Ind-inator: AI Guessing Game

*Course Project Report for CS 3346

Jay Jagdishkumar Bava
Computer Science
Western University
London, Canada
jbava@uwo.ca

Emily Jordan Berlinghoff
Computer Science
Western University
London, Canada
eberling@uwo.ca

Inderpreet Singh Doad
Computer Science
Western University
London, Canada
idoad@uwo.ca

Vidhi Joshi
Computer Science
Western University
London, Canada
vjoshi43@uwo.ca

***Abstract*—This project presents Ind-inator, an Akinator-style deductive guessing game that uses probabilistic inference and entropy-based question selection to identify a character the user is thinking of. The system supports graded answers ("Yes", "Probably Yes", "Maybe", "Probably No", and "No") and maps these to different likelihood strengths in a Bayesian update model. At each turn, the engine maintains a probability distribution over candidate characters and selects the next question using information gain combined with phase-based, decision-tree-style heuristics that prioritize broad franchise traits early and more specific traits later. A modular architecture consisting of a probabilistic reasoning engine, a game state manager, and a Flask-based web API connects to a React front end for interactive gameplay. Character, trait, and question data for 100 characters are generated and maintained programmatically via Python scripts, which produce 756 natural-language questions. Informal experiments show that the entropy-based question selector typically converges within a small number of questions, and the resulting system provides a flexible, extensible engine suitable for future datasets and feature extensions.**

***Index Terms*—Bayesian inference, information gain, decision-tree heuristics, question-answering, interactive systems, web API, React**

I. Introduction

Guessing-game systems such as Akinator showcase how an interactive system can identify a hidden entity using a sequence of questions. At each step, the system must decide which question to ask next and when it has enough confidence to make a guess. A naive strategy that asks questions at random is inefficient and leads to a poor user experience.

Ind-inator implements a similar character-guessing game but focuses on interpretable probabilistic reasoning, graded answers, and a modular web architecture. The engine maintains a probability distribution over a fixed set of characters and updates this distribution after each user response. Question selection is driven by entropy-based information gain combined with decision-tree-style heuristics reflecting the structure of the character–trait space.

The objectives are:

- Maintain and update a probability distribution over characters and guess the user's character in as few questions as possible.
- Support graded responses ("Probably Yes", "Probably No", etc.) in a Bayesian likelihood model.
- Select questions using information gain and phase-based heuristics rather than a fixed, precomputed decision tree.
- Expose the core AI engine via a Flask API and connect it to a React web front end.
- Generate and manage characters, traits, and questions programmatically for easy extension.

II. System Overview

The Ind-inator system comprises three major components:

A. Probabilistic Inference Engine

Implemented in `ai_engine.py`, the engine maintains:

- A probability distribution over all candidate characters.
- A mapping from questions to underlying traits indicating expected answers per character.
- An entropy-based scoring function to evaluate candidate questions.

At each turn:

1) The user answers the current question using one of five options (Yes, Probably Yes, Maybe, Probably No, No).
2) The answer is mapped to an internal code and likelihood parameters.
3) The engine applies a Bayesian-style update and renormalizes the distribution.

This yields an explainable selection strategy aligned with probabilistic reasoning while incorporating varying levels of user certainty.

B. Game State Manager

The `game.py` module manages the gameplay loop:

- Tracks which questions have been asked.
- Invokes the inference engine to update probabilities after each answer.
- Determines when the system is confident enough to guess (maximum character probability exceeds a threshold).
- Returns top-k candidate characters for debugging or UI display.

This layer orchestrates interaction between the front end and the inference engine.

C. Web API and Front End

The Flask server (`api_server.py`) exposes endpoints to:

- Start a new game and return the first question.
- Submit an answer and receive either the next question or a guess.
- Request a follow-up question after a wrong guess.
- Optionally record feedback about whether the final guess was correct.

A React front end (`frontend/`) consumes this API to provide an Akinator-style web experience. There is no desktop GUI in the submitted build; all interaction is through the React client and the Flask API.

III. Methods

A. Dataset Processing

Character attributes and question definitions live under `data/` and are constructed by scripts in `scripts/`. The pipeline:

- **Character definition:** 100 characters spanning franchises (anime, movies, video games, television) with uniform priors.
- **Trait assignment:** Category scripts assign traits (franchise, identity, appearance, abilities, role, archetype, personality). These are merged into a flattened binary mapping (`traits_flat.json`) such as `franchise_anime`, `appearance_color_orange`, `abilities_magic_user`.
- **Question generation:** `generate_questions.py` converts trait keys into natural-language questions, producing `questions.json` with 756 items.

Because data are generated programmatically, extending the set of characters or traits requires only script updates and regeneration.

B. Entropy-Driven Question Selection

For each unasked question q, the engine estimates expected information gain. Let P be the current distribution with entropy $H(P)$. Approximating answers as yes-like/no-like based on trait presence and frequencies:

$$IG(q) = H(P) - \left[P(\text{Yes})\, H(P_{\text{Yes}}) + P(\text{No})\, H(P_{\text{No}})\right],$$

where P_{Yes} and P_{No} are approximate posteriors. The engine selects the question with maximum $IG(q)$ and then applies phase-based adjustments (early: franchise/media traits; mid: series/visual traits; late: ability/role traits) and frequency-based boosts for discriminative traits.

C. Graded-Answer Likelihood Model

Five buttons are supported: Yes, Probably Yes, Maybe, Probably No, No. Mapping:

- **Yes:** strong positive signal for trait-bearing characters (high likelihood if trait present; low if absent).
- **Probably Yes:** moderate positive signal (narrower likelihood gap).
- **No:** strong negative signal (likelihoods reversed relative to Yes).
- **Probably No:** moderate negative signal.
- **Maybe:** "unknown" (no update to P).

For character c and question q with trait $t(q)$:

$$P(c \mid \text{answer}) \propto P(\text{answer} \mid c)\, P(c),$$

where $P(\text{answer} \mid c)$ depends on whether c has $t(q)$ and on the answer type. Probabilities are normalized after each update.

IV. Experimental Setup

We evaluated the system as follows:

- Initialized the engine with the 100-character dataset, flattened trait vectors, and the 756 generated questions.
- Conducted games by selecting target characters and answering either manually according to true traits or via a scripted simulator.
- Allowed the engine to ask questions until a confidence threshold (e.g., 0.85) triggered a guess or a maximum number of questions was reached.
- Tracked:
  - Mean number of questions to reach the confidence threshold.
  - Whether the final guess matched the target.
  - Qualitative transitions in question types (broad $\rightarrow$ specific).

Experiments ran on a standard laptop with Python 3.10 and no GPU.

V. Results

We evaluated Ind-inator under two response models: (1) an ideal user who always answers each question correctly with deterministic yes/no responses, and (2) a human-like user who answers using the full five-button scale (Yes, Probably Yes, Maybe, Probably No, No), with injected uncertainty intended to simulate hesitation, partial knowledge, or inconsistent recall. For both settings, we ran 100 games, one for each character in the dataset.

A. Ideal User (Perfect Oracle)

Under ideal conditions, the system achieved near-perfect performance, correctly identifying 99 out of 100 characters, corresponding to an accuracy of 99.00%. The single failure occurred when Ron Weasley was incorrectly classified as President Snow, reflecting a sparse trait overlap for certain character pairs.

The average number of questions required to reach the confidence threshold (0.85) was 10.87. As shown in Fig. 3, most games required between 7 and 14 questions, with some extending toward the upper bound of 25. The violin plot in Fig. 5 further illustrates the distribution's narrow concentration around the median.

The ideal accuracy summary is shown in Fig. 1. The probability convergence curve in Fig. 7 demonstrates stable, monotonic growth of the top candidate's probability, with rapid convergence compared to the human-like setting.

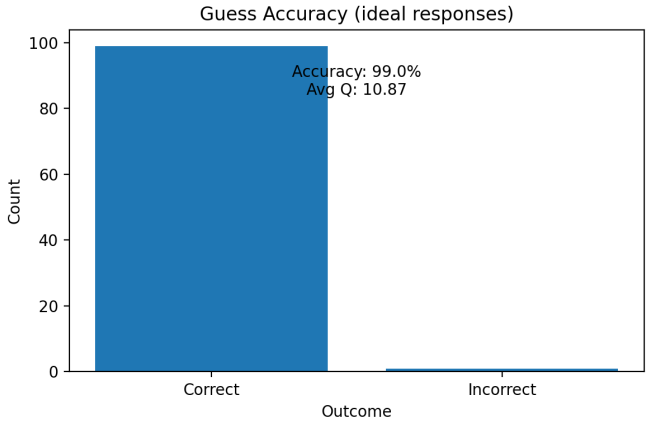

Fig. 1. Guess accuracy and average number of questions under the ideal user model (100 games).

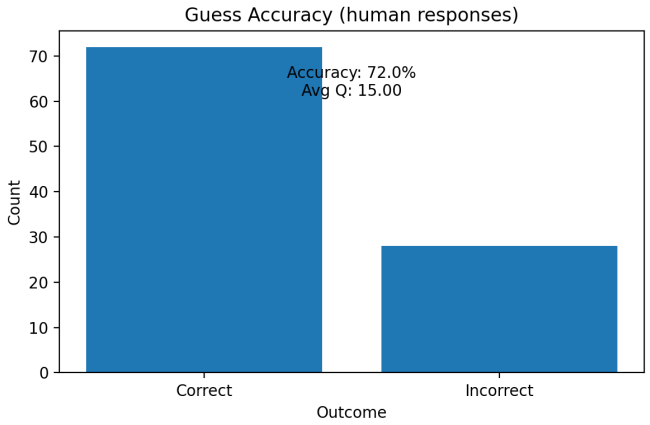

Fig. 2. Guess accuracy and average number of questions under the human-like response model (100 games).

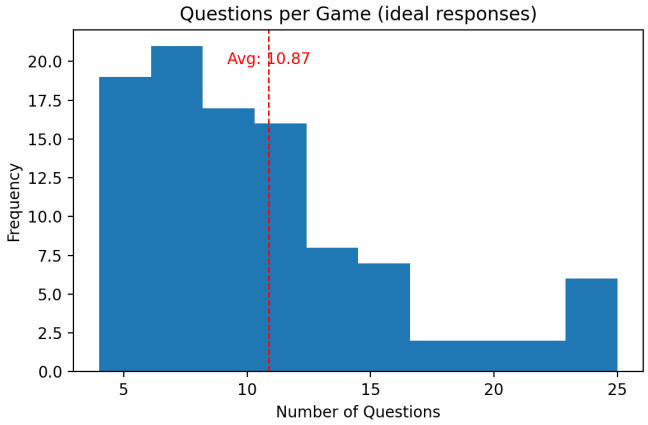

Fig. 3. Distribution of question counts under the ideal response model. The red dashed line marks the mean (10.87).

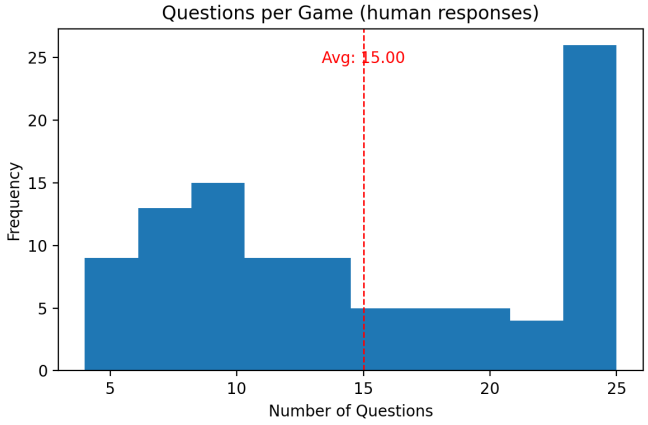

Fig. 4. Distribution of question counts under the human-like response model. The red dashed line marks the mean (15.00).

Because only one error occurred, the ideal confusion matrix in Fig. 9 consists of a single cell corresponding to Ron Weasley as President Snow.

B. Human-Like Responses With Graded Uncertainty

Introducing uncertainty significantly challenged the model, reducing accuracy to 72.00% (72 out of 100 games). The average number of questions rose to 15.00, reflecting slower probability accumulation and the disruptive effects of "Maybe" and "Probably No" responses.

The question distribution appears in Fig. 4, and the violin plot in Fig. 6 shows substantially higher variance than in the ideal case. The accuracy summary is provided in Fig. 2.

The probability convergence curve in Fig. 8 highlights significantly noisier progression compared to the ideal model, with oscillations and plateaus caused by uncertain answers.

The confusion matrix of incorrect predictions, shown in Fig. 10, reveals several systematic errors. For example, multiple unrelated characters were misclassified as Albus Dumbledore, indicating that the current trait taxonomy insufficiently distinguishes certain mentor-archetype characters. Other repeated confusions include Aang being predicted for multiple unrelated characters, suggesting trait sparsity issues and ambiguity in overlapping universes.

C. Comparison

Table I summarizes the performance gap between the two response models. These results show that Ind-inator performs extremely well when answers are consistent and reliable, but accuracy decreases when responses contain uncertainty. This suggests several possible improvements, including enhanced modeling of answer noise, expanded trait sets for ambiguous franchises, and stronger regularization during probability updates.

TABLE I
PERFORMANCE COMPARISON BETWEEN RESPONSE MODELS

Model	Accuracy	Avg. Questions
Ideal (perfect answers)	99.00%	10.87
Human-like (graded answers)	72.00%	15.00

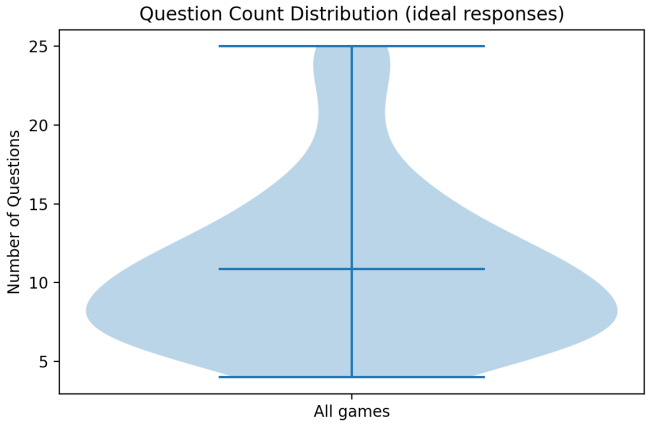

Fig. 5. Violin plot of the question-count distribution for ideal responses.

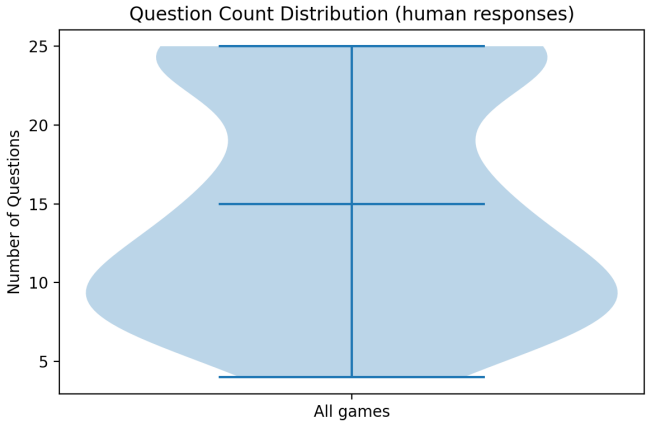

Fig. 6. Violin plot of the question-count distribution for human-like responses.

VI. Conclusion and Future Work

Ind-inator demonstrates that a combination of probabilistic inference, graded answers, and entropy-based, decision-tree-style question selection can support an effective and interpretable character-guessing game. The script-based data pipeline for characters, traits, and questions simplifies maintenance and extension of the dataset, and the modular design allows the same engine to drive both web and desktop interfaces.

Future work may include:

- Expanding the character set and trait taxonomy to reduce ambiguity among similar characters.
- Adding more fine-grained traits and refining questions for known difficult cases.
- Implementing more systematic evaluation, including large-scale automated simulations with quantitative performance statistics.
- Exploring alternative or complementary question-selection strategies, such as clustering-based approaches or precomputed decision trees for particular subsets of characters.

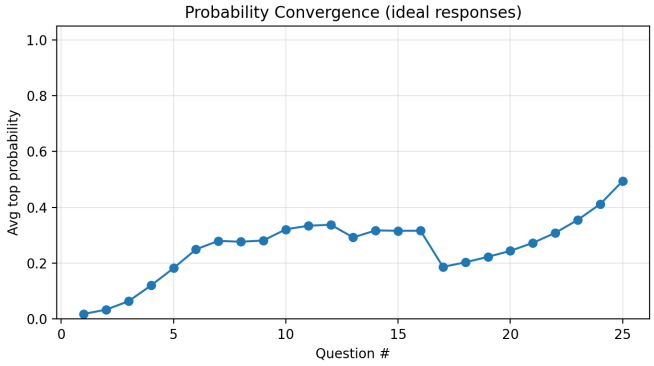

Fig. 7. Average convergence of the top candidate probability for ideal responses. Convergence is smooth and rapid.

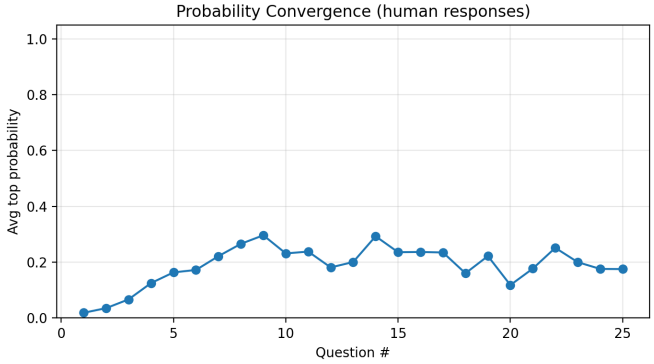

Fig. 8. Average convergence of the top candidate probability for human-like responses. Noise and slower convergence are clearly visible.

VII. Contribution

Although all team members collaborated on the overall design, implementation, debugging, and testing of Ind-inator, certain areas had primary leads.

Jay Jagdishkumar Bava took a leading role on the backend, including game state management, integration of the AI engine with the Flask API, handling probability-update flows, and contributing to debugging and performance tuning.

Inderpreet Singh Doad also led backend work, including the design of the probabilistic inference approach, the entropy-based and phase-based question-selection logic, the data and question-generation scripts, and the integration of the AI engine into both the backend and user interfaces, as well as preparation of the written report.

Emily Jordan Berlinghoff focused primarily on the React front-end, API integration, and UI/UX design, while also contributing to backend discussions, testing, and debugging to ensure consistent behaviour across the stack.

Vidhi Joshi focused on character and trait definition, validation of generated questions, and manual testing, while also participating in design decisions and providing feedback on both backend behaviour and front-end interaction flows.

All team members participated in design discussions, implementation decisions, and testing, and adhered to the course

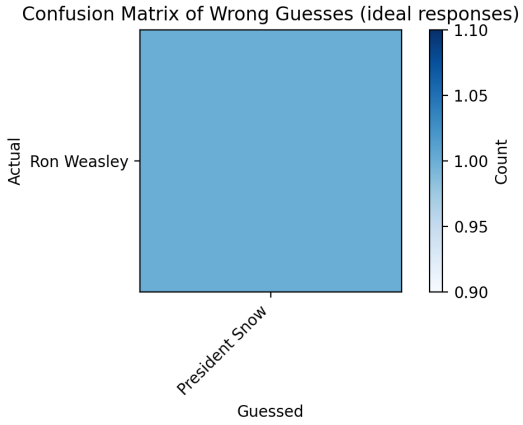

Fig. 9. Confusion matrix of incorrect guesses under the ideal model. Only one misclassification occurred: Ron Weasley as President Snow.

expectations for professionalism and academic integrity.

REFERENCES

[1] Akinator, "Akinator the Web Genius," 2024. [Online]. Available: https://en.akinator.com/

[2] DeepLearning.AI, "AI Education and Learning Resources," 2024. [Online]. Available: https://www.deeplearning.ai/

[3] A. Ronacher, "Flask (Python Web Framework)," 2010. [Online]. Available: https://flask.palletsprojects.com/

[4] Google, "TensorFlow: An end-to-end open source machine learning platform," 2024. [Online]. Available: https://www.tensorflow.org/

[5] Hugging Face, "Hugging Face: Open-source AI models and tools," 2024. [Online]. Available: https://huggingface.co/

[6] D. Mackay, *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003. Available online: https://www.inference.org.uk/mackay/itila/

[7] C. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008. Available online: https://nlp.stanford.edu/IR-book/

[8] MIT, "Introduction to Deep Learning," 2024. [Online]. Available: https://introtodeeplearning.com/

[9] Meta AI, "PyTorch: An open source machine learning framework," 2024. [Online]. Available: https://pytorch.org/

[10] Meta Platforms, "React: A JavaScript Library for Building User Interfaces," 2024. [Online]. Available: https://react.dev/

[11] scikit-learn Developers, "scikit-learn: Machine Learning in Python," 2024. [Online]. Available: https://scikit-learn.org/

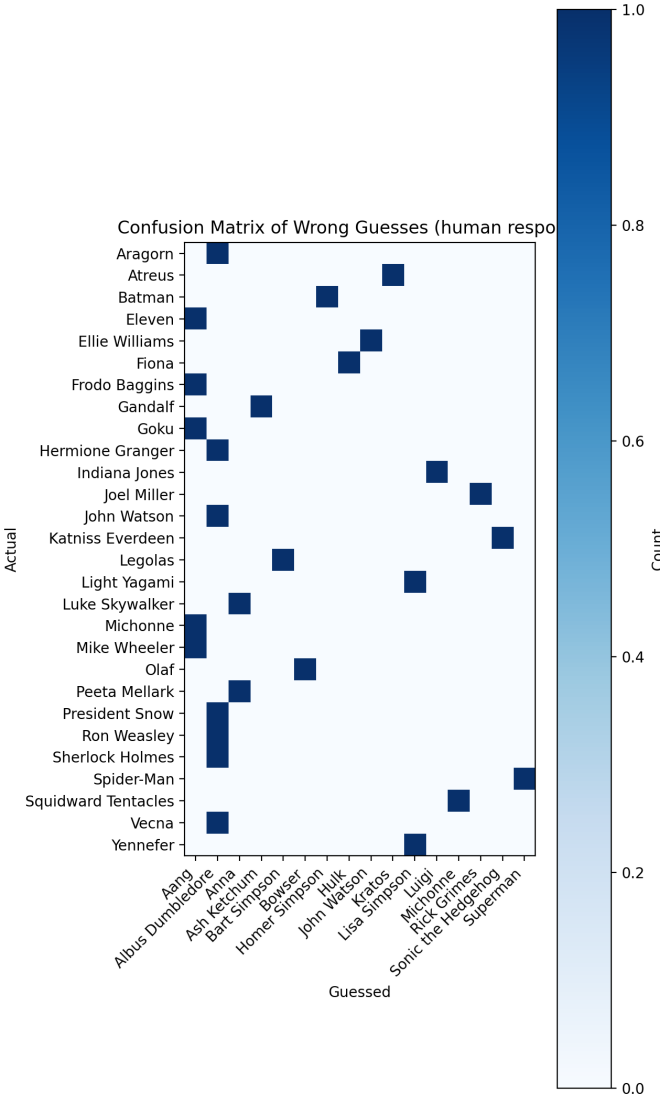

Fig. 10. Confusion matrix of incorrect guesses under the human-like model. Darker cells indicate repeated confusions between specific character pairs.