https://github.com/antonioclim/PAS_Brier_Clim

METHODOLOGY for test generation and data structure

Synthetic test construction: The script programmatically constructs each test by creating a pool of multiple-choice questions and generating both the "true" answers and simulated student responses for each question. All probabilities are handled in a **probabilistic framework**, meaning every answer or truth value is represented as a probability distribution across the options (summing to 1). This mirrors the setup in a probabilistic assessment system where students allocate confidence across all options, and there is a well-defined correct answer distribution.

- **Truth vectors:** For each question, a **truth vector** x is generated to represent the correct answer. This vector has length equal to the number of options (e.g., length 4 for a four-option multiple-choice item). The correct option is encoded probabilistically – typically as a 1 for the correct answer and 0 for all other options (a one-hot encoding of the answer key). For example, if option B is the correct answer out of {A, B, C, D}, the truth vector might be x = [0, 1, 0, 0], meaning 0% chance for A, 100% for B, 0% for C, 0% for D. In some cases, the script can also represent **partial credit** or multi-correct scenarios by distributing weight among options (e.g., x = [0.5, 0.5, 0, 0] could indicate two correct options each 50% correct, or a graded correctness notion). All truth vectors lie on the probability simplex (their components sum to 1), providing a rigorous ground truth for scoring calculations.

- **Student archetypes and belief distributions:** The script defines a set of hypothetical **student archetypes**, each representing a distinct profile of knowledge and confidence. For each archetype and each question, a **belief distribution** p is generated – this is a probability vector of the same form as the truth vector, but it represents the probabilities that a student of that archetype would assign to each option *as their answer*. These belief distributions reflect different strategies or levels of understanding:

    - *An **expert** archetype, for instance, might place a very high probability on the correct option and very little on the distractors (e.g., $p \approx [0.02, 0.95, 0.02, 0.01]$ if the second option is correct, indicating strong confidence in the right answer).*
    - *A **uniform guesser** archetype would distribute probabilities roughly evenly across all options (e.g., $p \approx [0.25, 0.25, 0.25, 0.25]$ for a 4-option question), signifying total uncertainty.*
    - *A **partial-knowledge** archetype might assign a moderate probability to one option and spread the rest among others, reflecting uncertainty or common misconceptions (for example, $p = [0.1, 0.1, 0.7, 0.1]$ could represent a student who is fairly sure about option C but not entirely certain).*
    - *Other archetypes might mimic overconfident but wrong students (high probability on an incorrect option) or underconfident students (who hedge even when they know the answer). These profiles are encoded in the script's random generation logic by tuning the probability distributions accordingly.*

https://github.com/antonioclim/PAS_Brier_Clim

- **Generation process:** The script uses the random seed to sample these probabilities in a consistent way. Typically, for each question, it first selects or assigns the correct answer (truth vector). Then for each archetype, it generates a probability vector that has random variation but is biased according to that archetype's profile relative to the truth. For example, an expert archetype's distribution might be drawn from a **biased Dirichlet** distribution concentrated near the truth vector (so that most probability mass falls on the correct answer), whereas a guessing archetype's distribution might be drawn nearly uniformly. The use of a fixed seed ensures that this stochastic process can be repeated exactly, and if --num_tests > 1, the procedure repeats independently for each test instance.

- **Data organization:** The output file collates all this information in a structured format. Each test's data typically includes:

    - A listing of the questions (often just numbered or indexed, since the content is abstract in simulation).
    - The truth vector x for each question (indicating the correct answer distribution).
    - The set of belief distribution vectors {p_arch} for each archetype (for each question). Depending on the format, these might be presented as separate tables or nested structures. For example, a JSON output might have an array of questions, each containing sub-objects for truth and for each archetype's probability array. A CSV output might list each question on a row with columns for truth and archetype probabilities, or use multiple CSV files (one for each archetype) packaged together (if so, the --zip flag is especially handy).

*In summary, the script produces a rich **synthetic dataset** that encapsulates both what the correct answers are and how different kinds of students might probabilistically respond to each question. This mirrors the data one would collect in a real probabilistic assessment setting, except here it is fully known and controlled. The methodology corresponds to the "Methods" in a research study, ensuring that any simulations or analyses performed on this data are transparent and replicable.*

RESEARCH CONTEXT & REPRODUCIBILITY

The primary purpose of generate_stem_sim.py is to support **reproducible experiments** and **simulation-based studies** in probabilistic assessment modeling. In an academic research context, using synthetic data is crucial for several reasons:

- **Controlled experiments:** Real educational data can be noisy or limited in scope, whereas this script allows generating an unlimited number of test scenarios with precise control over difficulty, randomness, and student behavior profiles. Researchers can craft scenarios to test specific hypotheses (for example, how do extreme guessers fare under logarithmic scoring compared to well-calibrated students?). By adjusting parameters like the number of questions or the archetype distributions, one can simulate tests of varying length and complexity to observe theoretical properties (such as the effect on score variance or reliability as test length increases).

https://github.com/antonioclim/PAS_Brier_Clim

- **Reproducibility:** By publishing the script and the random seed alongside a study, other researchers can exactly recreate the dataset used in experiments. This addresses the replicability standards of modern scientific work. For instance, if a paper reports a simulation result based on 1000 simulated students, the authors can share that they ran generate_stem_sim.py with a specific seed and parameter set – any reader can rerun the script with those settings to obtain identical data and verify the findings. This script-based data generation serves as a supplement or **reproducibility annex** to the research, ensuring that results are not tied to proprietary or irreproducible data.

- **Cross-validation of models and algorithms:** The synthetic data can serve as a testbed for validating the implementation of scoring algorithms, estimation methods, or analytical pipelines. Because the "ground truth" is known exactly (we know which option is correct and how the probabilities were assigned), one can verify that a scoring function (for example, the log-loss computation) produces expected values. Indeed, the script can generate known edge-case scenarios – e.g., a **uniform guesser vs. an informed student** – where the expected outcomes are theoretically calculable. This makes it possible to do consistency checks (for example, confirming that an *expert* archetype always achieves lower loss than a *guesser* on the same questions, aligning with the theory of strictly proper scoring rules).

- **Simulation of rare or hypothetical scenarios:** The script can create scenarios that might be rare in real life but are useful to study in theory. For example, one could simulate a student who is 100% confident in every answer (to see the impact of overconfidence errors), or a test with an unusually high number of options per question, etc. This flexibility allows stress-testing of the probabilistic assessment framework under various conditions. Because everything is under the experimenter's control, cause-and-effect can be isolated more easily than in observational data.

*In essence, generate_stem_sim.py underpins the research by providing a **standardized way to generate test data** for probabilistic assessments. It ensures that any simulation results or methodological experiments reported can be replicated exactly, and it integrates with analysis code to facilitate end-to-end reproducible research workflows.*

INTEGRATION INTO ANALYSIS PIPELINES

The data produced by this simulation script is meant to be readily integrated into analysis pipelines for further computation and interpretation. After generating the synthetic tests, researchers typically feed the output into statistical analysis scripts or interactive notebooks to evaluate performance metrics, test hypotheses, and visualize outcomes. Key aspects of how the generated data can be used include:

- **Loss function evaluation:** The probabilistic outputs (truth and belief vectors) are fully compatible with standard loss functions used in probabilistic assessments. For each question, one can compute the **logarithmic loss** (negative log-likelihood) for a given archetype's answer by applying the scoring rule to the truth vector x and the archetype's probability vector p. Summing over questions yields the total quiz loss for that archetype. Because the script simulates various archetypes, you can directly compare these losses to confirm theoretical expectations. For example, an *expert* archetype's average loss should be lower than a *guesser's* average loss over many questions, reflecting that knowledgeable, well-calibrated students are rewarded under a strictly

proper scoring rule. Similarly, one could compute the **Brier score** (quadratic probability score) for each response vector since the data format (probabilities and binary outcomes in truth vectors) aligns with the inputs to the Brier formula. These evaluations help validate that the scoring metrics behave as intended on the simulated data, and they allow testing of alternative scoring rules or loss functions by plugging in the generated p and x values.

- **Calibration experiments:** The synthetic dataset enables detailed calibration analysis. **Calibration** refers to how well the predicted probabilities align with actual outcomes. In our context, since we know the true answer, we can treat an archetype's probability assigned to the correct option as a "forecast" of a correct outcome. By grouping questions (or test instances) by the probability level assigned to the correct answer, one can compute how often the question was actually answered correctly in those groups. For instance, if in all cases where a student archetype said "80%" on the correct option, they indeed were correct 80% of the time, then their probabilities are well-calibrated. Using the synthetic data, one can construct **calibration curves** or reliability diagrams that plot predicted probability vs. observed frequency of correct answers. Because the data is generated from known probability distributions, we expect well-behaved calibration for certain archetypes (e.g., an honest archetype might be well-calibrated by design), whereas others might be intentionally miscalibrated (e.g., an overconfident student might assign 90% to correct answers but only be right 60% of the time). This provides a way to test calibration metrics or correction techniques. The script's output can be fed into calculations of calibration error, refinement (resolution), and other statistical measures – for example, by using the truth and prediction pairs to compute the **calibration component of the Brier score** or to run logistic regression for probability calibration.

- **Archetype-specific analysis and interpretation:** Since each response in the dataset is labeled by archetype (either explicitly or by how the data is structured per archetype), researchers can perform analysis stratified by archetype. This is extremely valuable for interpreting results:

- You can calculate summary statistics per archetype, such as the mean score (or loss) for each archetype across all questions or tests. This might show, for example, that the *expert* archetype achieves an average score corresponding to a high grade, whereas the *guesser* archetype performs no better than chance. Such results illustrate the efficacy of the scoring rule in differentiating levels of knowledge and confidence.
- **Item-level analysis** can also be done: for each question, check how each archetype performed. Did certain tricky questions stump even the high-confidence archetype, or did a particular wrong option attract high probability from the partially knowledgeable archetype? The synthetic data can be used to compute item discrimination indices by treating archetypes as groups – e.g., comparing the probability that the correct answer was identified by strong vs. weak archetypes. Though simulated, this mimics classical item response analysis in a controlled way.
- The archetype-labelled responses also facilitate qualitative interpretation. For instance, if one archetype consistently misassigns high probability to a particular wrong option, this could be analogous to a misconception in a real educational context. In a research paper, one might discuss how the scoring system penalizes that archetype's behavior, and what that implies pedagogically. The **per-archetype belief patterns** generated by the script can be visualized (for example, plotting probability distributions for each

archetype on each question) to provide insight into the confidence profiles. This was likely used in the study's appendix or supplementary materials to demonstrate the variety of student behaviors and how the system responds to each.

- **Pipeline compatibility:** Practically, the output of generate_stem_sim.py is structured to be easily read by data analysis libraries (e.g., pandas in Python or data.table in R). If the output is CSV, it can be loaded into a dataframe where each row might correspond to a question (with columns for truth and each archetype's probabilities). If JSON, it can be parsed into nested objects or converted to tables. The consistent format means that downstream analysis code (for computing losses, plotting graphs, running statistical tests, etc.) can be written once and re-used on any dataset generated by the script. This consistency is crucial for integration into automated pipelines – for example, one could integrate the script into a larger workflow that:
    - **(a)** generates a dataset,
    - **(b)** computes performance metrics and calibrations, and
    - **(c)** repeats this over many random seeds or parameter settings to conduct a Monte Carlo study.

  Indeed, in the associated research, the authors performed extensive simulations (e.g., thousands of synthetic students with varying parameters) to examine the effects on grade distributions and reliability measures; such experiments would have been infeasible without an automated data generator.

In summary, generate_stem_sim.py is a bit more than just a data generator – it is, pretty much, a foundational tool that connects the theoretical design of a probabilistic student assessment system (PAS) with empirical analysis. By using this script, researchers ensure that their models and metrics can be tested on a ***known ground truth dataset****, and they can demonstrate, with full transparency, how the probabilistic scoring framework behaves under a variety of simulated conditions. The (corresponding here) README has provided guidance on using the script (particularly in a Windows 11 environment) and has outlined how to adjust its parameters and interpret its output. Armed with this tool, specialists can confidently integrate simulation data into their assessment modeling research, whether for verifying loss function implementations, conducting calibration studies, or exploring how different archetypal students interact with innovative scoring mechanisms. The result is a robust, reproducible pipeline from simulation to analysis, strengthening the evidence base for the probabilistic approach to student assessment.*