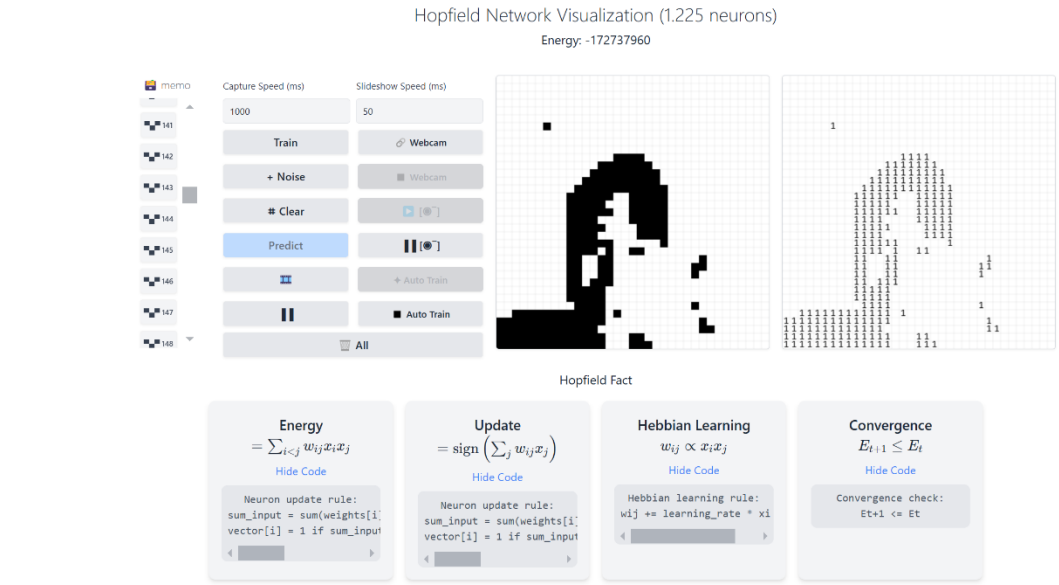
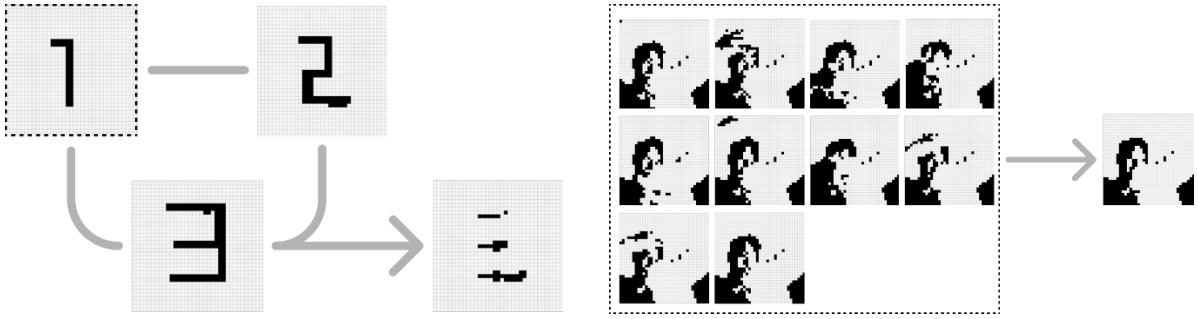
Hopfield Network Visualizer: Exploring Associative Memory and Neural Computations

By Riki Awal Syahputra | L1AC





1. Plagiarism/Cheating

BINUS International seriously regards all forms of plagiarism, cheating, and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity, and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

2. Declaration of Originality

By signing this assignment, I understand, accept, and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student:

Riki Awal Syahputra

Table of Contents

1. Plagiarism/Cheating 2
2. Declaration of Originality 2
I. Introduction 5
   A. Background 5
II. Problem Identification 6
   A. Problem Statement 6
III. Project Objectives 7
   A. Main Objectives 7
   B. Technical Objectives 7
IV. Solution Design 8
   A. System Architecture 8
   B. Key Features 8
      1. Pattern Training: 8
      2. Pattern Retrieval: 9
      3. Visualization: 9
      4. Flowchart 10
V. Implementation 11
   A. Code Overview 11
      1. Frontend Implementation: 11
      2. Backend Implementation: 11
      3. Webcam Input Solution 12
      4. Performance Optimization Approaches 12
      5. Error Handling and Robustness: 13
   B. Input Validation & Error Handling 13
VI. Evidence of Working Program 15
   A. Screenshots 15
   B. Demo 17
VII. Future Enhancements 18

Table of Figures

Figure 1: Hopfield Network Visualizer Flowchart 10
Figure 2: Webcam Based Rendering & Auto Train 15
Figure 3: Manual Based Rendering & Training 16
Figure 4: Noise Feature 16
Figure 5: Noise Resistant Capabiliy 16
Figure 6: Pattern Recognition Capability 17
Figure 7: Energy Calculation (always below zero) 17

I. Introduction

A. Background

The area of Hopfield Networks begins with the background of artificial neural network studies. Originally introduced by John Hopfield in the early 1980s, the network revolutionized computational neuroscience by conceptually explaining the process of memory storage and retrieval while using the brain as a distributed neural processor. In other words, it represents a new mechanism for modeling a process of associative memory and pattern recognition; hence, it allows them to simulate how biological systems store and recall information. Applications range from solving optimization problems such as the traveling salesman problem and image reconstruction to signal processing noise reduction. On the other hand, their properties of convergence to stable states have been used in combinatorial optimization and in associative databases where matching and retrieval of patterns require high speed. The Hopfield network, invented by a physicist named John Hopfield in the early 1980s, is a computational collective machine that mimics certain aspects of biological memory systems, using principles from distributed information storage and energy minimization, which converges to stable states corresponding to a pattern learned.

But what makes the Hopfield Networks so important is their modeling of emergent phenomena by such a modest yet strong interaction among neurons, connecting the abstract mathematical theory with practical computational applications. This project will somewhat demystify these networks by providing an interactive visualization platform to explore the complex dynamics within. As such, this allows one-a researcher or a student-to practically see Hebbian learning, energy landscapes, noise tolerance, and more.

What this work mainly does is transform abstract theories into a tangible framework; it is developing academic knowledge while opening new horizons for innovative applications in computational neuroscience and artificial intelligence.

II. Problem Identification

A. Problem Statement

Despite their elegance and theoretical importance, Hopfield Networks are usually inaccessible to learners because of their abstract mathematical formulation. Most of the currently available learning tools cannot provide enough interactivity and direct feedback to understand, in real time, the emergent properties of this kind of network, such as the convergence to a stable state in reaction to noisy inputs. For example, TensorFlow Playground allows interaction with visualizations but doesn't support special modules that can illustrate the dynamics of Hopfield Networks-how, for example, it achieves convergence through the process of energy minimization. This aspect is very important in the definition of Hopfield Networks; otherwise, one will not get an intuition about how such a network actually recalls a pattern in a reliable way, even under noise conditions. Following that, noise tolerance testing or state updates in real-time are commonly missing features in most stand-alone simulators of neural networks. This makes the latter less pedagogically useful. Classic static visualizations, typically taken from textbooks or simple, non-interactive simulations performed using stand-alone software, cannot represent real-time updates of neurons or changes in the energy state. In contrast, visualizers that either use TensorFlow Playground or create their own visualization of the neural networks offer interactive models, though only a few of them go into detailed dynamics specific to Hopfield Networks. This, in a sense, creates a discord in complete understanding and, subsequently, wider acceptance and deeper understanding of such systems, especially in computational neuroscience, where their applications are of high importance.

This work tries to overcome these drawbacks by proposing a complete visualization platform for the real-time exploration of Hopfield Networks, shrinking the gap between theory and practice as much as possible.

III. Project Objectives

A. Main Objectives

- Design a state-of-the-art, Web-based visualization platform for the exploration of Hopfield Neural Networks.
- Real time demonstration of pattern storage, retrieval, and energy dynamics.
- Design an intuitive interface that will make engagement and access easy for advanced learners and researchers.

B. Technical Objectives

1. Create a 35x35 neuron lattice and run associative memory experiments.
2. Flexibility in input provides: manual pattern drawing or automatic web-cam-based pattern pickup.
3. Develop robust algorithms for pattern memorization, retrieval, and testing of resilience to noise simulations.
4. Advanced visualization. It will provide dynamic energy state trajectories and interactive neuron updates that enhance user experience. Features are supporting particular learning goals; for example, showing how energy states evolve to stable configurations concretizes the abstract notion of energy minimization. Interactive neuron updates have the added benefit of allowing researchers to experiment with runtime pattern adjustments and their influence on network behavior, furthering both the understanding and the usability of the system. This provides intuitive observation of the network going toward stable states and gives the user insightful understanding of the dynamics of associative memory. Real-time feedback gives the visualization theoretical learning combined with practical insight-something particularly useful within an educational or research context.

IV. Solution Design

A. System Architecture

1. **Frontend**: The interface is designed with React.js. For high scalability and performance, Vite is used, whereas for styling, Tailwind CSS and for modern responsivity, DaisyUI is used.
2. **Backend**: Python, through the Flask framework, gives the computation backbone to the system by doing the heavy lifting of actual data processing and implementation of algorithms.
3. **Communication**: RESTful APIs will then help the frontend and backend communicate seamlessly, even possibly in real time.
4. **Drawing**: HTML5 Canvas enabled high-performance drawing for dynamic updates on neuron grid updates with mapping.
5. **Math Rendering**: The following section allows the typesetting of mathematical equations using KaTeX to develop better intuitions into the mathematics working behind this network.

B. Key Features

1. Pattern Training:

   a) Intuitive manual drawing with high precision in manipulating neuron states.

   b) The webcam-based input captures and processes live patterns for adaptive training demonstrations, which present seamless alternatives to manual input. Manual inputs require much precision and may be time-consuming, especially when dealing with complex patterns. Contrasts of this solution, however, make the process much easier because such an interface allows for fast and accurate pattern acquisition with much less user effort. It greatly enhances the usability to such an extent that users can input patterns quickly in real time and need not draw the exact pattern manually. Moreover, it allows changing the pattern dynamically, making the system much more flexible for exploring different scenarios in pattern recognition and recall.

2. Pattern Retrieval:

    a) Recall mechanisms are provided that enable the user to recover one or more patterns from the network, demonstrating associative memory.
    b) The added noise tests pattern recognition for robustness against different degrees of disruption.

3. Visualization:

    a) The activity of neurons and their transitions are visualized on a grid that refreshes in real time.
    b) Energy plots are provided as dynamic updates of the trajectory which the network follows to stabilize into static configurations.

4. Flowchart

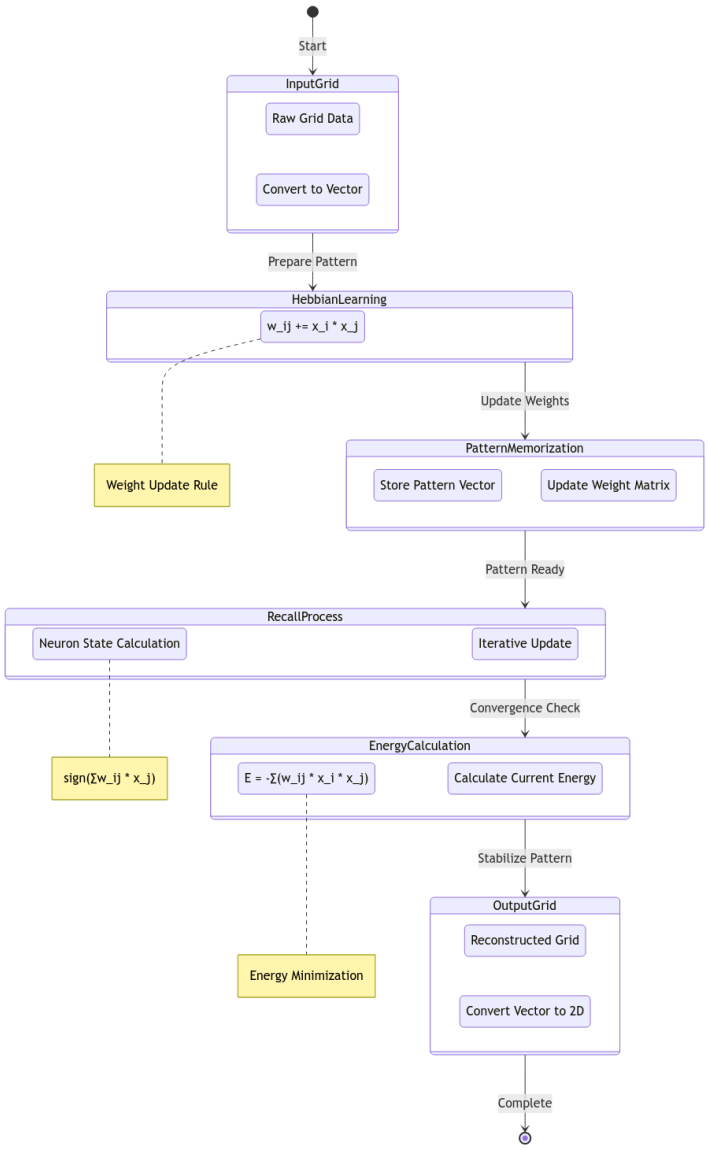


Figure 1: Hopfield Network Visualizer Flowchart

V. Implementation

A. Code Overview

1. Frontend Implementation:
   a) **Interactive Grid**: An HTML5 Canvas on which the neuron grid is displayed - one can draw patterns on it interactively. It immediately shows the update in real time.
   b) **Webcam Integration**: Frames of video coming from a webcam are processed through JavaScript and translated into binary states of neurons for training.
   c) **Dynamic Formula Rendering**: Important equations are directly shown in the interface using KaTeX, relating theoretical notions to practical observations made.
2. Backend Implementation:
   a) **Data Structure**: Patterns are encoded as one-dimensional vectors for efficient storage and computation.
   b) **Learning Algorithm**: Weight matrix updated by Hebbian learning with the formula: Computationally, this looks something like the following:

```
wij += xi * xj
```

c) **Neuron State Update**: Neurons are iteratively updated using the rule:

Implemented in code as:

```
sum_input = sum(weights[i][j] * vector[j] for j in
range(len(vector)))
 vector[i] = 1 if sum_input > 0 else -1
```

d) **Energy Minimization**: The network minimizes energy with each iteration, converging to a stable state. The energy function is defined as:

Computationally, this is represented as:

```
energy = 0
for i in range(len(vector)):
    for j in range(i + 1, len(vector)):
        energy -= weights[i][j] * vector[i] *
vector[j]
```

3. Webcam Input Solution

a) **Image Processing Strategy**: The webcam integration solves pattern input challenges through:

- Real-time video frame capture
- Grayscale conversion
- Binary thresholding
- 35x35 pixel grid transformation

Conversion Algorithm:

```
def process_webcam_frame(frame):
    grayscale = cv2.cvtColor(frame,
cv2.COLOR_BGR2GRAY)
    resized = cv2.resize(grayscale, (35, 35))
    binary = (resized > 128).astype(int) * 2 - 1
    return binary
```

4. Performance Optimization Approaches

a) **Computational Efficiency**:

- Vectorized NumPy operations
- Minimal matrix computations

- Cached weight calculations
- Efficient state update mechanisms

b) **Memory Management**:

- Circular buffer for pattern storage
- Constant-time pattern retrieval
- Bounded memory consumption

5. Error Handling and Robustness:

a) **Noise Tolerance Mechanism**: Implemented a configurable noise introduction method:

```
def add_noise(pattern, noise_level=0.1):
    noisy_pattern = pattern.copy()
    mask = np.random.random(pattern.shape) <
noise_level
    noisy_pattern[mask] *= -1
    return noisy_pattern
```

B. Input Validation & Error Handling

1. **Broad Input Data Validation**: Ensures Integrity—Manual and Automated Pattern Submissions Are Supported. Error validation for robustness may include things like validation of errors such as invalid grid dimensions, wrong binary pattern formatting, or incomplete webcam data. It makes for a very protective computational process and a fluent user experience without constant disturbance on account of faulty input data. Standard errors dealt with include invalid grid dimensions, incorrect binary pattern formatting, and incomplete data from captures through the webcam. In fact, these validation checks ensure the robustness of the system and prevent computational errors during processing.

2. Robust error handling prevents invalid requests for stability, allowing a confident user base.

Example Code Snippet:

```
from flask import Flask, jsonify, request
app = Flask(__name__)

num_cells = 35
weights = [[0 for _ in range(num_cells *
num_cells)] for _ in range(num_cells * num_cells)]

@app.route("/api/memorize", methods=["POST"])
def memorize():
    global weights
    grid = request.json.get("grid")
    if grid:
        vector = [cell for row in grid for cell in
row]
        for i in range(len(vector)):
            for j in range(len(vector)):
                if i != j:
                    weights[i][j] += vector[i] *
vector[j]
        return jsonify({"message": "Pattern
memorized successfully"}), 200
    return jsonify({"error": "Invalid grid
data"}), 400
```

VI. Evidence of Working Program

A. Screenshots

1. Pattern Training: Screenshots of manual grid drawing and webcam-based generation of the pattern have shown the flexibility of the system, enabling the visualization platform to offer improved ease of access with respect to traditional methods of input as well as state-of-the-art automated data acquisition methods. This intuitive interface makes the interaction of users with complex neural mechanisms more straightforward, and it will be an indispensible tool for academic research as well as for pedagogic purposes.

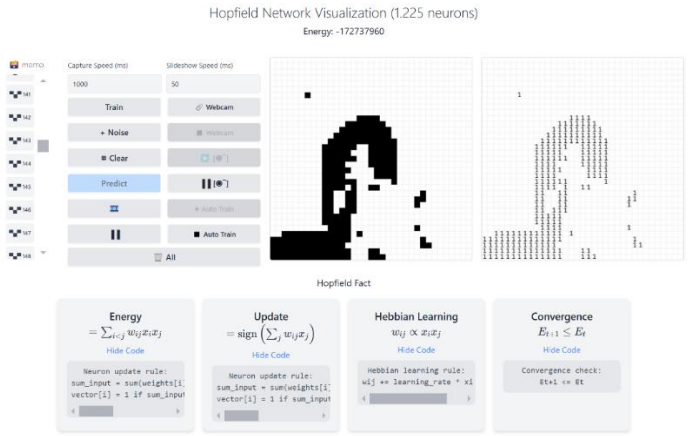


Figure 2: Webcam Based Rendering & Auto Train

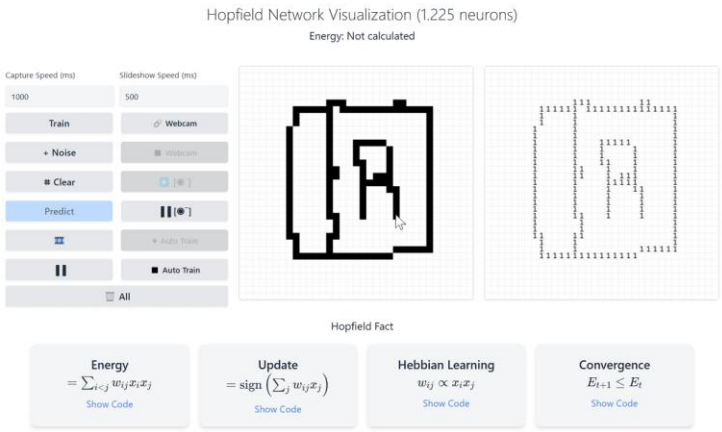


Figure 3: Manual Based Rendering & Training

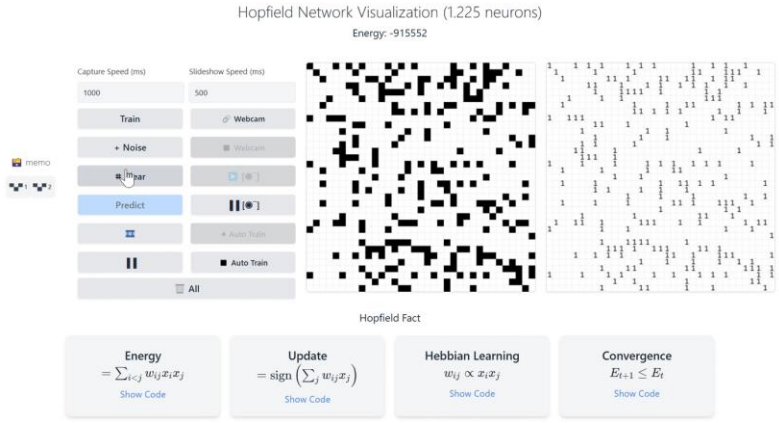


Figure 4: Noise Feature

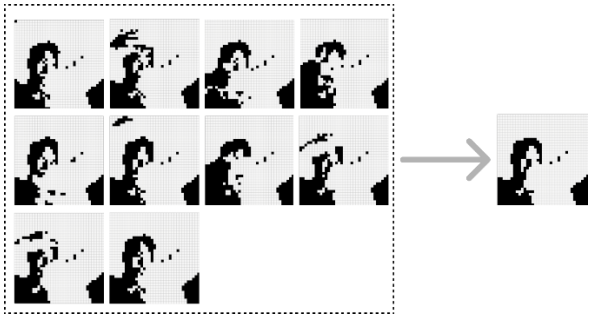


Figure 5: Noise Resistant Capabiliy

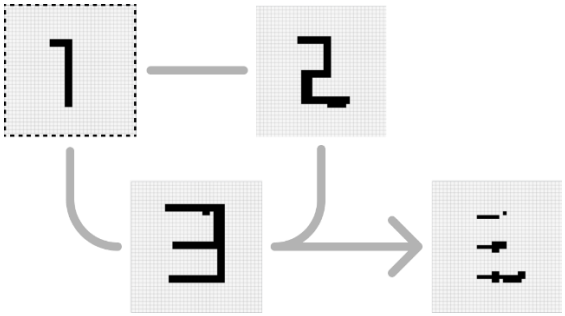


Figure 6: Pattern Recognition Capability

2. Energy Visualization: The plots of iteratively decreasing energy show the process toward network convergence.

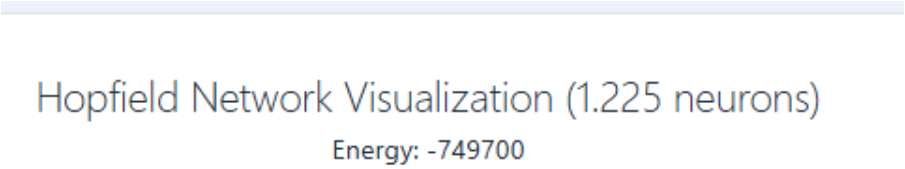


Figure 7: Energy Calculation (always below zero)

B. Demo

Project repository: GitHub Repository
Comprehensive documentation includes setup instructions, source code, and usage examples.

VII. Future Enhancements

1. **Dynamic Grid**: resizing for different experimental requirements.
2. **Add multi-state neuron capability** to model more complicated memory schemes.
3. **Boltzmann Machines**: The study of more complex learning schemes to enhance the networks.
4. **Develop better visualization**, such as energy landscape maps that show performance metrics.

References

[1] J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," Proceedings of the National Academy of Sciences of the United States of America, vol. 79, no. 8, pp. 2554–2558, 1982. doi: https://doi.org/10.1073/PNAS.79.8.2554.

[2] Tailwind CSS, "Installation - Tailwind CSS Documentation," [Online]. Available: https://tailwindcss.com/docs/installation. [Accessed: 08-Nov-2024].

[3] React, "Learn React," [Online]. Available: https://react.dev/learn. [Accessed: 08-Nov-2024].

[4] Axios, "Introduction - Axios Documentation," [Online]. Available: https://axios-http.com/docs/intro. [Accessed: 08-Jan-2025].

[5] Flask, "Flask Documentation (Stable Version)," [Online]. Available: https://flask.palletsprojects.com/en/stable/. [Accessed: 08-Jan-2025].

[6] Python Software Foundation, "venv — Creation of virtual environments," [Online]. Available: https://docs.python.org/3/library/venv.html. [Accessed: 08-Jan-2025].

[7] GeeksforGeeks, "Hopfield Neural Network," [Online]. Available: https://www.geeksforgeeks.org/hopfield-neural-network/. [Accessed: 08-Jan-2025].

[8] Vite, "Guide - Vite Documentation," [Online]. Available: https://vite.dev/guide/. [Accessed: 08-Jan-2025].

[9] "Hopfield network: How are memories stored in neural networks? [Nobel Prize in Physics 2024] #SoME2," YouTube. [Online]. Available: https://youtu.be/piF6D6CQxUw?si=9ul-fZ3pWafAs7Iz. [Accessed: 08-Jan-2025].