save-struggling-students
55. Resource For Assisting Struggling Learners - Government of New Brunswick, accessed August 25, 2025, https://www2.gnb.ca/content/dam/gnb/Departments/ed/pdf/K12/Inclusion/ResourceForAssistingStrugglingLearners.pdf
56. Advancing Education through Tutoring Systems: A Systematic Literature Review - arXiv, accessed August 25, 2025, https://arxiv.org/html/2503.09748v1
57. Intelligent tutoring system - Wikipedia, accessed August 25, 2025, https://en.wikipedia.org/wiki/Intelligent_tutoring_system
58. An intelligent tutoring system for programming education based on informative tutoring feedback: system development, algorithm design, and empirical study | Request PDF - ResearchGate, accessed August 25, 2025, https://www.researchgate.net/publication/388102880_An_intelligent_tutoring_system_for_programming_education_based_on_informative_tutoring_feedback_system_development_algorithm_design_and_empirical_study
59. Designing an intelligent tutoring system for computer programing in the Pacific - PMC, accessed August 25, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC8727464/
60. The Role of Feedback in Intelligent Tutoring System - ResearchGate, accessed August 25, 2025, https://www.researchgate.net/publication/311788176_The_Role_of_Feedback_in_Intelligent_Tutoring_System
61. 10 Data Points: Intelligent Tutoring Systems Boost Academic Success - Number Analytics, accessed August 25, 2025, https://www.numberanalytics.com/blog/10-data-points-intelligent-tutoring-academic-success
62. Automated Program Repair - Communications of the ACM, accessed August 25, 2025, https://cacm.acm.org/research/automated-program-repair/
63. Automated Program Repair - squaresLab, accessed August 25, 2025, https://squareslab.github.io/materials/legoues-cacm2019.pdf
64. An Evaluation of the Impact of Automated Programming Hints on Performance and Learning, accessed August 25, 2025,

https://www.researchgate.net/publication/334780669_An_Evaluation_of_the_Impact_of_Automated_Programming_Hints_on_Performance_and_Learning
65. (PDF) Scaffolding in technology-enhanced learning environments. Interactive Learning Environments, 15(1), 27-46 - ResearchGate, accessed August 25, 2025, https://www.researchgate.net/publication/240512938_Scaffolding_in_technology-enhanced_learning_environments_Interactive_Learning_Environments_151_27-46
66. The Design of Guide Learner-Adaptive Scaffolding in Interactive Learning Environments, accessed August 25, 2025, https://www.researchgate.net/publication/221516304_The_Design_of_Guide_Learner-Adaptive_Scaffolding_in_Interactive_Learning_Environments
67. Analyzing Adaptive Scaffolds that Help Students Develop Self-Regulated Learning Behaviors, accessed August 25, 2025, https://learninganalytics.upenn.edu/ryanbaker/Munshi_et_al_JCAL_Accepted_Draft_Nov_2022.pdf
68. An Adaptive e-Learning System for Enhancing Learning Performance: Based on Dynamic Scaffolding Theory - Eurasia Journal of Mathematics, Science and Technology Education, accessed August 25, 2025, https://www.ejmste.com/download/an-adaptive-e-learning-system-for-enhancing-learning-performance-based-on-dynamic-scaffolding-theory-5314.pdf
69. Adaptive Scaffolds in Open-Ended Computer-Based Learning Environments, accessed August 25, 2025, https://ir.vanderbilt.edu/items/9278e13f-23c5-4b9d-bbfa-851893cfcdf5/full
70. An Adaptive Scaffolding Framework for Self-Regulated Learning in an Open-Ended Learning Environment - Institutional Repository, accessed August 25, 2025, https://ir.vanderbilt.edu/items/03e07b44-d2a2-4e27-ba1c-ee0b304e9bb5/full
71. A Current Overview of the Use of Learning Analytics Dashboards - ResearchGate, accessed August 25, 2025, https://www.researchgate.net/publication/377326931_A_Current_Overview_of_the_Use_of_Learning_Analytics_Dashboards
72. Educational Data Mining: A Foundational Overview - MDPI, accessed August 25, 2025, https://www.mdpi.com/2673-8392/4/4/108
73. A Learning Analytics Dashboard for Improved Learning Outcomes and Diversity in Programming Classes - SciTePress, accessed August 25, 2025, https://www.scitepress.org/Papers/2024/127350/127350.pdf
74. Schools' online surveillance raises parental concerns over student ..., accessed August 25, 2025, https://www.k12dive.com/news/online-surveillance-parents-raise-concerns-student-learning/689998/
75. Attitudes Toward School-Based Surveillance of Adolescents' Social Media Activity: Convergent Parallel Mixed Methods Survey, accessed August 25, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC10879966/
76. Empowering Student Agency in the Digital Age: The Role of Privacy in EdTech, accessed August 25, 2025, https://www.newamerica.org/education-policy/briefs/empowering-student-agen

cy-in-the-digital-age-the-role-of-privacy-in-edtech/
77. Tucker_Vance Surveillance 2A.indd - ERIC, accessed August 25, 2025, https://files.eric.ed.gov/fulltext/ED582102.pdf
78. New ACLU Report Shines Light on Shadowy EdTech Surveillance Industry and the Dangerous Consequences of Surveillance in Schools, accessed August 25, 2025, https://www.aclu.org/press-releases/new-aclu-report-shines-light-on-shadowy-edtech-surveillance-industry-and-the-dangerous-consequences-of-surveillance-in-schools
79. Student Perspectives on School Surveillance - DiVA portal, accessed August 25, 2025, https://www.diva-portal.org/smash/get/diva2:1329863/FULLTEXT01.pdf
80. The Constant and Expanding Classroom: Surveillance in K-12 Public Schools - Carolina Law Scholarship Repository, accessed August 25, 2025, https://scholarship.law.unc.edu/cgi/viewcontent.cgi?article=6749&context=nclr
81. Understanding FERPA, CIPA and Other K–12 Student Data Privacy Laws, accessed August 25, 2025, https://edtechmagazine.com/k12/article/2022/04/understanding-ferpa-cipa-and-other-k-12-student-data-privacy-laws-perfcon
82. EdTech and Privacy: Navigating a Shifting Regulatory Landscape, accessed August 25, 2025, https://www.mwe.com/insights/edtech-and-privacy-navigating-a-shifting-regulatory-landscape/
83. The Policymaker's Guide to Student Data Privacy, accessed August 25, 2025, https://publicinterestprivacy.org/policymakers-guide-to-student-privacy/
84. Student Data Privacy - CT.gov, accessed August 25, 2025, https://portal.ct.gov/das/ctedtech/commission-for-educational-technology/initiatives/student-data-privacy
85. Student privacy issues in online learning environments (free ..., accessed August 25, 2025, https://www.researchgate.net/publication/348795542_Student_privacy_issues_in_online_learning_environments_free_download_httpslnkdinebfFUjr
86. (PDF) Privacy Issues in Online Learning: A Review of Literature and ..., accessed August 25, 2025, https://www.researchgate.net/publication/391230326_Privacy_Issues_in_Online_Learning_A_Review_of_Literature_and_Suggestions_for_Further_Research
87. Privacy and E-Learning: A Pending Task - MDPI, accessed August 25, 2025, https://www.mdpi.com/2071-1050/13/16/9206