THE 20TH EDITION OF THE INTERNATIONAL CONFERENCE
EUROPEAN INTEGRATION
REALITIES AND PERSPECTIVES

Technology in Education

Assisting Generative AI Processes Based on Automatic Reasoning

Radu Bucea Manea Tonis[1]

Abstract: Generative AI processes can be significantly improved by incorporating automatic reasoning techniques, such as the MiniMax algorithm, commonly employed in decision-making contexts like game theory. Functional programming prioritizes immutability and first-class functions, while logic programming centers around declarative problem-solving through the use of rules and facts. Also, AJAX requests are essential for facilitating asynchronous data exchange between the client and server, enabling real-time updates and smooth interactions in AI-driven applications.

Keywords: AI; MiniMax; AJAX

JEL Classification: technological change and the knowledge-based economy (O3)

1. Introduction

Integrating automatic reasoning with generative AI processes represents a revolutionary synergy, harnessing the strengths of both fields to improve efficiency and accuracy. Generative AI excels at producing content across various modalities but often requires structured guidance to ensure coherence and relevance (Sengar et al., 2024). Automatic reasoning facilitates logical inference and decision-making through the analysis of structured data and rules (Wu et al., 2025) By embedding reasoning mechanisms within generative AI systems, these processes can dynamically adapt to complex tasks, ensuring that outputs align with predefined goals and constraints. This fusion promotes advancements in areas such as automated content creation, intelligent problem-solving, and adaptive learning environments, according to (Kabudi et al., 2021). Future research should focus on developing frameworks for seamless integration, enhancing the scalability of reasoning algorithms, and addressing ethical considerations to realize the potential of this innovative collaboration fully.

[1] Senior Lecturer, Danubius International University, Galati, Romania, Address: 3 Galati Blvd, Galati, 800654, Romania, Corresponding author: radumanea@univ-danubius.ro.

Copyright: © 2025 by the authors.
Open access publication under the terms and conditions of the
Creative Commons Attribution-NonCommercial (CC BY NC) license
(https://creativecommons.org/licenses/by-nc/4.0/)

2. Types of Generative AI Models

2.1. Generative Adversarial Networks (GANs)

GANs are a class of machine learning frameworks introduced by Ian Goodfellow and his colleagues in 2014 (Arthur, 2019). They consist of two neural networks - the Generator and the Discriminator - that engage in a competitive process. The Generator aims to create data that closely resembles the real data distribution such as generating realistic images and attempts to deceive the Discriminator by producing outputs that are indistinguishable from genuine samples. Conversely, the Discriminator serves as a judge, distinguishing between real data and the synthetic data generated by the Generator. It provides feedback to the Generator, allowing it to enhance its output over time, after (Mills, 2023). The two networks are trained simultaneously within a zero-sum game framework, where the Generator refines its ability to produce realistic data as the Discriminator sharpens its ability to identify fake data.

GANs have transformed various fields, particularly in image generation and enhancement. They are capable of creating highly realistic images, altering image styles, improving resolution, generating lifelike video sequences, and producing synthetic datasets for training machine learning models (de Souza et al., 2023). However, GANs can also be employed to create deepfakes, which raises significant concerns about potential misuse, according to (Preeti et al., 2023). In summary, GANs illustrate the efficacy of adversarial training and have led to many innovations across industries. Nonetheless, addressing both ethical and technical considerations remains essential for their responsible use and further advancement.

2.2. Autoregressive Models (AR)

AR models are a fundamental concept in machine learning and statistics. They forecast future values of a sequence by leveraging its past values through a linear dependency. These models are commonly applied in various tasks involving sequential data, such as time series analysis and natural language processing. In an AR model, each value in the sequence is represented as a weighted sum of its preceding values, accompanied by a noise term (Gandhi, 2015):

$$X_t = \phi_1 X_{t\text{-}1} + \phi_2 X_{t\text{-}2} + \ldots + \phi_p X_{t\text{-}p} + \epsilon_t \quad [1]$$

where:

X_t: Current value of the sequence;

ϕ_i: Coefficients (weights) for previous values;

p: Order of the model (number of past values considered);

ϵ_t: Noise or error term.

In Machine Learning, autoregression (AR) involves predicting the next word or character in a sequence. Models such as GPT employ autoregressive methods to generate coherent text. Additionally, these models are utilized to forecast stock prices, weather patterns, and other types of sequential data (Dat, 2024). In summary, autoregressive models serve as a foundation for sequence modeling and have paved the way for numerous advanced techniques in modern AI. They continue to evolve and adapt, making significant contributions across various fields, including natural language processing, signal processing, and more.

2.2.1. Variational Autoencoders (VAEs)

VAEs are a class of neural networks utilized in unsupervised learning, particularly for generative modeling, and unlike traditional autoencoders that deterministically compress and reconstruct data, VAEs introduce a probabilistic approach (Mienye, 2025). They map input data into a latent space by representing it as a distribution, typically Gaussian, rather than as a single point. This probabilistic framework enables them to generate new data samples by sampling from the latent space and decoding these samples back into data that resembles the original input (Jiang, 2025). VAEs are extensively employed in various applications, including image generation, anomaly detection, and data compression.

2.2.2. Diffusion Models

While VAEs utilize a probabilistic latent space and DALL·E employs transformers for text-to-image generation, diffusion models emphasize a process of iterative refinement. According to (Guo, 2025), these models convert random noise into structured outputs through a series of steps, in contrast to directly sampling from a distribution as seen in VAEs. Latent diffusion models compress images into a latent space, enabling faster and more efficient generation—a principle that aligns with DALL·E's objective of producing high-quality images.

Developed by OpenAI, DALL·E is an innovative AI model created to generate images from textual descriptions. It employs a variant of the transformer architecture, seamlessly integrating elements of natural language processing with image synthesis. DALL·E is capable of producing unique and highly imaginative visuals based on prompts such as "a futuristic cityscape made of candy" or "an astronaut riding a horse in space." Utilizing transformers for image synthesis and incorporating concepts from latent diffusion models such as Stable Diffusion (Abideen, 2023), DALL·E has gained recognition as a leading example of next-generation text-to-image models, showcasing how generative models can effectively bridge the gap between language and visual artistry.

The following challenges and ethical considerations are to be taken into account:

- Bias in training data - models can inadvertently replicate biases in their training data, leading to harmful or unfair outputs.
- Misuse potential - AI-generated content can be exploited for creating deepfakes, misinformation, or plagiarized materials.
- Copyright issues - ensuring outputs respect intellectual property laws remains a key challenge.

3. Methodology

We have chosen to develop a Tic Tac Toe gaming web application[1] utilizing the MiniMax algorithm Prolog implementation from[2], which applies to zero-sum games such as Tic Tac Toe and chess. Initially, the application generates a tree encompassing all possible moves. It then evaluates each move, aiming to maximize the chances of victory for one player while minimizing the chances for the opponent. By comparing scores from the bottom up within the decision tree, it selects the optimal next move. Ultimately, the application implements the most suitable winning strategy, as illustrated in Fig. 1.

[1] radubm1/TicTacToe at ReactTicTac.
[2] https://github.com/jaunerc/minimax-prolog.

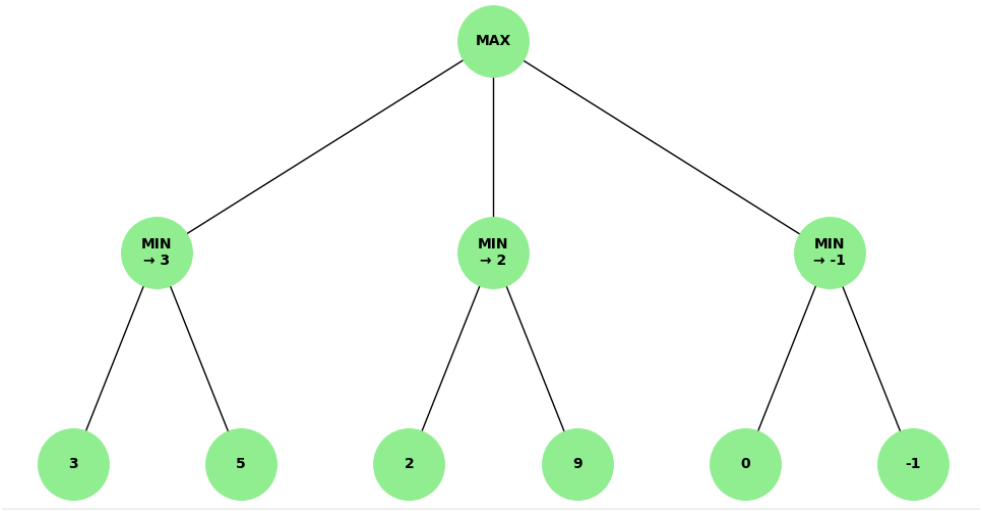


Figure 1. Decision tree based on determining the best possible move, after (Bratko, 1990)

This React-based code creates a straightforward game application that allows users to engage with a tic-tac-toe board. We opted for functional programming due to its focus on immutability, pure functions, and higher-order functions (Shahu, 2024). Additionally, the app incorporates a backend API to facilitate specific operations, such as retrieving updated board states. Below is an overview of its components and functions:

Imported Libraries and Components

- `isomorphic-fetch`: A fetch polyfill to make HTTP requests for both client and server environments.

- `React`: The core library used for building UI components.

- `Row` & `GameList`: Custom React components representing individual rows of the game board and a list of games, respectively.

Functions

1. `getInitialState()`

   - Returns the initial state of the game, including:

     - `rows`: A 3x3 matrix representing the board.

     - `turn`: Tracks the current player's turn (`o` by default).

     - `winner`: Indicates the game's winner.

     - `board`: A representation of the game board in a linear format.

2. `checkWin(rows)`

   - Checks if there is a winner by evaluating possible winning combinations.

   - Winning combinations are defined as indices of rows, columns, or diagonals.

3. `getData(arr)`

   - Makes a POST request to the backend API (`http://localhost:8080/minimax`) to fetch a new board state.

   - Processes the response and integrates the updated board into the current state.

4. `transformTo2DArray(arr)`

- Converts a one-dimensional array (linear board) into a two-dimensional matrix for easier display and manipulation.

App Component's constructor initializes the app state using the `getInitialState()` function. The next internal functions are the following:

`handleClick(row, square)`:

- Handles click events on the game board squares:

- Updates the board and rows based on the current player's move.

- Checks if a winner has been determined using `checkWin()`.

- Adjusts the state accordingly.

`render()`:

- Returns the UI representation of the application:

- Displays the game board (composed of individual `Row` components).

- Shows the current turn or winner information.

- Includes buttons for resetting the board or fetching the updated board state.

For the back-end component, we have selected the Prolog logic programming language to facilitate rule-based reasoning through the MiniMax algorithm. The advantages of employing functional and logic programming stem from their blend of flexibility and precision, all enabled by the use of the Reactide IDE [1], as demonstrated in Fig. 2.

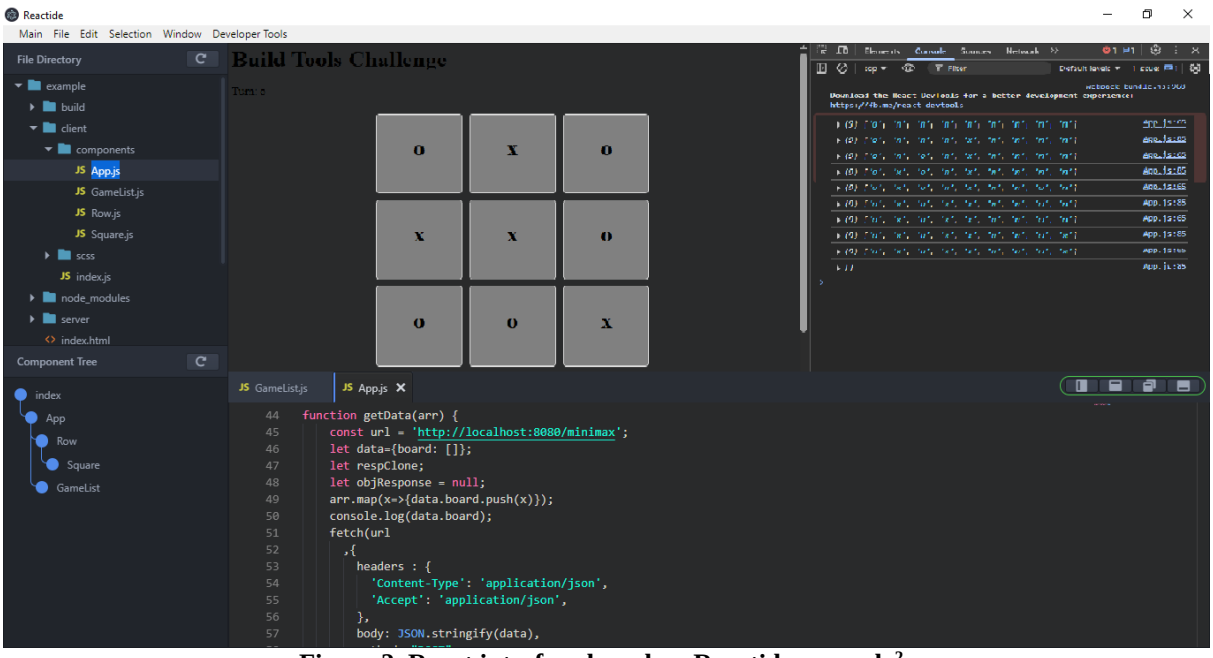


Figure 2. React interface based on Reactide example[2]

[1] https://github.com/reactide/reactide

[2] https://github.com/reactide/reactide/tree/master/example

4. Conclusion

In conclusion, our application showcases a harmonious blend of techniques, such as the MiniMax algorithm and AJAX requests, and incorporates several key features for effective problem-solving:

- Dynamic interaction, enabling real-time engagement on the tic-tac-toe board;

- Backend integration, allowing communication with a backend API to retrieve updated board states;

- State management, efficiently updating the board, player turns, and winners through React state.

Looking ahead, potential advancements will focus on:

- Improved optimization algorithms;

- Advanced frameworks for enhanced integration;

- Smarter, more interactive generative AI models.

References

Abideen, Z. (2023). *How OpenAI's DALL-E works?* Retrieved from https://medium.com/@zaiinn440/how-openais-dall-e-works-da24ac6c12fa.

Arthur I. M. (2019). 10 Ian Goodfellow's Generative Adversarial Networks: AI Learns to Imagine. In *The Artist in the Machine: The World of AI-Powered Creativity*. MIT Press.

Bratko, I. (1990). *PROLOG Programming for Artificial Intelligence* (2nd ed.). Addison-Wesley Longman Publishing Co., Inc.

Dat, M. (2024). *StockGPT: A GenAI Model for Stock Prediction and Trading*. Retrieved from https://arxiv.org/html/2404.05101v1.

de Souza, V. L., Marques, B. A. D., Batagelo, H. C., & Gois, J. P. (2023). A review on Generative Adversarial Networks for image generation. *Computers & Graphics*, 114, 13-25.

Djaghouri, B. (n.d.). *Reactide is the first dedicated IDE for React web application development*. Gift Hub. Retrieved from https://github.com/reactide/reactide.

Gandhi, V. (2015). Interfacing Brain and Machine. In V. Gandhi (Ed.), *Brain-Computer Interfacing for Assistive Robotics*.

Guo, Z., Lang, J., Huang, S., Gao, Y., Ding, X. (2025). *A Comprehensive Review on Noise Control of Diffusion Model*. Retrieved from https://arxiv.org/html/2502.04669v1.

Jauner, C. (n.d.). *Minimax-prolog*. Gift Hub. Retrieved from https://github.com/jaunerc/minimax-prolog.

Jiang, Y., Ma, X., & Li, X. (2025). Towards virtual sample generation with various data conditions: A comprehensive review. *Information Fusion*, 117.

Kabudi, T., Pappas, I., & Olsen, D. H. (2021). AI-enabled adaptive learning systems: A systematic mapping of the literature. *Computers and Education: Artificial Intelligence*.

Manea, R. (n.d.). *TicTacToe*. Gift Hub. Retrieved from https://github.com/radubm1/TicTacToe/tree/ReactTicTac.

Mienye, I. D., Swart, T. G. (2025). Deep Autoencoder Neural Networks: A Comprehensive Review and New Perspectives. *Archives of Computational Methods in Engineering*.

Mills, O. (2023). *Inside GANs: Understanding the duel of generator and discriminator*. Retrieved from https://oliviermills.com/articles/inside-gans-understanding-duel-generator-discriminator.

Preeti, Kumar, M., & Sharma, H. K. (2023). A GAN-Based Model of Deepfake Detection in Social Media. *Procedia Computer Science*, 218, 2153-2162.

Sengar, S. S., Hasan, A. B., Kumar, S. (2024). *Generative artificial intelligence: a systematic review and applications*. Multimed Tools Appl.

Shahu, A. (2024). *Mastering the Art of Functional JavaScript: Immutability, Pure Functions, and Beyond*. Retrieved from https://blogs.perficient.com/2024/03/04/mastering-the-art-of-functional-javascript-immutability-pure-functions-and-beyond/.

Umar, K. (n.d.). *Reactide example*. Gift Hub. Retrieved from https://github.com/reactide/reactide/tree/master/example.

Wu, J., Zhu, J., & Liu, Y. (2025). *Agentic Reasoning: Reasoning LLMs with Tools for the Deep Research*. arXiv [Cs.AI]. Retrieved from http://arxiv.org/abs/2502.04644.