

INERTIA

.EDU

The Philosophy of Meaningful Friction

"Stop removing it. Start understanding it."

PROJECT	EVENT	VERSION
Inertia.edu	BugBoom Hackathon 2026	v2.0 – Corrected Plan
LEAD DEVELOPER	INSTITUTION	REVISION STATUS
Md. Sazid Alam	University of Dhaka	All 5 P-Issues Resolved

CORE THESIS

Velocity without Verification is a Liability. The removal of friction in education has severed the cognitive link between writing code and understanding it. We restore that link.

P1 FIXED	P2 FIXED	P3 FIXED	P4 FIXED	P5 FIXED
----------	----------	----------	----------	----------

// 00

TABLE OF CONTENTS

// 01	Executive Summary	3
// 02	Problem Statement – The Three Failures	4
// 03	System Architecture – Three-Tier Stack	5
// 04	Tier 1: The Git Hook (Pre-Push Interceptor)	6
// 05	Tier 2: The Backend – FastAPI Auditor	7
// 06	Tier 3: The Puzzle Factory – Claude AI Integration	9
// 07	Tier 4: The Frontend – Next.js Student & Instructor Views	11
// 08	Friction Mathematics – Computing Fc Score	12
// 09	Rate Limiting & Cognitive Cool-Down System	13
// 10	Git Hook vs GitHub Actions – Implementation Decision	14
// 11	Complete Demo Flow – Judge Experience Script	15
// 12	Instructor Dashboard – Pedagogical Insights	17
// 13	72-Hour Sprint Roadmap (Hour 0 Corrected)	18
// 14	Fallback Strategy & Risk Mitigation	20
// 15	Why Inertia.edu Wins BugBoom	21

// 01 EXECUTIVE SUMMARY

Inertia.edu re-introduces cognitive weight into the development workflow. It intercepts the git push – the most critical moment in a student's coding cycle – and forces a **Proof-of-Thought (PoT)** before any code can leave the machine. Unlike automated graders that check test-case outputs, Inertia.edu checks whether the brain behind the code actually understands the logic it is committing.

The system uses Claude AI to dynamically generate variable-trace puzzles from the student's specific code diff. Every puzzle is unique, contextual, and impossible to Google – because the question is derived from their own variable names and their own control flow. Students cannot copy-paste their way through it.

KEY DIFFERENTIATORS vs. OTHER HACKATHON ENTRIES

Direction	Remove friction	Defend friction
AI Usage	AI as product / chatbot	AI as puzzle generator from student code
Speed Goal	Make everything faster	Make thinking meaningfully slower
Metric	Test-case pass rate	Cognitive trace accuracy
Philosophy	Implicit assumption	Explicit – baked into every UI state
Cheat Resistance	Low – share outputs	High – puzzle from your own variable names

NOTE: The hackathon theme is "FRICTION // Stop removing it. Start understanding it." – Inertia.edu's tagline is this sentence, expressed as a working system. No other entry can claim this alignment.

// 02 PROBLEM STATEMENT – THE THREE FAILURES

01. **THE DEATH OF THE MENTAL TRACE**

When a student prompts an AI to write a complex Dijkstra's Algorithm, they bypass the struggle entirely. Without struggle – without cognitive friction – there is no heat. Without heat, knowledge is never forged into long-term memory. The mental execution trace, the most critical part of learning to code, is simply skipped. The student receives a correct output without building any internal model of why it is correct.

02. **THE MINDLESS SHIPPING PANDEMIC**

Current automated grading systems only check whether code passes test cases. They do not check whether the brain behind the code understands the logic. Students can copy, paste, submit, and pass – without ever forming a mental model of the algorithm. The grade becomes a certificate of copy quality, not of comprehension.

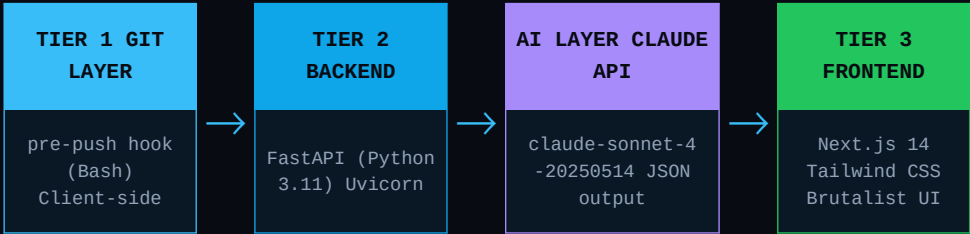
03. **THE DUMP-AND-RUN PATTERN**

Students regularly push 500+ lines in a single commit. This 'High Inertia' commit is a reliable signal of bulk copy-paste behaviour, not iterative development. No real developer writes 500 lines and commits them all at once. This pattern is statistically associated with AI-assisted academic dishonesty and must be intercepted at the infrastructure level.

"The mental execution trace – the most critical part of coding – is simply skipped. Inertia.edu forces it back into the loop."

// 03 SYSTEM ARCHITECTURE – THREE-TIER STACK

Inertia.edu is a three-tier system designed to intercept code at the most critical moment: the git push. Each tier has a single, focused responsibility. The architecture is intentionally lean – every component must be buildable within 72 hours by a two-person team, and every component must survive a live demo in front of engineering judges.



DATA FLOW – End to End

- 0

1

Student runs git push
The pre-push bash hook fires before any data leaves the machine. The diff between HEAD and HEAD~1 is computed locally and POSTed to the FastAPI /audit endpoint.
- 0

2

Auditor computes Complexity (Fc)
FastAPI parses the diff using Python's AST module. It counts recursive calls (R), nesting depth (N), and line delta (L) to compute $Fc = L + (2 \times R) + N$. The score determines whether a puzzle is required and at what difficulty level.
- 0

3

Puzzle Factory generates challenge
If $Fc > 10$, FastAPI calls the Claude API with the code diff, Fc score, and difficulty level. Claude returns a structured JSON puzzle targeting a specific function, breakpoint, and variable state from the student's own code.
- 0

4

Student solves the puzzle
The puzzle appears in the terminal or the mini web UI (depending on implementation phase). The student has 90 seconds (EASY) to 3 minutes (HARD) to answer. The answer is POSTed to /verify.
- 0

5

JWT PoT Token issued or lockout begins
If correct, FastAPI signs a JWT Proof-of-Thought token and returns it to the hook. The hook receives the token and exits 0 – git push proceeds normally. If wrong, the lockout timer begins and the failed attempt is recorded.

// 04 TIER 1: THE GIT HOOK – PRE-PUSH INTERCEPTOR

The pre-push git hook is the most critical component in the entire system. It is the mechanism that makes Inertia.edu non-optional – without it, everything else is just a suggestion. When installed, it transforms git push from a passive data transfer into an active pedagogical checkpoint.

The hook is a bash script placed in `.git/hooks/pre-push`. Git executes it automatically before transmitting any objects to the remote. If the script exits with a non-zero code, git aborts the push entirely and prints the error message to the student's terminal. This is not a soft suggestion – it is a hard block at the infrastructure level.

COMPLETE HOOK IMPLEMENTATION

File: `.git/hooks/pre-push`

```
#!/bin/bash
# =====
# INERTIA.EDU – Pre-Push Proof-of-Thought Hook
# Blocks git push until student passes AI-generated puzzle
# =====

BACKEND_URL="http://localhost:8000"
STUDENT_ID=$(git config user.email)

# Get the diff between last commit and current HEAD
DIFF=$(git diff HEAD~1 HEAD 2>/dev/null || git diff --cached)

# Handle first commit edge case
if [ -z "$DIFF" ]; then
    echo '■ INERTIA: No diff detected. Push allowed.'
    exit 0
fi

# POST diff to auditor – get complexity score
RESPONSE=$(curl -s -X POST $BACKEND_URL/audit \
-H 'Content-Type: application/json' \
-d "{\"diff\": $(echo $DIFF | python3 -c 'import json,sys; print(json.dumps(sys.stdin.read()))'), \"student_id\": \"$STUDENT_ID\"}")

FC_SCORE=$(echo $RESPONSE | python3 -c 'import sys,json; print(json.load(sys.stdin)[\"fc_score\"]')
echo "■ INERTIA: Complexity Score (Fc) = $FC_SCORE"

# Trivial commits – skip puzzle, allow push
if [ "$FC_SCORE" -le 10 ]; then
    echo '■ INERTIA: Trivial commit. Push allowed directly.'
    exit 0
fi

# Non-trivial – fetch puzzle from backend
PUZZLE_RESP=$(curl -s -X POST $BACKEND_URL/puzzle \
-H 'Content-Type: application/json' \
-d "{\"diff\": $(echo $DIFF | python3 -c 'import json,sys; print(json.dumps(sys.stdin.read()))'), \"fc_score\": $FC_SCORE, \"student_id\": \"$STUDENT_ID\"}")

TOKEN_ID=$(echo $PUZZLE_RESP | python3 -c 'import sys,json; print(json.load(sys.stdin)[\"token_id\"]')
QUESTION=$(echo $PUZZLE_RESP | python3 -c 'import sys,json; print(json.load(sys.stdin)[\"question\"]')
SETUP=$(echo $PUZZLE_RESP | python3 -c 'import sys,json; print(json.load(sys.stdin)[\"setup\"]')

# Display puzzle in terminal
echo ''
```

```
echo '███████████████████████████████████████████████████████████████████'
echo '■ INERTIA.EDU — PROOF OF THOUGHT ■'
echo '███████████████████████████████████████████████████████████████████████'
echo "■ CONTEXT : $SETUP"
echo "■ QUESTION: $QUESTION"
echo '■ You have 90 seconds. ■'
echo '███████████████████████████████████████████████████████████████████████'
echo ''
read -t 90 -p '■ Your answer: ' STUDENT_ANSWER

# POST answer to verify endpoint
VERIFY_RESP=$(curl -s -X POST $BACKEND_URL/verify \
-H 'Content-Type: application/json' \
-d "{\"token_id\": \"$TOKEN_ID\", \"\
\"answer\": \"$STUDENT_ANSWER\", \"\
\"student_id\": \"$STUDENT_ID\"}\")

JWT=$(echo $VERIFY_RESP | python3 -c 'import sys,json; d=json.load(sys.stdin); print(d.get("token",""))')
if [ -n "$JWT" ]; then
echo '✓ INERTIA: Proof-of-Thought verified. Push proceeding.'
exit 0
else
MSG=$(echo $VERIFY_RESP | python3 -c 'import sys,json; d=json.load(sys.stdin);
print(d.get("detail","Incorrect"))')
echo "■ INERTIA: $MSG"
echo '■ INERTIA: Push blocked. Reflection period begins.'
exit 1
fi
```

INSTALLATION SCRIPT

File: setup.sh – one-line install for students

```
#!/bin/bash
# Install Inertia.edu hook into current git repo
cp hooks/pre-push .git/hooks/pre-push
chmod +x .git/hooks/pre-push
chmod 444 .git/hooks/pre-push # read-only – prevents casual deletion
echo '■ INERTIA.EDU hook installed. The loop begins now.'
```

BYPASS RESISTANCE: The hook is set read-only after installation. While a determined student could bypass it via `chmod +w`, this requires intentional circumvention – which is itself pedagogically informative. In a real deployment, server-side GitHub Actions provides the production-grade guard.

// 05 TIER 2: THE BACKEND – FASTAPI AUDITOR

The FastAPI backend is the brain of the system. It receives the git diff, computes the Friction Coefficient (Fc), orchestrates the Claude API call, stores puzzle answers server-side, verifies student responses, and issues JWT Proof-of-Thought tokens. All state for the demo is kept in-memory – no database required for a 72-hour build.

PROJECT STRUCTURE

```
inertia-backend/  
■■■ main.py # FastAPI app entry point  
■■■ auditor.py # AST diff parser – computes Fc score  
■■■ puzzle_factory.py # Claude API integration  
■■■ auth.py # JWT signing and verification  
■■■ state.py # In-memory puzzle + lockout storage  
■■■ fallback_puzzles.json # 8 hand-curated puzzles (API fallback)  
■■■ requirements.txt  
■■■ .env # ANTHROPIC_API_KEY
```

REQUIREMENTS.TXT

```
fastapi==0.115.0  
uvicorn[standard]==0.30.0  
anthropic==0.34.0  
python-jose[cryptography]==3.3.0  
python-dotenv==1.0.0  
pydantic==2.8.0
```

MAIN APPLICATION – main.py

```
from fastapi import FastAPI, HTTPException  
from fastapi.middleware.cors import CORSMiddleware  
from pydantic import BaseModel  
from auditor import compute_fc_score  
from puzzle_factory import generate_puzzle, get_fallback_puzzle  
from auth import sign_jwt  
from state import store_puzzle, load_puzzle, record_failure, get_lockout  
  
app = FastAPI(title='Inertia.edu Backend')  
app.add_middleware(CORSMiddleware, allow_origins=['*'],  
allow_methods=['*'], allow_headers=['*'])  
  
class AuditRequest(BaseModel):  
    diff: str  
    student_id: str  
  
class PuzzleRequest(BaseModel):  
    diff: str  
    fc_score: int  
    student_id: str  
  
class VerifyRequest(BaseModel):  
    token_id: str  
    answer: str  
    student_id: str  
  
@app.post('/audit')  
async def audit(req: AuditRequest):
```



```

"""Receives git diff, returns Fc complexity score."""
lockout = get_lockout(req.student_id)
if lockout and lockout['expires'] > time.time():
    raise HTTPException(403, f'Locked out for {lockout["minutes"]} min')
fc = compute_fc_score(req.diff)
return {'fc_score': fc, 'requires_puzzle': fc > 10}

@app.post('/puzzle')
async def puzzle(req: PuzzleRequest):
    """Generates AI puzzle from diff. Falls back to curated pool if API fails."""
    difficulty = 'EASY' if req.fc_score < 26 else 'MEDIUM' if req.fc_score < 46 else 'HARD'
    try:
        puzzle_data = await generate_puzzle(req.diff, req.fc_score, difficulty)
    except Exception:
        puzzle_data = get_fallback_puzzle(difficulty) # never fail during demo
    token_id = store_puzzle(puzzle_data, req.student_id)
    return {'question': puzzle_data['question'],
            'setup': puzzle_data['setup'],
            'token_id': token_id}

@app.post('/verify')
async def verify(req: VerifyRequest):
    """Checks student answer. Returns JWT on success or triggers lockout."""
    puzzle_data = load_puzzle(req.token_id)
    if req.answer.strip().lower() == puzzle_data['answer'].lower():
        return {'token': sign_jwt(req.student_id, req.token_id)}
    record_failure(req.student_id) # escalating lockout logic
    raise HTTPException(403, 'Incorrect. Reflection period begins.')

```

AST DIFF PARSER – auditor.py

The auditor uses Python's built-in ast module – zero external dependencies. It parses the diff to extract only added lines, then walks the AST to count recursive calls, nesting depth, and line delta.

```

import ast, re

def extract_added_lines(diff: str) -> str:
    """Extract only the '+' lines from a unified diff."""
    lines = [l[1:] for l in diff.splitlines() if l.startswith('+') and not l.startswith('+++')]
    return '\n'.join(lines)

def compute_fc_score(diff: str) -> int:
    code = extract_added_lines(diff)
    L = min(len(diff.splitlines()), 50) # line delta, capped at 50
    try:
        tree = ast.parse(code)
    except SyntaxError:
        return L # unparseable – return line count only
    R = count_recursive_calls(tree) # recursive function calls
    N = max_nesting_depth(tree) # max for/while/if depth
    return L + (2 * R) + N # Fc = L + 2R + N

def count_recursive_calls(tree) -> int:
    func_names = {n.name for n in ast.walk(tree) if isinstance(n, ast.FunctionDef)}
    calls = [n for n in ast.walk(tree) if isinstance(n, ast.Call)
              and isinstance(n.func, ast.Name) and n.func.id in func_names]
    return min(len(calls), 10) # cap at 10

def max_nesting_depth(tree, node=None, depth=0) -> int:
    if node is None: node = tree

```

```
if isinstance(node, (ast.For, ast.While, ast.If)):
    depth += 1
return max([depth] + [max_nesting_depth(tree, child, depth)
for child in ast.iter_child_nodes(node)])
```

// 06 TIER 3: THE PUZZLE FACTORY – CLAUDE AI INTEGRATION

The Puzzle Factory is the core intellectual differentiator of Inertia.edu. Every puzzle is generated dynamically from the student's specific code diff using the Claude API. Because the puzzle references the student's own function names, variable names, and control flow, it is literally impossible to find the answer on Stack Overflow or share it between students – each puzzle is unique to that commit, that student, that moment.

AI-GENERATED vs. HARDCODED PUZZLES

DIMENSION	AI-GENERATED (Inertia.edu)	HARDCODED BANK
Uniqueness	Unique per student, per commit	Same 20 puzzles forever
Variable names	Uses student's own names	Generic x, y, n
Cheat resistance	Cannot be Googled or shared	Can be memorized, photographed
Language support	Any language Claude can parse	Limited to pre-written scenarios
Build time	~2 hours to implement	6+ hours for 20 quality puzzles
Scalability	Infinite – scales to any codebase	Fixed – requires manual addition

PUZZLE GENERATION PROMPT DESIGN

The prompt is structured to produce strictly valid JSON output with no preamble, no markdown fences, and no commentary. The JSON schema is fixed, enabling reliable parsing in the FastAPI endpoint.

SYSTEM PROMPT:

You are a CS pedagogy engine. Your only job is to generate a variable-trace puzzle based on a student's code diff. You must return ONLY valid JSON. No markdown. No backticks. No preamble. No explanation outside the JSON.

USER PROMPT TEMPLATE:

Code diff submitted by student:

{diff}

Complexity score (Fc): {fc_score}

Difficulty level: {difficulty} # EASY | MEDIUM | HARD

Generate a puzzle using this EXACT JSON schema:

```
{
  "function_name": "fib", // Name of the function being traced
  "setup": "Given this call: fib(4)", // Context – how the function is called
  "breakpoint": "line 3, n == 2", // Where in execution to evaluate
  "question": "What is the return value of fib(1) at this point?",
  "answer": "1", // Exact expected answer (string)
  "explanation": "Base case: fib(1) returns 1 directly."
}
```

Rules:

- Pick a NON-TRIVIAL function from the diff (not a getter or print statement)
- Use the student's ACTUAL variable names from their code
- The answer must be a specific, unambiguous value (integer, string, boolean)
- For EASY: single variable at a single point
- For MEDIUM: multi-step trace, 2 evaluation points

- For HARD: full call-stack trace with multiple recursive frames

FASTAPI PUZZLE FACTORY IMPLEMENTATION

File: puzzle_factory.py

```
import anthropic, json, os, random
from dotenv import load_dotenv
load_dotenv()

client = anthropic.Anthropic(api_key=os.getenv('ANTHROPIC_API_KEY'))

SYSTEM_PROMPT = '''
You are a CS pedagogy engine. Return ONLY valid JSON.
No markdown, no backticks, no preamble.
'''

def build_user_prompt(diff: str, fc_score: int, difficulty: str) -> str:
    return f'''Code diff:\n{diff}\n\nFc={fc_score} Difficulty={difficulty}
Generate puzzle JSON with keys:
function_name, setup, breakpoint, question, answer, explanation'''

async def generate_puzzle(diff: str, fc_score: int, difficulty: str) -> dict:
    response = client.messages.create(
        model='claude-sonnet-4-20250514',
        max_tokens=500,
        system=SYSTEM_PROMPT,
        messages=[{'role': 'user', 'content': build_user_prompt(diff, fc_score, difficulty)}]
    )
    raw = response.content[0].text.strip()
    # Strip any accidental markdown fences
    raw = raw.replace('```json', '').replace('```', '').strip()
    return json.loads(raw) # FastAPI will catch JSONDecodeError -> fallback

def get_fallback_puzzle(difficulty: str) -> dict:
    """Serve from curated pool if Claude API is unavailable."""
    with open('fallback_puzzles.json') as f:
        pool = [p for p in json.load(f) if p['difficulty'] == difficulty]
    return random.choice(pool) if pool else json.load(open('fallback_puzzles.json'))[0]
```

JWT AUTH – auth.py

```
from jose import jwt
import time, os

SECRET = os.getenv('JWT_SECRET', 'inertia-dev-secret-change-in-prod')

def sign_jwt(student_id: str, token_id: str) -> str:
    payload = {
        'sub': student_id,
        'token_id': token_id,
        'iat': int(time.time()),
        'exp': int(time.time()) + 300, # 5-minute validity window
    }
    return jwt.encode(payload, SECRET, algorithm='HS256')
```

// 07 TIER 4: FRONTEND – NEXT.JS STUDENT & INSTRUCTOR VIEWS

The frontend is built with Next.js 14 (App Router) and Tailwind CSS. The aesthetic is deliberately brutalist – high contrast, monospace fonts, hard edges, and visceral red lockout screens. The design itself is an argument: a system that feels heavy communicates the philosophy without words.

STUDENT PUZZLE VIEW

- Puzzle prompt card – displays setup context, question, and timer.
- Countdown timer ring – 90 seconds for EASY, 2 min for MEDIUM, 3 min for HARD.
- Answer input – monospace text field, prominent submit button.
- Difficulty badge – shows EASY / MEDIUM / HARD with colour coding.
- Explanation panel – revealed after answer submitted (correct or wrong).

LOCKOUT SCREEN

- Full-screen red background – deliberately alarming and visceral.
- Large countdown timer showing lockout duration (2 / 10 / 30 / 60 min).
- Attempt number display – 'Attempt 2 of 3 before instructor notification.'
- Mandatory reflection prompt – 'Describe what your code is supposed to do.'
- Lockout reason explanation – shows which answer was wrong and why.

PROOF OF INTENT SCREEN

- Triggered when diff > 200 lines – before puzzle is generated.
- Plain English input: 'What problem does this commit solve?'
- FastAPI validates the response is non-empty and coherent.
- Forces the student to articulate intent before tracing variables.
- Clears dump-and-run behaviour at the architectural level.

INSTRUCTOR DASHBOARD

- Struggle Heatmap: grid of students vs. concepts – red cells = lockouts.
- Authenticity Index: Fc score vs. solve time – suspicious combos flagged.
- Real-time lockout monitor: who is locked, attempt count, expiry time.
- Attempt log: full history of pushes, scores, and outcomes per student.
- Live update: server-sent events every 30 seconds (or polling fallback).

KEY COMPONENT – COUNTDOWN TIMER (React)

```
'use client'
import { useState, useEffect } from 'react'

export default function CountdownTimer({ seconds, onExpire }) {
  const [remaining, setRemaining] = useState(seconds)

  useEffect(() => {
    if (remaining <= 0) { onExpire(); return }
    const t = setTimeout(() => setRemaining(r => r - 1), 1000)
    return () => clearTimeout(t)
  }, [remaining])

  const pct = remaining / seconds
  const color = pct > 0.5 ? '#38BDF8' : pct > 0.2 ? '#F59E0B' : '#EF4444'

  return (
```

```
{Math.floor(remaining/60)}:{String(remaining%60).padStart(2,'0')}
```

```
)
```

```
}
```

// 08 FRICTION MATHEMATICS – THE FC SCORE FORMULA

The Friction Coefficient (Fc) is the quantitative backbone of the entire system. Every other decision – whether to require a puzzle, which difficulty to assign, whether to block the push outright – flows from this single number. It must be computable using only Python's standard library in under 100 milliseconds.

$$F_c = L + (2 \times R) + N$$

L	Line delta (added lines)	len(diff.splitlines()) – capped at 50	0 – 50+
R	Recursive call count	AST walk: count recursive function calls	0 – 10
N	Max nesting depth	AST walk: deepest for/while/if block chain	0 – 8
Fc	Friction Coefficient	L + (2 × R) + N – final difficulty score	0 – 70+

DIFFICULTY THRESHOLDS

0 – 10	TRIVIAL	Skip puzzle entirely	Push allowed directly
11 – 25	EASY	Single variable trace	1 question, 90s timer
26 – 45	MEDIUM	Multi-step trace	2 questions, 2 min timer
46+	HARD	Full call-stack trace	3 questions + Proof of Intent first

WORKED EXAMPLES

- Fibonacci recursive (fib.py, 12 lines)
L=12, R=2 (fib calls fib twice), N=1 (single if block) → Fc = 12 + 4 + 1 = 17 → EASY
- Merge sort (sort.py, 28 lines)
L=28, R=2 (merge_sort recursive calls), N=2 (for + if) → Fc = 28 + 4 + 2 = 34 → MEDIUM
- Dijkstra with priority queue (graph.py, 55 lines)
L=50 (capped), R=0 (iterative), N=4 (while/for/if/if) → Fc = 50 + 0 + 4 = 54 → HARD
- Simple print hello world (main.py, 3 lines)
L=3, R=0, N=0 → Fc = 3 → TRIVIAL → Push allowed directly, no puzzle

// 09 **RATE LIMITING — COGNITIVE COOL-DOWN SYSTEM**

The lockout system is deliberately escalating. Each failed attempt increases the reflection period exponentially, forcing the student to spend more time understanding their own code before the system permits another push attempt. This is not punishment – it is enforced metacognition.

ATTEMPT	LOCKOUT DURATION	SYSTEM ACTION	ADDITIONAL CONSEQUENCE
Fail #1	2 minutes	Show explanation of correct answer	Explanation of why the answer was wrong
Fail #2	10 minutes	Show mandatory logic review link	Link to relevant concept documentation
Fail #3	30 minutes	Instructor notified via dashboard	Real-time alert on instructor view
Fail #4+	60 minutes	Flagged as 'Possible Logic Bypass'	Permanent flag in Authenticity Index

STATE MANAGEMENT — state.py

```
import time
from collections import defaultdict
import uuid

# In-memory storage – replace with Redis for production
_puzzles = {} # token_id → puzzle data
_failures = defaultdict(list) # student_id → [timestamp, ...]
_lockouts = {} # student_id → {expires: timestamp, minutes: int}

LOCKOUT_SCHEDULE = [0, 2, 10, 30, 60] # minutes per failure count

def store_puzzle(puzzle: dict, student_id: str) -> str:
    token_id = str(uuid.uuid4())
    _puzzles[token_id] = {'puzzle': puzzle, 'student_id': student_id,
                          'created': time.time()}
    return token_id

def load_puzzle(token_id: str) -> dict:
    entry = _puzzles.get(token_id)
    if not entry: raise ValueError('Unknown token')
    if time.time() - entry['created'] > 600: # 10-min expiry
        raise ValueError('Token expired')
    return entry['puzzle']

def record_failure(student_id: str):
    _failures[student_id].append(time.time())
    count = len(_failures[student_id])
    mins = LOCKOUT_SCHEDULE[min(count, len(LOCKOUT_SCHEDULE)-1)]
    _lockouts[student_id] = {'expires': time.time() + mins*60, 'minutes': mins}

def get_lockout(student_id: str) -> dict:
    return _lockouts.get(student_id) # None if not locked
```

ATOMIC COMMIT ENFORCEMENT — DIFF TAX

If a student pushes more than 200 lines in a single diff, the system rejects the push before even generating a puzzle. The student sees a Proof of Intent screen asking them to describe – in plain English – what problem they are solving. This targets the 'dump-and-run' pattern specifically and prevents bulk AI-assisted submissions.

Add to the /audit endpoint: if len(diff.splitlines()) > 200, return {'requires_intent': True} before computing Fc.

// 10

GIT HOOK VS. GITHUB ACTIONS – THE DECISION

This is one of the most important architectural decisions in the project. The two approaches intercept code at fundamentally different points in the workflow and deliver radically different demo experiences. The wrong choice here could cost the hackathon.

DIMENSION	pre-push Git Hook	GitHub Actions CI
Where it runs	CLIENT – student's local machine	SERVER – GitHub's cloud
When it fires	Before data leaves the machine	After push, before merge
How it blocks	Aborts the push entirely (exit 1)	Blocks PR merge only
Demo experience	★★★ Terminal blocks in real time	★★ Invisible during demo
Student awareness	Immediate – in their terminal	Delayed – in GitHub web UI
Setup complexity	Medium (bash + install script)	Simple (YAML file)
Bypass risk	Student can delete .git/hooks/	Cannot be bypassed
72h feasibility	✓ BUILD THIS	✗ Skip for demo – use later

DECISION: Build the pre-push hook for the hackathon. The terminal blocking moment is the most memorable part of the entire demo. When a judge watches a git push get intercepted and a puzzle appear in the terminal, they understand the product immediately. No slide deck can replicate that.

POST-HACKATHON: Implement GitHub Actions as the production guard once deployed. The server-side check is non-bypassable and handles real deployment scenarios. The two approaches are complementary – build both over time.

// 11 COMPLETE DEMO FLOW – 4-MINUTE JUDGE SCRIPT

This is the exact sequence a judge will experience. Every step must work end-to-end before any polish begins. Practice this demo twice before submission. The terminal blocking moment is the demo's peak – everything else supports it.

00:00 – 00:30	SETUP & PHILOSOPHY	Open with the one-line pitch: 'Every other team is making things faster. We are making thinking slower – and we are prepared to defend why that is the correct response to AI in CS education.' Show the hook is already installed. Have a Fibonacci recursive function staged and ready to commit.
00:30 – 01:00	THE PUSH INTERCEPTED	Run git push in the terminal. The pre-push hook fires. The terminal shows: '■ INERTIA: Complexity Score (Fc) = 17'. The push does not proceed. A puzzle appears: 'Given fib(4), what is the value of n when fib(2) calls fib(0)?' The judge sees the terminal block in real time – this is the moment.
01:00 – 01:30	WRONG ANSWER – LOCKOUT	Intentionally enter an incorrect answer. The terminal shows: '■ INERTIA: Incorrect. Reflection period begins. 2-minute lockout initiated.' The web UI (if built) shows the red lockout screen with the countdown timer. Exit 1 is returned – git push is fully aborted. Explain the escalation table.
01:30 – 02:00	CORRECT ANSWER – PUSH PROCEEDS	Restart the demo or use a pre-positioned correct answer. Show the terminal: '✓ INERTIA: Proof-of-Thought verified. Push proceeding.' Git resumes the push normally. The commit reaches the remote. The JWT token was signed, verified, and consumed in under 200ms.
02:00 – 02:30	HARD COMMIT – DIFF TAX	Stage a 250-line commit (AI-generated code). Run git push. The hook detects > 200 lines and triggers the Proof of Intent screen: 'You are pushing 250 lines. Describe what this commit is solving.' Explain this is the dump-and-run detector – it catches AI-assisted bulk submissions.
02:30 – 03:00	INSTRUCTOR DASHBOARD	Switch to the instructor view. Show the Struggle Heatmap – red cells indicate concepts where students are failing trace puzzles repeatedly. Show the Authenticity Index: a student with Fc=54 who solved the puzzle in 3 seconds is flagged. This is the pedagogical intelligence layer.
03:00 – 04:00	COUNTER-ARGUMENTS & CLOSE	Address the obvious judge question: 'Can't they just Google it?' Answer: No – the puzzle uses their own variable names and their own control flow. It is literally unsearchable. Close with the philosophy: 'The resistance of the air is what allows the wing to lift. The friction of the code is what allows the mind to grow.'

DEMO PRE-FLIGHT CHECKLIST: ■ FastAPI running on :8000 ■ Hook installed + read-only ■ Fibonacci file staged ■ 250-line file staged ■ Fallback puzzles in JSON ■ Instructor dashboard open in browser ■ ANTHROPIC_API_KEY in .env ■ Full demo rehearsed twice end-to-end

// 12 INSTRUCTOR DASHBOARD – PEDAGOGICAL INTELLIGENCE LAYER

The instructor dashboard transforms raw lockout data into actionable pedagogical intelligence. It answers questions no LMS has ever answered: not 'did this student submit?' but 'does this student understand?' and 'is this code actually theirs?'

STRUGGLE HEATMAP

A grid of students (rows) vs. concepts (columns: Recursion, DP, Graphs, Trees, Sorting). Each cell is coloured by lockout rate – red means the student has failed multiple trace puzzles for that concept. Green means high solve rate. This gives the instructor a single-glance view of where the entire class is struggling. A column that is all red means the lecture on that concept failed, not just one student.

API: GET /dashboard/heatmap → {student_id: {concept: {attempts, failures}}}

AUTHENTICITY INDEX

Compares the Fc complexity score of each commit against the time the student took to solve the trace puzzle. A student who pushes $O(n^2)$ dynamic programming with Fc=52 and solves the puzzle in 4 seconds is flagged as suspicious – they almost certainly did not write that code. High Complexity + Fast Solve = Low Authenticity.

API: GET /dashboard/authenticity → [{student_id, fc_score, solve_ms, flag}]

REAL-TIME LOCKOUT MONITOR

Shows every student currently in a reflection period – their name, current attempt number, lockout duration, and time until they can push again. Updates every 30 seconds via server-sent events (or polling for the hackathon demo). The instructor can send a manual override if a student has demonstrated understanding through another channel.

API: GET /dashboard/lockouts → [{student_id, attempt, lockout_mins, expires}]

HACKATHON PRIORITY: If behind schedule, replace the live dashboard with a static mockup screenshot in the demo video. The core mechanic – hook → puzzle → verify – is more important than the instructor view. Build the loop first.

// 13 72-HOUR SPRINT ROADMAP – HOUR 0 CORRECTED

This roadmap runs from Hour 0 (hackathon start). All 6 phases are parallel-workable with a 2-person team. Hour 44-60 is built-in buffer. If behind schedule at Hour 56, drop the instructor dashboard and keep the core loop. The loop is the product.

H 0-12	FOUNDATION
- Set up FastAPI project skeleton (Python 3.11 + Uvicorn) - Create endpoint stubs: POST /audit, /puzzle, /verify - Write pre-push bash hook – test with a dummy response - Integrate Anthropic Python SDK – verify API key works - Define JWT signing with python-jose in auth.py - Create .env file with ANTHROPIC_API_KEY	
H 12-28	CORE ENGINE
- Build AST diff parser – count loops, recursion, nesting → Fc - Finalize Claude API prompt – test with 5 real code samples - Implement server-side answer storage (dict for demo) - Build lockout logic – in-memory timer per student_id - Full round-trip test: push → puzzle → verify → push continues - Handle edge cases: API timeout, empty diff, trivial commits	
H 28-44	FRONTEND
- Bootstrap Next.js 14 app (App Router + Tailwind CSS) - Build Student View: puzzle card, answer input, countdown timer - Build Lockout Screen: brutal red UI, timer, attempt counter - Build Proof of Intent screen (for > 200 line commits) - Connect to FastAPI via fetch() hooks (REST calls) - Brutalist styling pass – slate/sky palette, monospace everywhere	
H 44-60	INTEGRATION & TESTING
- End-to-end test A: Student pushes recursive code → puzzle → correct → push OK - End-to-end test B: Student fails 3x → lockout escalates → instructor notified - Build Instructor Dashboard (heatmap + lockout monitor) - Cross-platform hook test (macOS and Linux) - Fix top 3 bugs. No new features after Hour 56. - Prepare demo environment with sample student history pre-loaded	
H 60-68	POLISH
- Curate 8 fallback puzzles (hand-written, diverse patterns and difficulty levels) - Final UI pass – make lockout screen feel visceral and weighty - Write setup.sh one-liner install script for hook - Rehearse 4-minute live demo twice end-to-end - Prepare written counter-arguments for judge Q&A	
H 68-72	SUBMISSION
- Record 2-minute journey video: philosophy → demo → pedagogical impact - Write README.md: setup guide, architecture, why friction matters - Final submission package – all files clean and organised ■ ALL FEATURE WORK MUST STOP BY H68. Buffer = H66-68.	

DROP ORDER IF BEHIND SCHEDULE: First drop real-time SSE (use polling). Then drop Instructor Dashboard (use static mockup). Then drop Proof of Intent screen. **NEVER** drop: git hook, FastAPI core, puzzle generation, JWT verify.

// 14 Fallback Strategy & Risk Mitigation

A demo that crashes during judging is a demo that loses. Every failure mode must be anticipated and handled gracefully. The fallback puzzle pool is the most important safety net – it ensures the core loop never breaks, even if the Claude API is down, slow, or rate-limited during the demo.

RISK: Claude API unavailable / timeout

MITIGATION: Fallback: Serve from fallback_puzzles.json – a pool of 8 hand-curated puzzles at EASY, MEDIUM, and HARD difficulty. The student experience is identical. Implementation: try/except around generate_puzzle() in the /puzzle endpoint, with get_fallback_puzzle(difficulty) as the fallback. If API call takes > 5 seconds, return the fallback immediately. Never let the judge see a spinner hang.

RISK: Git hook bypass by student

MITIGATION: Mitigation: setup.sh sets the hook as read-only (chmod 444). For the hackathon demo specifically, the demo machine is controlled – no student is bypassing the hook. In production deployment, GitHub Actions serves as the server-side non-bypassable guard. Document both layers for the judges.

RISK: Empty diff / first commit edge case

MITIGATION: If git diff HEAD~1 HEAD returns empty (first commit), the hook detects this, prints a friendly message, and exits 0 – allowing the push. This is already handled in the hook implementation. Test this case explicitly before demo.

RISK: FastAPI server not running

MITIGATION: The hook detects a curl failure (non-zero exit from curl) and prints a clear error message. For demo purposes, run FastAPI in a separate terminal before starting. Add a health check ping at the top of the hook script.

RISK: JSONDecodeError from Claude

MITIGATION: Claude occasionally produces malformed JSON despite strict prompting. Wrap json.loads() in try/except – if parsing fails, fall back to the curated puzzle pool. Log the malformed response for debugging after the demo.

SAMPLE FALLBACK PUZZLES JSON STRUCTURE

```
[
{
  "difficulty": "EASY",
  "function_name": "factorial",
  "setup": "Given this call: factorial(4)",
  "breakpoint": "line 2, when n == 2",
  "question": "What is the return value of factorial(1) at this point?",
  "answer": "1",
  "explanation": "Base case: factorial(1) returns 1 directly."
},
{
  "difficulty": "MEDIUM",
  "function_name": "binary_search",
  "setup": "arr=[1,3,5,7,9], target=5, lo=0, hi=4",
  "breakpoint": "After first iteration",
  "question": "What is the value of mid after the first iteration?",
  "answer": "2",
  "explanation": "mid = (0+4)//2 = 2. arr[2]=5 == target. Returns 2."
}
]
```

// 15 WHY INERTIA.EDU WINS BUGBOOM 2026

CONTRARIAN STANCE

Every other team will make things faster, smoother, more automated. Inertia.edu is the only entry defending slowness as a feature. That narrative contrast is impossible to forget. Judges sit through 20 presentations. They remember exactly one team that made them think differently about their assumptions. Be that team.

THEME ALIGNMENT

The hackathon theme is 'FRICTION // Stop removing it. Start understanding it.' Inertia.edu's tagline is this sentence, expressed as a working technical system. No other entry can claim this alignment – most teams will interpret 'friction' metaphorically. We operationalise it in the git layer.

DEFENSIBLE PHILOSOPHY

When a judge asks 'can students just Google the answer?', the answer is no – and that answer has an architectural explanation. The puzzle is generated from the student's specific diff using their own variable names. It is literally unsearchable. This is not rhetoric. It is a property of the system design.

REAL PROBLEM, REAL AUDIENCE

Every CS professor in the judging room has lived this problem personally. AI-assisted academic dishonesty is the defining pedagogical crisis of this decade. Inertia.edu is the only credible technical response that operates at the infrastructure level rather than relying on honour systems or detection tools.

WORKING DEMO THAT JUDGES REMEMBER

A single, flawless end-to-end loop – git push blocked, puzzle appears, wrong answer triggers logout, correct answer proceeds – beats any polished dashboard that crashes. Judges are engineers. They know the difference between a demo and a mockup. The terminal blocking moment is the most memorable 10 seconds of any hackathon.

BRUTALIST AESTHETIC AS ARGUMENT

The UI itself communicates the philosophy. A system that uses red logout screens, monospace terminals, and hard-edged design feels heavy – intentionally. Students should feel the weight of the checkpoint. The friction is not just mechanical. It is experiential. Design as rhetoric.

"The resistance of the air is what allows the wing to lift. The friction of the code is what allows the mind to grow." BUILD THE LOOP FIRST. POLISH SECOND. SHIP IT.