EduCode: An Authenticity-Ensuring Algorithm for AI-Mediated Education

Lei Shi
Collaborator: Tangrui Li

Course: CIS 5603 – Artificial Intelligence

1. Introduction

The recent rise of large language models (LLMs) such as ChatGPT has fundamentally transformed the educational landscape. These tools can assist students in drafting essays, solving problems, and refining their language, creating new opportunities for personalized support and creative expression. However, they also raise critical concerns about academic authenticity, authorship attribution, and the integrity of student learning. School districts and universities have responded with mixed strategies: ranging from outright bans to ad hoc AI-detection tools, yet none of these approaches has proven fully effective or sustainable.

AI-detection systems, for instance, suffer from well-documented limitations. They are often unreliable and susceptible to adversarial evasion, and recent studies have shown they disproportionately flag non-native English writers, leading to potential discrimination. Meanwhile, completely prohibiting AI use risks depriving students of valuable learning aids. Research increasingly supports the view that, when used responsibly, AI tools can enhance student engagement, metacognition, and self-explanation. As such, educators now face a dilemma: how to preserve trust and fairness in academic work without discarding the benefits of LLM-assisted learning.

To address this challenge, we propose **EduCode**, a protocol-based system designed to embed authorship transparency directly into the student workflow. Rather than attempting to identify AI-written text after the fact, EduCode constrains the copy, paste, and typing operations within the learning platform itself, ensuring that any AI-assisted content is transparently sourced and contextually traceable. At its core, EduCode reframes the problem of AI misuse not as one of text classification but of **interaction design** — by shaping the way content moves through the platform, it creates a space where LLMs can be used ethically, with oversight, and without undermining academic trust.

The idea builds on earlier conceptual work in the **Cognitive Authorship Protocol (CAP)**, which aimed to verify student understanding by prompting short, reflective responses after submission. CAP remains the philosophical foundation of EduCode, emphasizing the need to validate the learning process rather than simply evaluate the final product. However, technical limitations, including the complexity of real-time NLP feedback and instructional UI integration, made CAP difficult to implement at scale. This led to the design of EduCode as a lighter-weight, platform-

centered system that achieves similar goals through structural enforcement and traceable authorship data.

This report outlines the motivation, system architecture, implementation details, and experimental results of the EduCode prototype. We also compare our approach to related work in authorship attribution, AI-detection, and trustworthy learning environments.

2. Related Work

2.1 AI-Generated Text Detection Tools and Limitations

A surge of tools for detecting AI-generated student writing (e.g., GPTZero, Turnitin's AI detector) has emerged to uphold academic integrity. However, studies have found that these detectors often exhibit poor reliability and biases. For example, Weber-Wulff *et al.* tested 12 public and commercial detectors (including Turnitin) and found a tendency to misclassify AI outputs as human-written, with overall low accuracy[1]. Similarly, Elkhatat *et al.* reported high false-negative and false-positive rates across detection tools[2]. Moreover, adaptive strategies can easily evade such detectors: Lu *et al.* demonstrate that by paraphrasing through semantic substitutions, LLM-generated text can slip past multiple detectors[3]. These findings underscore the limitations of purely post hoc text analysis for AI misuse detection. In practice, relying solely on final-document classifiers is problematic, motivating exploration of alternative approaches beyond traditional detection algorithms.

2.2 Process-Based Authorship Verification

Rather than analyzing the submitted artifact alone, recent work has proposed verifying authorship by capturing the writing process itself. Kundu *et al.* introduce a keystroke dynamics method to distinguish genuine student writing from AI-assisted writing by analyzing how a text was typed[4]. By training on typing pattern data (e.g., timing, pauses), their system detected AI assistance with promising accuracy, highlighting distinctive process signatures of AI-generated content. Aburass and Abu Rumman take a complementary approach with the "Writer's Integrity" framework, which logs granular writing-process metrics (typing speed, edits, copy-paste ratio) and produces a *certificate of authenticity* attesting that a document was human-authored[5]. This provenance-based approach emphasizes transparency and trust: instructors or reviewers can inspect the step-by-step evolution of a document as evidence of original student work. Such process-focused frameworks align closely with the goals of **EduCode**, which verifies authorship via recorded writing sessions rather than post-submission detection. They offer a preventive deterrent to misconduct by making the creation process traceable.

2.3 Transparent and Guided Use of LLMs in Education

Beyond detection and verification, researchers have called for pedagogical strategies to integrate AI writing tools ethically and transparently. Hoque *et al.* present *HaLLMark*, an interactive provenance system that logs a writer's interactions with an LLM and visualizes this history for readers[6]. By clearly disclosing where and how AI assistance was used, the tool helped writers maintain a sense of agency over their work while assuring audiences of human oversight. Such transparency-oriented systems support informed use of AI rather than outright prohibition. In a broader context, Park and Ahn argue that maintaining academic integrity in the age of ChatGPT requires a holistic socio-technical approach encompassing human factors and institutional policy, not just technical detection[7]. They and others advocate for guided use policies—educating students on acceptable AI assistance, requiring process documentation, and fostering a culture of integrity alongside AI tools. In contrast to reactive AI-output detectors, these approaches (and the EduCode system) aim to *design integrity into the writing process*, combining technological support with clear pedagogical guidelines for responsible AI use.

3. Learning Process and Pivot

The original goal of this project was to implement the Cognitive Authorship Protocol (CAP), a conceptual framework that verifies student understanding by generating personalized reflection prompts after submission. CAP directly targets the cognitive authenticity of student work, making it a strong philosophical fit for educational integrity in the age of LLMs. However, as we began implementation planning, we encountered significant technical challenges: CAP required NLP-based prompt generation, real-time semantic comparison between reflections and submissions, and integration with dynamic instructional interfaces — all of which exceeded the feasible scope of a semester-long project. This led us to pivot toward **EduCode**, a system-level protocol that enforces process authenticity through structural constraints and traceable content history. While CAP remains our conceptual foundation, EduCode provides a more implementable framework that still preserves the core value of transparent, AI-assisted learning. This pivot taught us the importance of aligning project ambitions with realistic engineering constraints, and how theoretical goals can inform practical system design.

4. Method

EduCode is designed with the philosophy that, given the ubiquity of AI tools, the most practical way to preserve academic integrity is to **increase the effort required** for unsanctioned AI usage while seamlessly supporting sanctioned, transparent usage. In other words, the system creates an environment that deters students from using external LLMs inappropriately by making such behavior inconvenient or ineffective, while encouraging them to use LLMs in approved ways within the platform. EduCode's implementation focuses on regulating three core user behaviors on an educational portal: **(1) typing**, **(2) copying**, and **(3) pasting**. By controlling these

operations, any content that a student attempts to move in or out of the platform can be tracked and managed.

Figure 1 gives an overview of the EduCode architecture and its major components. EduCode defines three types of endpoints to mediate interactions (explained in Section 3.2). It also introduces a specialized encoding for copy-paste operations (Section 3.1) and maintains a structured **history graph** of all content modifications (Section 3.3). Together, these mechanisms ensure that students' use of LLMs is conducted in a controlled, teacher-auditable manner, without overly hindering legitimate learning activities.

4.1 Specialized Unicode

One core idea in EduCode is to prevent students from simply copying text from the platform and pasting it into an external LLM (or vice versa) to get unsupervised answers. We achieve this by altering the way text is encoded at the user interface level. By default, almost all input methods and text areas use standard Unicode characters. EduCode, however, intentionally *scrambles* copied text into a human-readable form that is only meaningful within the educational platform.

Concretely, when an instructor enables EduCode for an assignment, the system generates a mapping from normal Unicode code points to a set of Unicode **Private Use Area (PUA)** code points. This mapping acts as a simple substitution cipher for text. Only the educational platform knows the mapping (via a secret key or seed), so any text copied out will appear garbled in other applications. Conversely, pasted text that doesn't conform to a valid mapping will not be accepted.

The mapping is generated per assignment (using a secret key plus an assignment-specific nonce to seed a pseudo-random generator). We use an HMAC-SHA256 construction with the instructor's secret key and a nonce to produce a pseudo-random bit stream. This stream is used to shuffle a subset of Unicode characters via a Fisher–Yates shuffle, assigning each character a new code point in the PUA range. For example, if the chosen PUA block starts at 0xE000 and the character 'A' is assigned an offset of 4, then 'A' would be represented by the code point 0xE004 in the EduCode interface. This procedure produces a deterministic encoding mapping for the allowed characters. It needs to be run only once when the instructor issues the assignment and is computationally inexpensive.

All copy operations in the platform are then **overwritten** to use this mapping. Whenever a student copies text, instead of placing the plain text on the clipboard, the frontend produces a **formatted, encrypted payload** string. This payload encapsulates not only the encoded content, but also metadata about its origin (as discussed below in the Copy/Paste protocol). If a student tries to copy text by any other means (for instance, through developer tools or unapproved methods), the system will instead yield an unreadable bit stream. In effect, EduCode's specialized Unicode scheme ensures that text leaving the platform cannot directly be fed into an external AI, and text coming from outside won't be understood unless properly encoded.

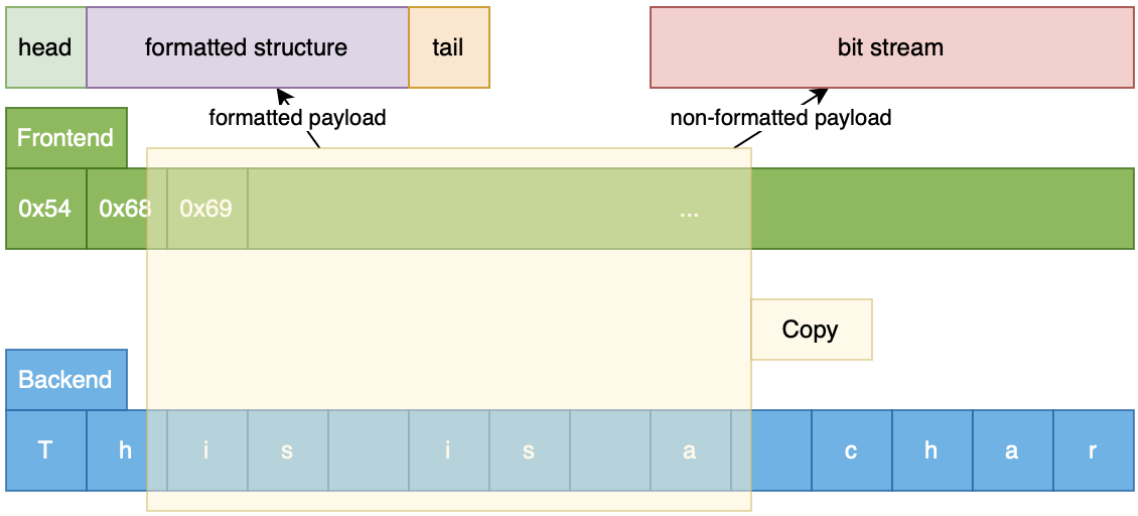

Figure 1: EduCode copy/paste encoding mechanism

As shown in Figure 1, all text copied within the EduCode platform is converted into a structured, encrypted payload (with header, body, etc.). The diagram illustrates how a copy operation takes user content (e.g., a draft answer) and produces a formatted payload. The "Head" and "Tail" segments carry metadata and checksums, while the "Body" carries the content (in specialized Unicode encoding). If a user attempts to copy via any non-approved method, the clipboard will receive only meaningless data. Conversely, when pasting, the system expects a well-formed payload and will reject or flag any text that does not conform.

It is important to note that EduCode's encoding is a **frontend-enforced** constraint. The platform itself (backend) does not need to decode the PUA characters back to standard Unicode – it can simply store and handle the content in the encoded form. Only when displaying to the user (or exporting under instructor permission) is the text rendered back in a normal readable form. This design minimizes any performance impact on the LLM service providers, as they are not required to support or handle the encoding. EduCode focuses on the education side, imposing minimal constraints on the LLM provider. From the LLM's perspective, queries coming from EduCode might be slightly obfuscated text, but in permitted scenarios the platform could decode queries before sending them to the LLM API.

4.2 Endpoints and Architecture

EduCode's controlled environment is organized around three types of **endpoints**: (1) the **Education** endpoint (EDU), (2) the **Draft** endpoint (DRAFT), and (3) the **LLM** endpoint. These roughly correspond to the roles of content provider (instructor/platform), student workspace, and AI assistant respectively. Each endpoint has a distinct purpose and a unique encryption key. Figure 2 illustrates the overall architecture and interactions between these endpoints.

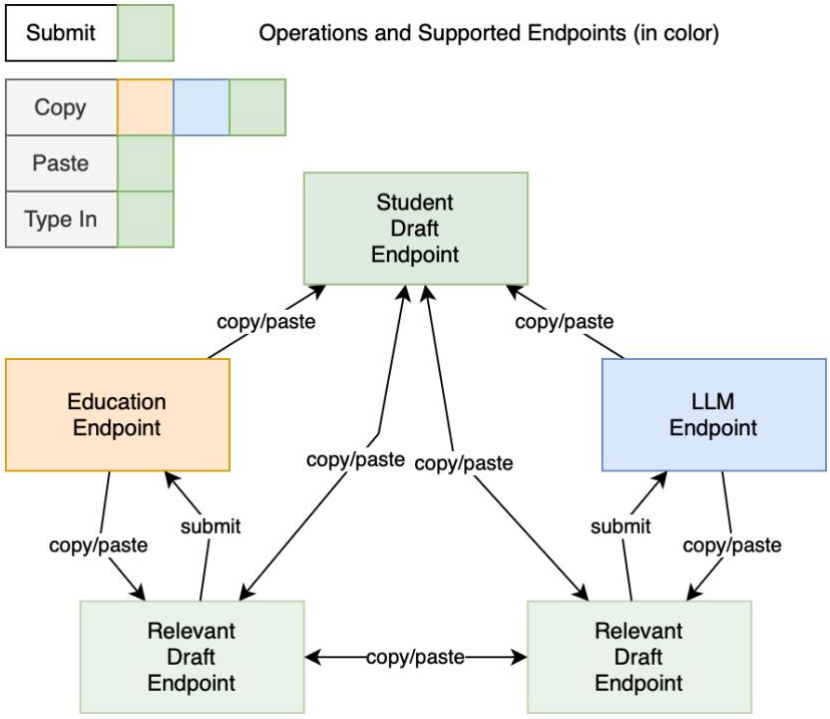

Figure 2: EduCode system architecture – endpoints and interactions

Figure 2 demonstrates that Each box represents an endpoint where content can reside: the Education endpoint (managed by the instructor/platform), the Draft endpoint (the student's working draft space), and the LLM endpoint (a regulated interface to an AI model). Arrows indicate supported operations: Students can Copy content from any endpoint, Paste into any endpoint (if permitted by policy), Type (enter original text) in Draft or Education endpoints, and Submit final answers from the Education endpoint. Copying always produces an encrypted payload tied to the source endpoint. Pasting triggers decryption and insertion of content as a new node in the target endpoint's history graph (with provenance data attached). The architecture ensures that any AI assistance (via the LLM endpoint) is only accessible through these regulated copy/paste actions, preventing raw text from leaving or entering the system without authorization.

In this architecture, the **Education endpoint** represents the problem content and the final submission space. It is controlled by the educational platform/instructor. The **Draft endpoint** is a sandbox where the student can write and revise content (think of it as a student's private notes or working area for an assignment). The **LLM endpoint** is a special interface in EduCode that allows the student to query an AI model for help, but only under the platform's supervision. Importantly, **each endpoint has its own encryption context**. This means that a copy operation from one endpoint produces a payload that is labeled with that endpoint's ID (in the payload header) and can only be decrypted by the corresponding key when pasted into another endpoint.

For instance, if a student copies text from the EDU endpoint, the payload header will indicate "source=EDU" and include enough information for the EDU backend to identify the correct key to decode it upon paste. If someone somehow obtained that payload and tried to paste it into a different system (or the wrong endpoint), it would fail to decode properly.

All payloads share a common format comprising: **(1) a header** with the source endpoint ID (so the system knows which key to use), **(2) a body** containing the actual content (in PUA encoding) possibly segmented by sub-parts (like question text, draft text, or AI response depending on source), **(3) a history graph snapshot** representing the provenance of that content (discussed more in next section), **(4) optional AAD (Additional Authenticated Data)** which can include public info such as instructions or policy notes relevant to the content, and **(5) a verification checksum** to ensure integrity. This rich payload ensures that when content is pasted, the target endpoint not only gets the text but also knows *where it came from* and *how it fits into the overall work history*.

By defining these endpoint-specific operations, EduCode prevents arbitrary transfer of text. Students can only move content via **Copy** and **Paste** within this tri-endpoint system. Any attempt to, say, select text and use the usual clipboard outside of EduCode will result in gibberish (as per Section 3.1). Likewise, if they try to paste text from an external source, the EduCode frontend will reject it unless it's formatted as a valid payload from one of the known endpoints. Standard typing (keyboard input) is allowed in the draft or directly in the education prompt, but even here EduCode can impose restrictions—e.g., it could prevent pasting large blocks of text that weren't generated in the system, forcing students to actually type out any external content (which at least increases effort and likelihood of detection).

Through this multi-endpoint structure, we effectively create a **walled garden** for AI-assisted work. The LLM endpoint acts as a gated portal to AI: students cannot directly query ChatGPT or similar by copy-pasting the assignment prompt into it unless they do so via the EduCode LLM endpoint (which would log that action). The instructor or platform can set policies on the LLM endpoint (such as limiting how much of the prompt can be sent, or requiring that the AI's answer be logged and cited). Meanwhile, any content going from the LLM back into the draft or final answer must travel via an **encrypted payload** that carries its origin information. This way, the final submitted answer can be accompanied by a trace of whether AI was used in its creation.

4.3 History Graph

To establish a traceable record of each student's problem-solving process, EduCode introduces a **history graph** mechanism. All interactions – copying from one endpoint, pasting into another, typing new text, submitting final answers – are represented as events in a directed acyclic graph (DAG) data structure. The purpose of this history graph is to encode the provenance of every piece of content in the student's work.

In the history graph, **nodes** represent states or content versions, and **edges** represent operations that transform one state into another (e.g., a "copy-paste" action linking a node in the source endpoint's graph to a new node in the target endpoint's graph). Each endpoint (EDU, DRAFT, LLM) maintains its own local history graph capturing the sequence of actions in that context. However, when content is transferred between endpoints, these graphs are connected via special cross-links (using the payload's embedded history data).

For efficiency and privacy, the actual content of each node is not broadcast in the payload. Instead, **only hash values (identifiers)** of nodes and edges are included. The platform's database stores the full history graphs (with actual content at nodes), but the payload just contains cryptographic hashes representing those nodes. This ensures that even if someone intercepted the payload or if parts of the history graph were exposed, they could not reconstruct the student's answer or the AI's answer from the hashes alone. The hashes act as unique IDs, enabling the platform to merge or extend the graphs when a paste occurs, without revealing the text.

Every operation updates the history graph in a defined way: - **Typing** (user manually writing new text) does *not* immediately create a new node in the history graph; it just updates a "current working node" in the draft. (Essentially, typing is considered an in-place edit to the current draft node until a significant action occurs.) - **Copying** finalizes the current active node (freezing that version of content) and creates a new node that represents the "copy event" output[25]. That new node's content is the payload created. The history graph thus branches: the node in the source endpoint from which the copy was made now has an outgoing edge representing the copy, pointing to the new payload node. - **Pasting** in a target endpoint also finalizes the current node in that endpoint (if any edit was in progress) and then **integrates the incoming history structure** from the payload. Essentially, the payload carries a subgraph (or references to nodes) from the source; the target will attach that subgraph as predecessor(s) to a new node in the target graph which contains the inserted content. The new node in the target is thus linked to the source's node(s) as parents, reflecting that the content originated elsewhere.

Using these rules, the history graph grows as the student works. By the end, we have a DAG representing exactly how the student composed their answer: which parts were typed originally, which parts came from the LLM (and how), which came from copying the problem statement or earlier drafts, and so on. All of it is **traceable** and time-sequenced.

This history mechanism is crucial for accountability. It means that even if a student consults the LLM, the manner and extent of that consultation is recorded. For instance, if a student copies the assignment prompt and pastes it to the LLM endpoint, we will have a record (a node that shows an edge from EDU prompt node to an LLM query node). If they then paste the LLM's answer into their draft, that content's origin is marked as from the LLM. If they edit it, the edits are noted as well (some typing on the draft node). In the end, when they submit from EDU endpoint, the submission node in EDU's graph will have incoming edges from various sources (some from purely typed content, some from LLM-derived content).

Because only hashes are transmitted, the approach is scalable and privacy-preserving. The graphs can potentially be published or shared (like as part of assignment submission metadata) without leaking the answer text, since they are just cryptographic identifiers. This could allow, for example, third-party verification or audits of the process without seeing the content (useful for competitions or research on writing processes).

Each endpoint managing its own graph also means if a student works outside the system (which we cannot stop entirely), those actions simply won't appear in the history. EduCode can't track what happens completely off-platform, but if the student then tries to integrate externally generated content, it will lack a valid history and be flagged. In practice, a student might try to circumvent by, say, using ChatGPT on another device and then manually re-typing or pasting text. EduCode's design raises the effort for this: they would have to **manually type** large sections (since direct paste of external text is disallowed), which is cumbersome and increases the chance of inconsistencies or detection by an instructor. By contrast, using the LLM endpoint within EduCode is convenient and gives them help but with oversight.

5. Experiment and Authorship Analysis

To demonstrate EduCode's functionality, we constructed a small-scale simulated assignment scenario. The goal was to show how the system captures the interactions between a student and an LLM and to evaluate the **authorship contribution** of the AI in the final work. We combined the Education endpoint and Draft endpoint on the front-end for simplicity during this simulation (i.e. the student's main interface allowed entering text that counted as both the prompt space and a draft), and similarly adjusted the LLM endpoint to streamline the workflow. The student (simulated) then followed a series of steps to solve a problem, sometimes using the AI help and sometimes working on their own, as outlined below.

Workflow Simulation:

1. *Copy the instructor's question from the EDU endpoint and paste it into the DRAFT endpoint.* (This simulates the student starting their draft by grabbing the assignment text.)
2. *Type some original content in the DRAFT endpoint.* (Student adds their own thoughts or answer elements.)
3. *Make an edit to the question or notes on the EDU side.* (Perhaps the student rephrases the question or adds a note in the prompt area – demonstrating an edit in the EDU context.)
4. *Copy text from the DRAFT endpoint and paste into the LLM endpoint.* (Student asks the AI for help by providing their current draft to the AI.)
5. *On the LLM endpoint, have the AI generate a response; copy a portion of the AI's answer.* (This represents an AI assistant answering, and the student selecting useful parts of that answer.)
6. *Paste the copied AI content into the DRAFT endpoint.* (Incorporate the AI's suggestion into the draft.)

7. *Copy another piece of content from the LLM endpoint (perhaps another part of the answer or a follow-up) and paste it into the DRAFT.* (Further use of AI answer content, done twice in our simulation.)
8. *Make further edits on the DRAFT (student's own contribution), then copy the refined draft and paste into the LLM endpoint for one more AI check/refinement.* (Simulating an iterative refinement loop with the AI.)
9. *Finally, copy the completed answer from the DRAFT endpoint back into the EDU endpoint and submit it.* (The answer is turned in through the official channel, completing the cycle.)

The above sequence covers a realistic scenario: the student oscillates between using the AI and their own editing. Every single one of these actions is captured by EduCode. The resulting **history graph** is shown in Figure 3. This graph is a DAG where each node represents a version or action and is color-coded by endpoint (green for EDU, orange for DRAFT, blue for LLM in the figure). An edge indicates a copy/paste or submit relation. For clarity, we simplified the graph by merging consecutive typings and focusing on the major copy/paste events.

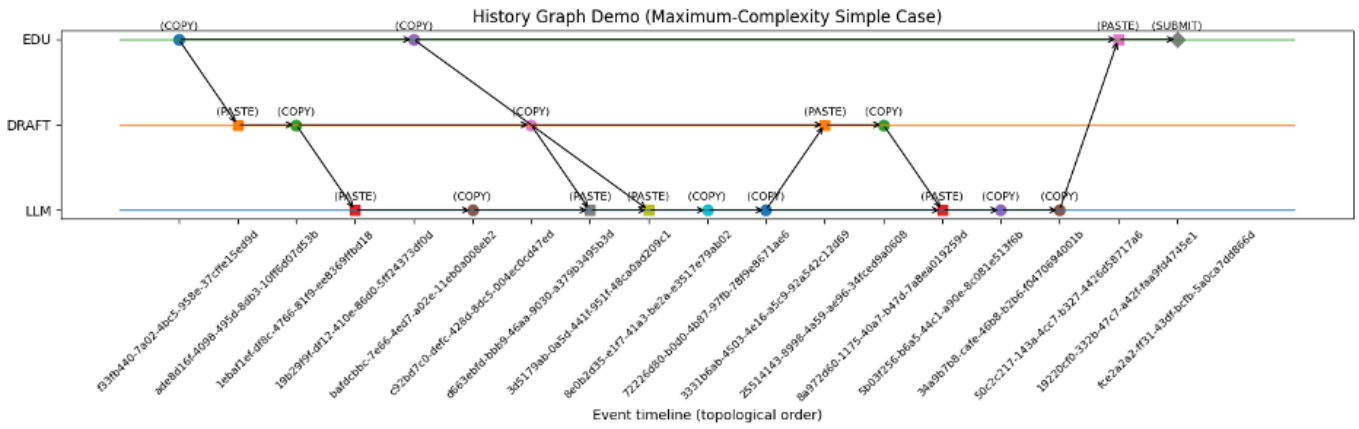

Figure 3: History graph of the example workflow

Figure 3 shows that The directed acyclic graph (DAG) represents the sequence of operations from the simulated experiment. Each node corresponds to a state or content piece (annotated with the action and endpoint), and edges show the flow of content. For example, the graph shows the initial copy from EDU to DRAFT (step 1), the drafting and editing in DRAFT (step 2), the copy from DRAFT to LLM (step 4) and subsequent return of AI-generated content to DRAFT (steps 5–7), etc. The final submission from DRAFT back to EDU (step 9) is also shown. By tracing this graph, an instructor can see exactly how the final answer was constructed and identify which parts originated from the AI.

With the history graph in hand, EduCode enables a form of **authorship analysis** – evaluating the contribution of human vs AI at each stage. We implemented an initial analysis module that treats the history graph as a flow network of content. In simple terms, we assign "credit" to each source at the **root** of the graph and propagate those credits through the edges to the final submitted node. In our scheme, content originating from the LLM endpoint is tagged as *AI-derived*, content typed in the Draft (or EDU) by the student is tagged as *human-derived*, and content that came from the EDU prompt (if reused in answer) could be tagged as *prompt-derived*. We propagate these tags in a proportional manner: when two pieces (e.g., one AI-generated and one human-generated) are merged or edited together, the resulting node inherits a mix of the credits from its parent nodes. We also consider the sizes of contributions when propagating (e.g., copying a large chunk from AI vs a small edit by human).

A potential extension of EduCode lies in leveraging the history graph for fine-grained authorship attribution. By tracing data provenance through node-level operations—including AI generations (LLM endpoint), student edits (DRAFT/EDU), and prompt reuse—the system can, in theory, estimate weighted source contributions to the final submission. Each node in the graph retains metadata about its origin and transformation. Through topological traversal and proportional credit propagation (e.g., averaging contributions from merged parent nodes), EduCode could compute an approximate authorship breakdown—such as 58% AI-generated, 31% human-authored, and 11% prompt-derived. With calibrated thresholds, this model might assign interpretive tags like "Moderate AI assistance," supporting process-level analysis rather than static detection. While not implemented in this version, such functionality would represent a principled, data-driven alternative to current AI detection tools—one that evaluates writing as a traceable synthesis of contributions, not a black-box output.

6.Conclusion

We have presented **EduCode**, a novel protocol-level approach to maintaining academic integrity in the presence of AI tools. Unlike traditional plagiarism detection or AI output detection systems, EduCode does not attempt to analyze the text artifact alone; instead, it **architects the interaction process** such that any AI involvement is explicitly marked and controlled. By combining **endpoint-specific encryption** and **structured payloads** for copy/paste, the system prevents unsanctioned use of external LLMs while allowing approved usage in a transparent manner. All student activity – whether typing, copying from materials, or querying an AI – is encoded as a **history DAG** of actions. This provides an immutable audit trail of the solution development process.

An initial **authorship analysis module** demonstrates how the history graph can be leveraged to quantify AI contributions. Rather than guesswork based on linguistic features, EduCode can literally trace which portions of an answer originated from an AI model. This process-aware, provenance-based assessment is far more robust to obfuscation and evolution of AI models, since it doesn't depend on cracking the AI's "style" but on having constrained the workflow itself.

In its current prototype form, EduCode has been tested on a limited scale. There are practical considerations for future work, such as integration with real LLM APIs, refining the user interface, and ensuring scalability in terms of graph storage and performance. Usability is also important – the system should not overly burden honest students or instructors. Nonetheless, our results point toward a promising direction: **by embedding integrity into the tools and protocols students use**, we can support innovative AI-aided learning while preserving trust and accountability. EduCode illustrates that it is feasible to let students harness powerful LLMs *without* sacrificing the transparency of their learning process. We envision this approach could be extended beyond coding or Q&A scenarios to essay writing or other domains, ultimately contributing to AI-compatible evaluation methods in education.

Acknowledgments: The author thanks Tangrui Li for the help on this project. His contribution is invaluable to the project's development.

References

[1] D. Weber-Wulff et al., "Testing of detection tools for AI-generated text," International Journal for Educational Integrity, vol. 19, no. 1, p. 26, 2023.

[2] A. M. Elkhatat et al., "Evaluating the efficacy of AI content detection tools in differentiating between human and AI-generated text," International Journal for Educational Integrity, vol. 19, no. 1, p. 17, 2023.

[3] N. Lu, D. Xu, T. Zhang, J. Liu, and X. Yuan, "Large Language Models can be Guided to Evade AI-Generated Text Detection," Transactions on Machine Learning Research, preprint, 2024.

[4] D. Kundu, A. Vattikonda, and N. Ganguly, "Keystroke Dynamics Against Academic Dishonesty in the Age of LLMs," in Proc. IEEE Int. Joint Conf. Biometrics (IJCB), 2024.

[5] S. Aburass and M. A. Rumman, "Authenticity in Authorship: The Writer's Integrity Framework for Verifying Human-Generated Text," arXiv preprint arXiv:2404.10781, 2024.

[6] M. N. Hoque, D. Lee, and D. S. Weld, "The HaLLMark Effect: Supporting Provenance and Transparent Use of LLMs in Writing with Interactive Visualization," arXiv preprint arXiv:2311.13057, 2024.

[7] H. Park and D. Ahn, "The Promise and Peril of ChatGPT in Higher Education: Opportunities, Challenges, and Design Implications," in Proc. CHI Conf. Human Factors Comput. Syst., 2024.

[8] E. Kasneci et al., "ChatGPT for Good? On Opportunities and Challenges of Large Language Models for Education," Learning and Individual Differences, vol. 103, 2023.

[9] D. Dalalah and O. M. A. Dalalah, "The false positives and false negatives of generative AI detection tools in education and academic research: The case of ChatGPT," International Journal of Management Education, vol. 21, no. 2, art. 100822, 2023.