DSAN 6550 Final Project Report

Adaptive Measurement with AI: Building a 2PL-Based Computerized Adaptive Test

Authors: Idriss Moluh

Date: April 30, 2025

1. Introduction

In this project, we focus on creating a Computerized Adaptive Test (CAT) system, based on concepts from IRT (Item Response Theory), algorithmic test selection, and AI-based item generation. While the foundation is theoretical, we also built a working Streamlit dashboard.

We work as a team, with different members handling different parts, some choosing item creation and simulation of data, while some choosing modeling and visualization. Although we worked on different parts, our final system was well integrated and functioned smoothly as a complete CAT.

2. AI Item Development

We chose to select mathematics as the test domain, because it's ordered and quantifiable. Our test-takers were middle school students, and the material consisted of arithmetic, algebra, and a little basic geometry.

To develop our item pool, we used ChatGPT (text-davinci-003) to generate 30 multiple-choice items. We balanced them on difficulty: 10 easy, 10 medium, and 10 hard. The AI generated a good start, but we went through every question to analyze and edit to make it simple, have believable distractors, and the right answer. We wound up typing all questions out into a CSV file with item ID, stem, options, correct answer, and an applied item difficulty.

3. Simulating Examinee Responses

We then simulated the responses of 500 simulated examinees. We assigned each examinee a latent ability value (theta) from a standard normal distribution. We posited that ability followed a standard normal distribution and then calculated the probability of a correct response on each item with the 2PL model.

In order not to overload the task, we set the same discrimination value (a = 1.0) for all items and only changed the difficulty (b) based on whether the item was an easy, medium, or hard one. For example, easy items were given -1 as difficulty, mediums 0, and hard questions were given 1.

Once we got the probabilities from the 2PL model, we generated 0/1 response data through Bernoulli sampling. The result was a realistic pattern of correct and incorrect answers: low-theta students performed poorly on hard items, and high-theta students performed better overall. We calculated Cronbach's alpha to ensure data quality. A 0.895 value confirmed high internal consistency.

4. Estimating Item Parameters

Now was the stage of modeling. With logistic regression from scikit-learn, we estimated item parameters directly from the simulated data. Each regression had a response as a function of theta, and from the intercept and slope, we derived discrimination (a) and difficulty (b).

We did not keep everything indiscriminately—some items produced strange or flat curves and were rejected. The remainder of the set was then plotted ICCs and IICs to check that each question performed as required: its probability and information curves reached their maxima in the right places.

5. Adaptive Engine Logic

Its strength is its ability to learn. Each test started with theta set to zero. After that, the system selected the question that could provide the most information about the test-taker’s ability. We asked that question, got the response, updated theta using Maximum Likelihood Estimation, and repeated.

The system continued to repeat this process until one of two conditions was met: either the student had answered 10 questions, or its own estimate of theta had become sufficiently precise, meaning its standard error was less than 0.3. This gave the test flexibility and control.

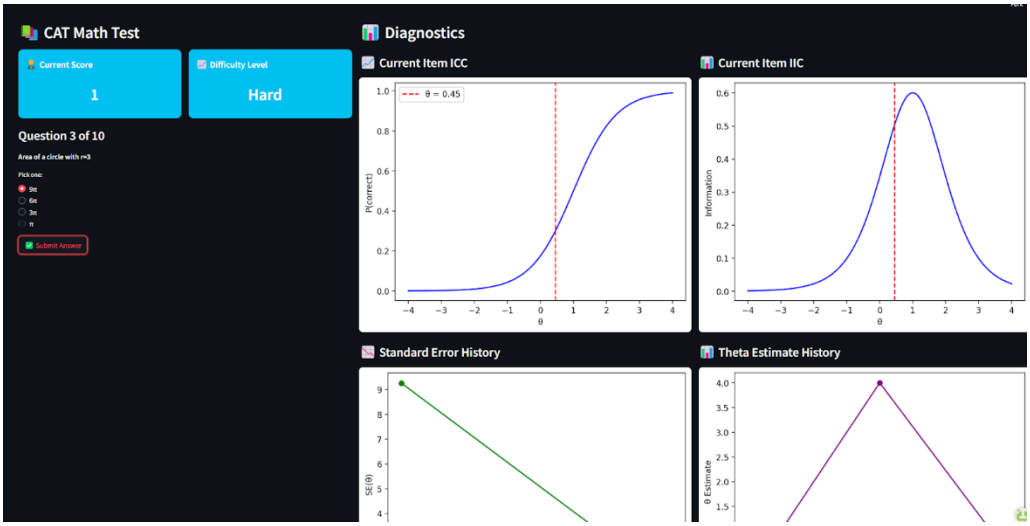
For instance, when the system estimated $\theta \approx 0.45$ after a correct answer, it chose an item with an information peak near that value. One of our figures depicts such an instance: the ICC curve has a steep slope in the neighborhood of $\hat{\theta}$, and the IIC clearly peaks near 0.5, indicating that the selected item gives maximum information for narrowing the examinee's estimate. Towards the end of the test, as θ increased (say, to 1.1 or 1.2), the selected items also shifted, each time being chosen in order to keep up with the updated estimate and reduce uncertainty.

6. Depicting the System

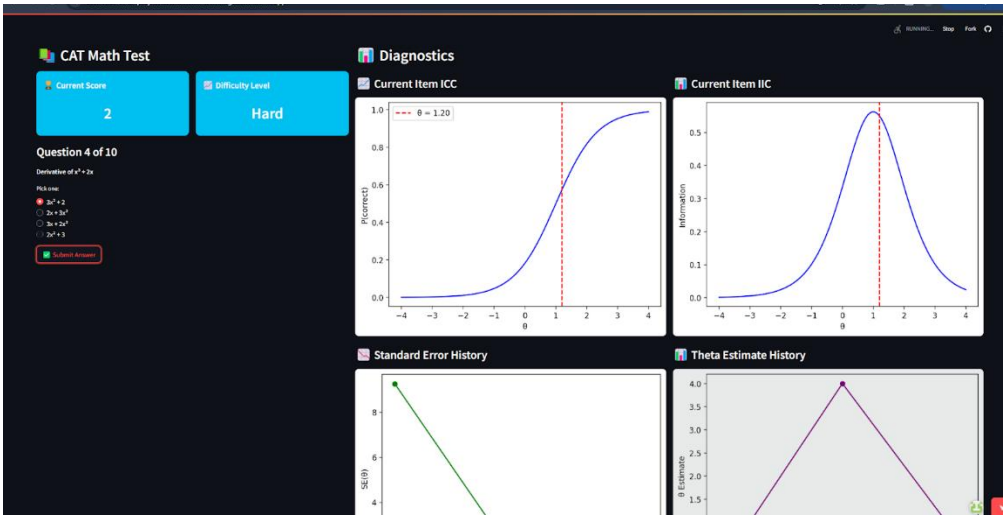
We built the front end in Streamlit, and the result is a plain, interactive dashboard that takes users through the CAT process in a transparent and easy-to-follow manner. One question is shown to the user at a time. After they have answered the question, their score and difficulty are updated immediately. Meanwhile, a right-hand panel displays two dynamic plots: the ICC of the question being answered, and its IIC, both tailored to the current estimate of theta.

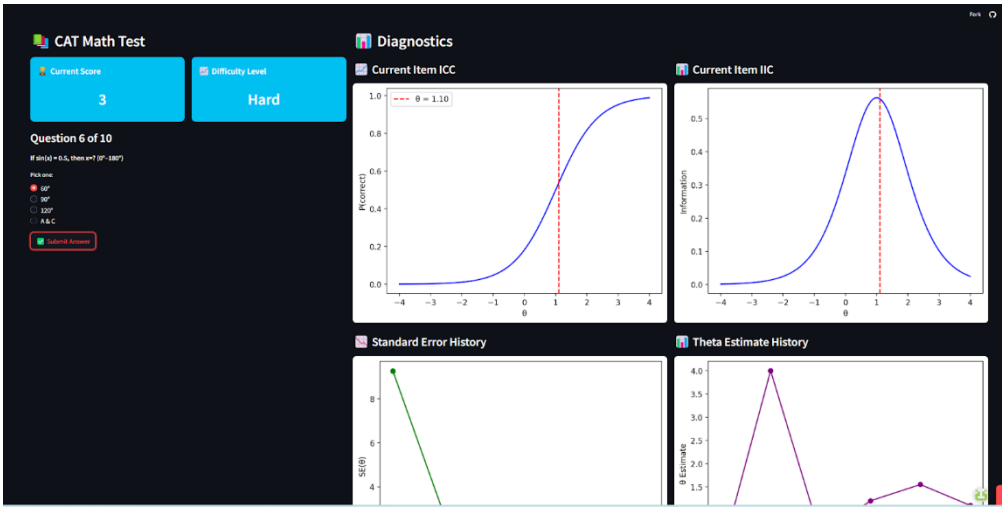
Below the dashboard are the plots showing how the user's theta and standard error change over the course of the test. These plots serve to further illustrate the adaptation process in real time.

Below is a screenshot of the dashboard for the first question:



Below are some more examples showing selection logic and information targeting:





Each question was chosen to optimize information at the present $\hat{\theta}$, and plots visually confirmed that. For instance, at $\hat{\theta} \approx 1.20$, the IIC of the selected question maximized exactly where the red dashed line was. This shows that the system is able to adjust item difficulty in real time.

To see how the system behaved, we ran it with three example respondents. Two were low-ability (theta ≈ -1.2) and one average (theta ≈ 0), and one high (theta ≈ +1.3). As expected, the system gave the low-level respondent less difficult questions and sped up quickly for the high achiever. The flow felt natural and validated the adaptive engine.

7. Challenges and Limitations

We struggled with working on short response vectors with MLE. Theta estimates would vary too erratically near the beginning of the test. We corrected that by introducing plain smoothing and setting an upper bound on the standard error early in the test.

Streamlit state management also frustrated us. It was a question of learning how to have variables persisted across reruns and avoid unwanted resets, especially navigating multi-step interactions. In the end, we used session state effectively to store progress and facilitate step-wise updates.

Of course, there are some limitations in our system. We stayed with the 2PL model for simplicity and thus did not implement guessing behavior (no "c" parameter). Also, our data were entirely simulated—no actual examinees participated. And finally, with only 30 items, the pool is too small to seriously implement, though it works well enough for us as a demo.

8. Final Reflections

This project was truly a learning process. We combined theoretical knowledge with practical implementation, which helped us gain a deeper understanding of adaptive testing, IRT, and tools like Streamlit and AI-assisted item generation.

More importantly, we learned how difficult it is to design tests that are both effective and fair. The CAT model resolves that tension by targeting items to ability level, and we're happy with the working system we came up with.