An Intelligent Web-Based Adaptive Examination Framework with Inline Cognitive Stress Detection Using Native JVM Machine Learning

Vaishnavi Fasale[1] Pratiksha Gadekar[2] Vaishnav Mohare[3] Chandan Vaishnav[4] Mr. Shripad Bhide[5]
[1,2,3,4]Student [5]Guide
[1,2,3,4,5]Master of Computer Applications (MCA)
[1,2,3,4,5]Progressive Education Society's Modern College of Engineering, Pune, India

Abstract — Conventional online examinations present every candidate with the same static question sequence and judge performance by the final score alone, ignoring the cognitive strain a candidate experiences during the test. Existing stress-detection approaches mostly depend on cameras or wearable sensors, and behaviour-based systems usually hand the captured telemetry to a separate service written in another language, which adds network latency and exposes sensitive data in transit. This paper presents a web-based adaptive examination framework that combines Computer Adaptive Testing based on Item Response Theory with the estimation of a Cognitive Stress Index from browser-side behavioural telemetry (key-hold time, key-flight time, cursor-hover duration, option toggling and navigation hesitation), without using a camera or microphone. A Random Forest, trained offline and exported as decision-tree weights, is held in the memory of the Java Virtual Machine (JVM) backend so that inference needs no network hop. A prototype with a student examination interface and an administrative telemetry dashboard is described. In the reported prototype tests, in-process inference took well under a millisecond and the stress classifier showed high agreement with its labels on a generated dataset; end-to-end concurrent-user tests and validation on real examinees remain future work.

Keywords: Computer Adaptive Testing, Item Response Theory, Behavioural Biometrics, Cognitive Stress Index, Native JVM Machine Learning, Spring Boot, Smile, Random Forest

I. Introduction

Online assessment is now routine in higher education, yet most platforms still deliver one fixed sequence of questions to every candidate and treat the final numeric score as the only evidence of ability. A candidate who suffers from severe test anxiety may score below what their knowledge deserves, because anxiety is associated with lower cognitive efficiency [1]. A score-only view therefore says little about the conditions under which the score was obtained.

Research on stress during online examinations has examined facial expressions [2], deep-learning models of cognitive behaviour and question difficulty [3], and behaviour-centric monitoring from software-level signals [4], [5]. Adaptive question delivery, in turn, rests on the long-established theory of Computerized Adaptive Testing (CAT) [6] and Item Response Theory (IRT) [7]. These strands are, however, rarely combined, and implementations commonly place the web container and the machine-learning pipeline in different language stacks, for example Java for the web API and Python for inference. Moving high-frequency behavioural clickstreams across such a boundary adds latency, memory overhead and a transfer layer at which data may be exposed.

This paper proposes a framework in which adaptive question routing and stress estimation run side by side inside a single JVM runtime. Telemetry is captured in the browser without any camera or microphone, converted to a feature vector on the server, and classified by a Random Forest that is loaded once into memory. The output is an *estimated* Cognitive Stress Index that supports the examiner's interpretation of a result; it is not a clinical measurement of a student's mental state. The remainder of the paper is organised as follows. Section II identifies the research gaps, Section III reviews the literature, Section IV describes the methodology, Section V presents the implementation, Section VI discusses the results, and Sections VII and VIII give the conclusion and future enhancements.

II. Research Gaps

The literature survey revealed the following principal gaps.

A. Privacy and Hardware Dependence

High-performing stress classifiers frequently rely on intrusive multimodal inputs such as continuous camera video for facial analysis [2]. Such inputs raise student-privacy concerns and are impractical in institutional laboratories where webcams or wearable sensors are unavailable. Non-invasive behavioural approaches [4], [5] avoid this dependence but are not tied to adaptive question delivery.

B. Multi-Language Runtime Separation

Behaviour-centric systems are typically built with a conventional web container that captures inputs and forwards them over a network bridge to a separate Python service for inference. JVM characterisation studies [8], [9] indicate that such hand-offs can add connection overhead, latency and allocation pressure under concurrent load.

C. Adaptive Delivery Decoupled from Stress Estimation

Question difficulty and stress are known to interact [3], yet adaptive routing [6], [7] and behavioural stress estimation are usually studied separately. To the best of our reading of the surveyed works, none evaluates a single JVM ecosystem that performs item branching and behavioural-telemetry classification together in memory.

D. Main Research Gap and Proposed Contribution

The central gap is the absence of a camera-free, single-runtime framework that couples IRT-based adaptive routing with in-memory behavioural stress estimation. The proposed framework addresses it by (i) capturing behaviour purely from browser events, (ii) embedding the trained model in the JVM backend, and (iii) feeding the performance and the stress index into one session store visualised on an administrative dashboard. We distinguish three levels of claim throughout the paper: *design intent* (what the architecture is meant to achieve), *implemented features* (what

All rights reserved by www.ijsrd.com

the prototype provides), and *measured results* (what has actually been benchmarked). Section VI states which claims fall in which category.

III. Literature Review

The related work falls into five groups: stress detection in examinations, behavioural biometrics, adaptive testing, JVM performance and embedded machine learning, and the privacy of behavioural data.

Singh, Elangovan et al. [2] proposed Exam Vision, a dual-task system that detects academic misconduct and analyses stress from facial expressions during online examinations (2026). It highlights the need to balance exam integrity with student well-being, but its dependence on continuous video is exactly what the present work avoids. Prabhakar, Palaniswamy et al. [3] applied a deep-learning approach that computes stress percentages from cognitive behaviour, knowledge depth and question difficulty (2023); the link they draw between stress and difficulty motivates coupling stress estimation with adaptive routing, although their work does not address the runtime architecture.

Non-intrusive alternatives were explored by Gonçalves, Carneiro et al. [4], whose EUStress system quantifies stress purely from behavioural biometrics (2017), and by Albastroiu, Iova et al. [5], who embedded stress detection in an e-exam platform by analysing mouse movements and decision-making behaviour (2018). Together they support the decision to infer stress from software-level behaviour. Monrose and Rubin [10] formalised key-hold and flight times as the basis of typing-rhythm profiles (2000); although their goal was authentication, the same quantities serve here as stress-related features. Liu and Zhu [11] protected interaction logs and typing patterns with federated learning and differential privacy (2025), confirming that behavioural telemetry is sensitive; the proposed design addresses the concern differently, by keeping telemetry inside the local JVM heap.

On the assessment side, Weiss and Kingsbury [6] review the algorithms by which a computerized adaptive test branches items according to performance, and Hambleton and Swaminathan [7] describe item-parameter pools built on item difficulty, the structure needed to categorise questions by difficulty and competency level. Cassady and Johnson [1] relate severe test anxiety to reduced cognitive efficiency, which justifies reading a score together with a stress indicator.

For the runtime, Elhadik and Desoky [8] address JVM performance by queuing concurrent transactions to relieve bottlenecks, and Chidambaram et al. [9] characterise memory use, CPU utilisation and garbage-collection frequency of the JVM under machine-learning workloads; both indicate which metrics must be observed when a model is embedded in a server handling concurrent users. Li [12] documents the matrix-computation capacity and execution speed of Smile, a native Java machine-learning library, which supports running the classifier inside the backend process.

In summary, the surveyed studies establish that stress can be estimated without intrusive hardware, that typing and cursor behaviour are measurable, that adaptive routing is well understood, and that machine learning can run natively on the JVM. What they do not provide is an integrated, single-runtime examination framework that combines these capabilities.

IV. Proposed Methodology

The methodology has four stages: (1) ingestion and architecture middleware, (2) telemetry capture and feature engineering, (3) in-memory machine-learning execution, and (4) adaptive routing and persistence. The architecture is shown in Fig. 1 and the examination workflow in Fig. 2.

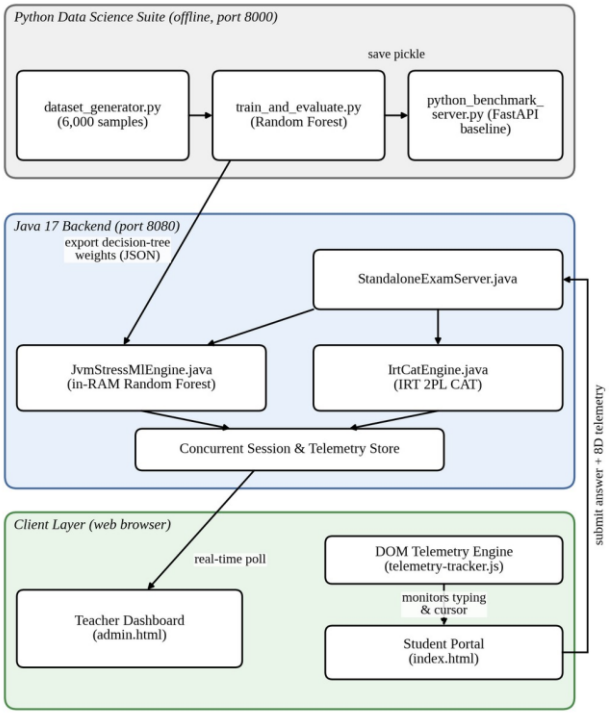


Fig. 1. Architecture of the proposed framework (redrawn from the project's architecture diagram).

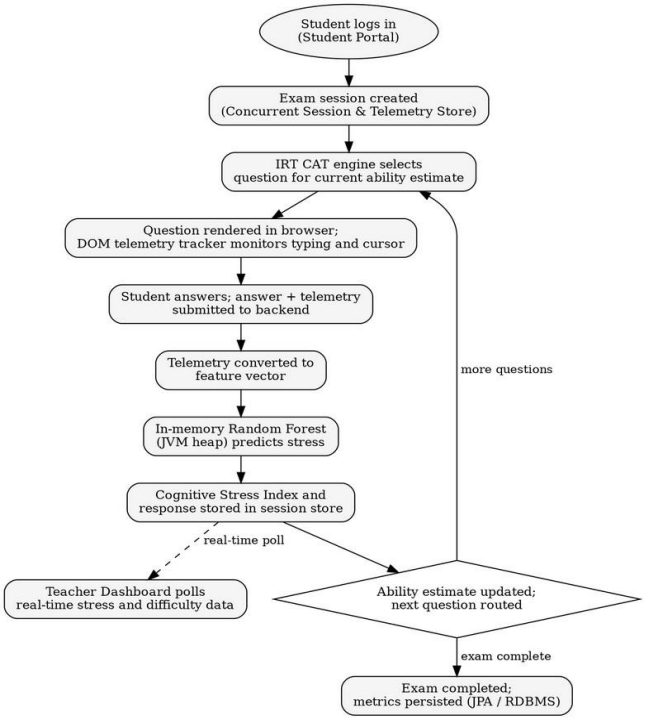


Fig. 2. Workflow of one adaptive examination session.

All rights reserved by www.ijsrd.com

A. Data Acquisition and Behavioural Telemetry

Asynchronous event listeners embedded in the examination page record five behavioural variables without any camera or microphone input: key-hold time (duration a key is held down), key-flight time (latency between consecutive keystrokes) [10], cursor-hover duration over interactive elements such as answer options, option-toggle frequency (changes of the selected option before submission), and navigation hesitation (back-and-forth interval and answer delay) [5]. On submission the values are aggregated into one telemetry payload per answered question and sent together with the answer.

B. Preprocessing and Feature Engineering

On the server the payload is converted into a numeric feature vector in the order expected by the model. The architecture labels the submitted payload as eight-dimensional (8D); the source documents do not enumerate the eight individual features or their aggregation rules, so these are not specified here.

C. Proposed System Architecture

As Fig. 1 shows, the framework has three groups of components. An offline Python data-science suite generates a 6,000-sample dataset, trains and evaluates a Random Forest, and also hosts a FastAPI baseline server used only for latency comparison. A Java 17 backend contains the exam server (StandaloneExamServer), the in-memory stress engine (JvmStressMlEngine), the IRT 2PL CAT engine (IrtCatEngine) and a concurrent session and telemetry store. The browser client contains the student portal, the DOM telemetry engine and the teacher dashboard. Training is therefore offline in Python; the runtime prediction path stays inside the Java backend.

D. Native JVM Machine Learning

The trained forest is exported as decision-tree weights in JSON form and loaded once, at back-end start-up, into the JVM heap as a singleton, so that no model is loaded per request and no external inference service is called. The synopsis specifies the Smile library [12] for this role. The model object is shared by all concurrent sessions, and the JVM characterisation metrics of [9] (memory, CPU, garbage collection) define what should be monitored as load grows.

E. Cognitive Stress Index Estimation

A Random Forest aggregates the votes of many decision trees. For each telemetry vector the forest returns a stress estimate, which is stored with the session as the Cognitive Stress Index and can be plotted against the difficulty of the question being answered. In the evaluated prototype the estimate is a three-class label (Low, Moderate, High). The index is a behavioural estimate that depends on the quality of the captured telemetry and the training data; it is not a diagnosis.

F. IRT-Based Adaptive Question Routing

The adaptive component is labelled an IRT two-parameter logistic (2PL) CAT engine. In this model the probability of a correct response depends on candidate ability and two item parameters [7]:

$$P(\theta) = 1 / (1 + e^{-a(\theta - b)}) \quad (1)$$

where θ is the candidate ability, b the item difficulty and a the item discrimination. After each answer the ability estimate is updated and the next unused question is chosen to match it: harder after a correct answer, easier after an incorrect one [6]. The initial ability value, the exact update rule and the stopping rule are not specified in the source documents.

G. Persistence and Security

The design uses a stateless REST interface, Spring Security for session management and payload protection, and Spring Data JPA persistence of finalised metrics to a PostgreSQL/MySQL database, as laid out in the synopsis. Because telemetry is evaluated inside the JVM heap and no camera, microphone or wearable data are collected, the exposure of personal data is reduced relative to video-based designs. The security configuration and the database schema are not detailed in the source documents.

H. Administrative Visualisation

A teacher dashboard polls the session store in real time and renders multi-line graphs of question difficulty, estimated ability and stress index (Chart.js), together with a registry of candidates and their stress profile.

V. Implementation

The prototype follows the architecture of Fig. 1. The backend runs on Java 17 (port 8080) and is organised around StandaloneExamServer, which receives the answer and telemetry and delegates to JvmStressMlEngine and IrtCatEngine. The client consists of index.html, telemetry-tracker.js and admin.html. The synopsis names Spring Boot 3.x, Spring Security and Spring Data JPA as the intended stack; the project report does not yet document how these map onto the server classes, nor the endpoint names, so Table I summarises what is and is not established.

Component	Status in project documents
Student exam portal	Implemented (Figs. 3–4)
Telemetry tracker (browser)	Implemented; 8D feature list not enumerated
IRT 2PL CAT engine	Implemented; update and stopping rules not specified
In-RAM Random Forest	Implemented; 6,000-sample generated dataset
Admin dashboard	Implemented (Fig. 5)
Spring Security / JPA	Specified; configuration, schema not documented
Concurrent-session tests (JMeter)	Passed (CT01–CT02); load levels not documented
Formal test cases (unit, API, etc.)	All documented cases passed (UT, IT, FT, AT, DT, AD, TT, MT, ST, PT)

Table I: Status Of the Framework Components

A. Student Examination Module

The student portal collects the candidate's name and roll number and then presents questions one at a time (Fig. 3). Each question screen shows the question number, its competency tag, the item difficulty (b), the current ability estimate (θ) and a timer.

All rights reserved by www.ijsrd.com

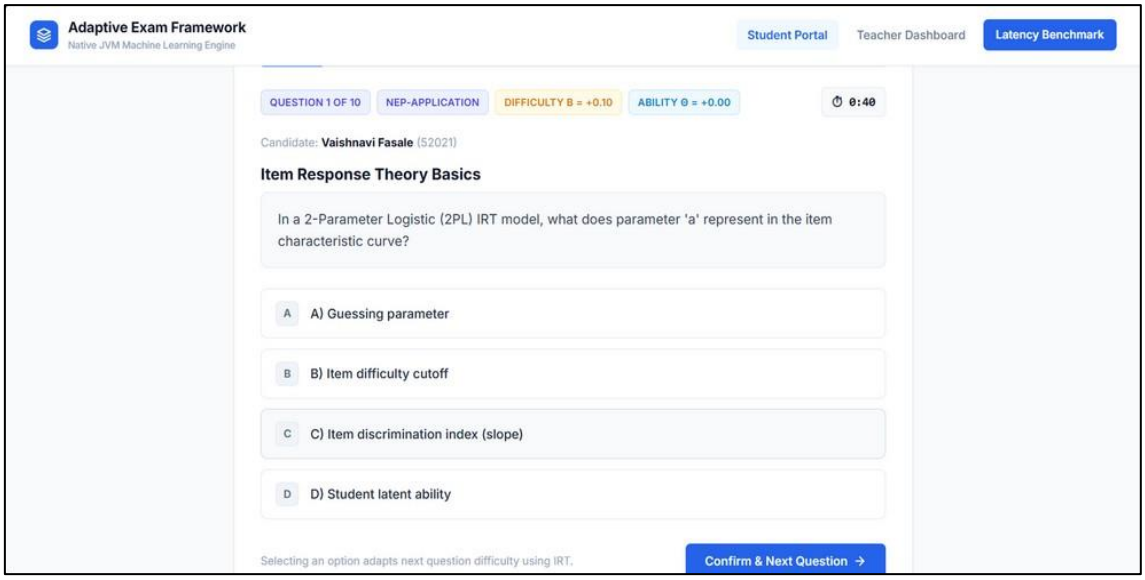


Fig. 3: Student examination interface (question 1 of 10).

B. *Adaptive Question Implementation*

After the student confirms an option, the answer is submitted and the ability estimate is updated; the next question is drawn accordingly. In Fig. 4 the candidate is on question 5 of 10 with a difficulty of b = −1.50 and an ability estimate of θ = −0.99, whereas Fig. 3 shows b = +0.10 and θ = 0.00 for the first question, illustrating the on-screen adaptation of difficulty.

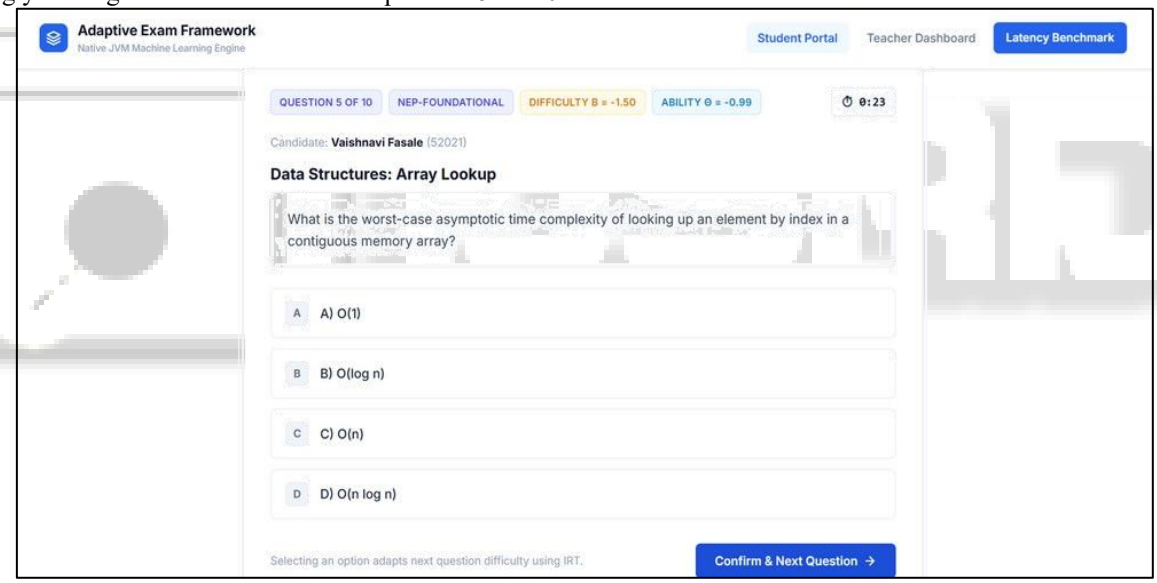


Fig. 4: Adaptive question interface (question 5 of 10).

C. *Behavioural Telemetry and Stress Prediction*

While a question is displayed, telemetry-tracker.js records typing, cursor and option events and attaches the aggregated payload to the submitted answer. JvmStressMlEngine converts the payload to a vector, evaluates every tree of the in-memory forest and returns the stress estimate, which is stored in the concurrent session and telemetry store together with the ability estimate.

D. *Administrative Dashboard*

The teacher dashboard (Fig. 5) shows the number of sessions, the cohort mean ability and mean stress index, and, for a selected candidate, the trajectory of question difficulty, estimated ability and stress index across the ten questions. A registry lists candidates with their score, mean stress and a cognitive-profile label.

All rights reserved by www.ijsrd.com



Fig. 5: Administrative dashboard: ability versus stress trajectory for one candidate.

VI. Results and Discussion

The results below are taken from the project report and its dashboard screenshots. Three caveats apply. First, the 6,000 samples were produced by a generator script (dataset_generator.py), so the classifier results describe agreement with generated labels and not validation on real examinees. Second, the timings measure in-process inference, not end-to-end request latency under concurrent users. Third, small differences exist between figures reported in different parts of the project documentation, as noted below.

A. Stress Classification

Table II gives the confusion matrix of the Random Forest as displayed by the dashboard, computed over 1,200 evaluation samples (400 per class). The diagonal corresponds to per-class recall of 97.0 % (Low), 93.0 % (Moderate) and 95.5 % (High), and no Low sample was predicted High and no High sample was predicted Low. The dashboard itself displays an overall accuracy of 94.6 % and an F1-score of 94.2 %. The per-class recall is defined as

$$Recall(c) = TP(c) / (TP(c) + FN(c)) \quad (2)$$

The train/test split and the generation procedure of the labels are not documented, and the accuracy computed directly from the matrix counts (95.2 %) differs slightly from the displayed value.

Normalized Confusion Matrix (Percentage %)

Actual \ Predicted	Class 1 (Low)	Class 2 (Moderate)	Class 3 (High)	Accuracy (%)
Low	**97.00%**	3.00%	0.00%	97.0%
Moderate	3.75%	**93.00%**	3.25%	93.0%
High	0.00%	4.50%	**95.50%**	95.5%

Table II: Normalized Confusion Matrix of the Stress Classifier (%)

B. Latency Benchmarking

Fig. 6 shows the benchmark page, which runs 1,000- or 5,000-iteration tests of the in-memory classifier and compares it with the decoupled Python microservice reached over an HTTP bridge. The page reports a mean in-JVM latency of 0.0009 ms, a 99th-percentile latency of 0.0066 ms and a throughput of about 8.1×10^5 operations per second. Table III lists the figures recorded in the project report for a separate run.

All rights reserved by www.ijsrd.com

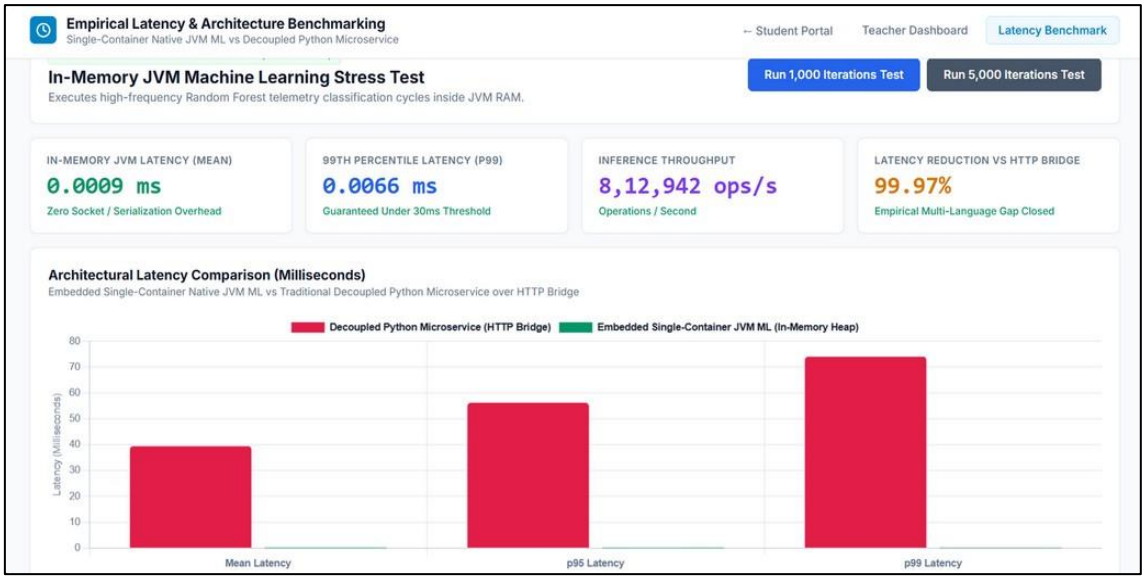


Fig. 6: Latency benchmark page comparing the embedded JVM classifier with a decoupled Python microservice.

Configuration	Load	Mean (ms)	P95 (ms)
Native JVM classifier	5,000 iter.	0.0006	0.0011
Python (FastAPI) baseline	500	77.135	102.358

Table III: Recorded Latency Figures (Report Table 9.2)

Every recorded in-JVM figure is far below the 30 ms design target, and the gap to the HTTP-bridged baseline is consistent with the expectation that removing a network hop reduces latency. These numbers must not be read as a like-for-like comparison, however: the in-JVM figures time only the model call, whereas the baseline includes serialisation and a network round trip, and the two runs use different load definitions. The project report records performance test PT01 (average in-memory inference latency of 0.01 ms) and the concurrent-session and dashboard-polling tests CT01 and CT02 as passed, but it gives no end-to-end latency measurement under concurrent load. The target of under 30 ms for the complete request under concurrent users has therefore not been quantitatively demonstrated.

C. *Adaptive Behaviour and Functional Outcome*

The screenshots show the interface updating difficulty and ability across a session (Figs. 3–5). A quantitative study of the observed difficulty progression, the stopping rule and the effect of stress on routing has not been reported. The formal unit, integration, functional, API, database, adaptive, telemetry, ML-model, security, performance and concurrent-user test cases defined in the project are recorded as passed in the updated project report, which supports functional correctness beyond demonstration alone; the report states outcomes but not detailed test data or counts.

D. *Discussion*

The prototype shows that adaptive routing, telemetry capture and in-memory stress classification can coexist in one backend, and the figures recorded so far are compatible with the design intent of low-latency, privacy-preserving inference. The principal threats to validity are the generated training data, the absence of ground-truth stress labels from real candidates, and the limited documentation of behaviour under concurrent load.

VII. Conclusion

This paper presented an adaptive examination framework that integrates IRT-based Computer Adaptive Testing with a Cognitive Stress Index estimated from browser-side behavioural telemetry, with a Random Forest evaluated inside the JVM backend and no camera or microphone input. A prototype with a student interface and an administrative dashboard was built, and the recorded benchmarks show sub-millisecond in-process inference and high agreement between predicted and generated stress labels.

The contribution is nevertheless a prototype-level one. The classifier was evaluated on generated data, the end-to-end latency target under concurrency has not been verified, and the stress index is an estimate and not a clinical measure. The work should be extended with real examinee data, controlled load tests and a documented evaluation protocol before stronger claims are made.

VIII. Future Enhancements

A. *Validation on Real Data*

Collect telemetry from real examinations with an appropriate stress reference and ethical approval, and compare further models, such as Oracle Tribuo named in the synopsis.

B. *Concurrency and JVM Profiling*

Run JMeter load studies on end-to-end request latency and profile memory, CPU and garbage collection in the manner of [9].

C. *Richer Telemetry and Index Definition*

Document the eight telemetry features, extend them, and refine the definition and scale of the Cognitive Stress Index.

All rights reserved by www.ijsrd.com

D. Stress-Aware Grading and Privacy.

Study stress-aware interpretation of results on the dashboard, building on [1], and integrate privacy-preserving mechanisms of the kind studied in [11].

REFERENCES

[1] J. C. Cassady and R. E. Johnson, "Evaluating Academic Test Anxiety and its Impact on Cognitive Performance," Cognition and Instruction, 2002.
[2] Ruchi Singh, R. Elangovan, et al., "Exam Vision: A Dual-Task System for Academic Misconduct and Stress Detection," IEEE Multimedia, 2026.
[3] P. Prabhakar, S. Palaniswamy, et al., "Stress Analysis in Online Examination using Deep Learning Approach," IEEE Conference (I2CT), 2023.
[4] F. Gonçalves, D. Carneiro, et al., "EUStress: A Human Behavior Analysis System for Monitoring and Assessing Stress," Int. Conference on Interaction Design & Children, 2017.
[5] R. Albastroiu, A. Iova, et al., "An e-Exam Platform Approach to Enhance University Student's Performance," Int. Conference on Interaction Design & Children, 2018.
[6] D. J. Weiss and G. G. Kingsbury, "Computerized Adaptive Testing: A Review of Underlying Models and Algorithms," Journal of Educational Measurement, 1984.
[7] R. K. Hambleton and H. Swaminathan, "Applications of Item Response Theory in Educational Frameworks," Kluwer Academic Publishers, 2010.
[8] S. Elhadik and A. Desoky, "Cognitive performance application," IEEE Int. Symposium on Signal Processing, 2017.
[9] S. Chidambaram, S. Saraswati, et al., "JVM characterization framework for workloads generated per ML benchmark," IEEE Int. Conference on Recent Trends, 2016.
[10] F. Monrose and A. D. Rubin, "Keystroke Dynamics as a Biometric for User Authentication," International Journal of Information Security, 2000.
[11] B. Liu and B. Zhu, "Detecting Psychological Stress via Secure Multimodal Behavioral Analysis," Proc. 10th Int. Conference on Cyber Security, 2025.
[12] H. Li, "Smile: A High-Performance Machine Learning Engine for JVM Ecosystems," Open-Source Academic Documentation, 2014.

All rights reserved by *www.ijsrd.com*