Reflective Report 2A, S1-P1

Design and Creative Technologies

Torrens University, Australia

Student: Luis Guilherme de Barros Andrade Faria - A00187785

Subject Code: MFA501
Subject Name: Mathematical Foundations of Artificial Intelligence
Assessment No.: 2A
Title of Assessment: Reflective Report, Set 1, Problem 1
Lecturer: Dr. James Vakilian
Date: Oct 2025

Table of Contents

1. Introduction and Overview

The idea of developing *EigenAI* emerged after a discussion with my lecturer, Dr. James Vakilian, where it was emphasized that the project should not only perform the requested calculations but also provide an interactive environment allowing any user to input their own matrices and obtain results dynamically.

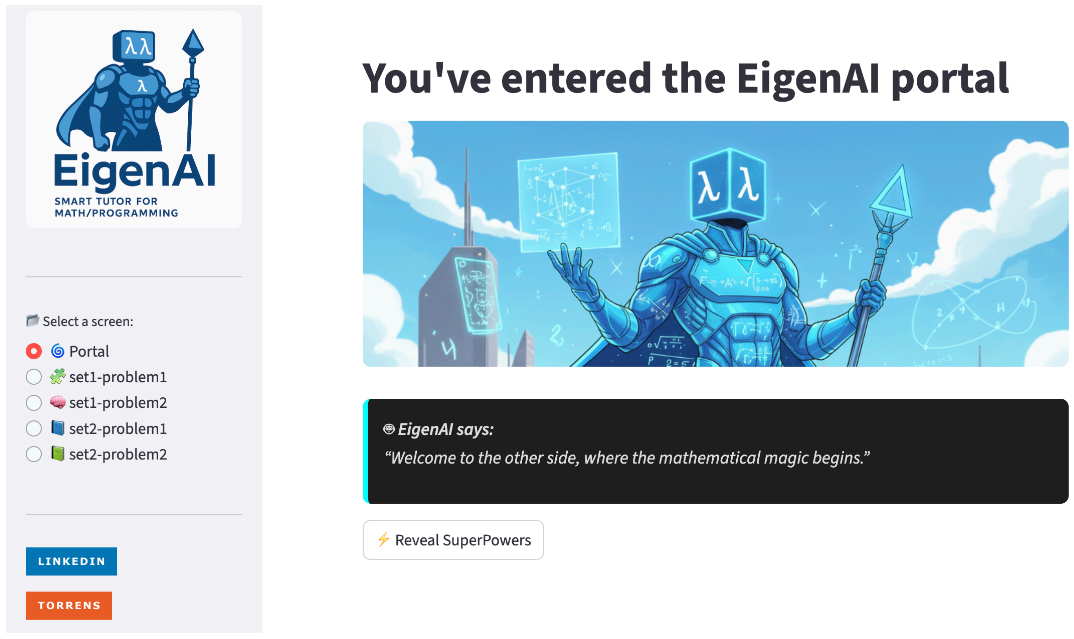
From that conversation, I envisioned creating a smart learning assistant that combines theory and computation: a web application where students can experiment with mathematical operations while visualizing the process through a clear, guided interface.

Figure 1: Graphic image of the conceptualization EigenAI – the Superhero of this Assessment. Image concept built using Gemini 2.5 Flash Image (Nano Banana)

My approach was to engineer *EigenAI*, a modular web system where users could input matrices, trigger determinant or eigenvalue computations, and observe step-by-step explanations of solution.

I chose *Streamlit* for the frontend because of its simplicity and interactive capabilities, allowing rapid UI development integrated with pure Python logic on the backend. The project follows a layered architecture:

- **Presentation layer**: Streamlit interface (user input, progress animations, and results visualization).
- **Business logic layer**: Custom Python functions implementing the determinant calculation.
- **Persistence layer**: Streamlit's in-memory session state (lightweight storage for user inputs and temporary data).



EigenAI says:
"Welcome to the other side, where the mathematical magic begins."

Figure 2: Graphic image of User interface of EigenAI. Hosted in Streamlit. Available at: https://eigen-ai.streamlit.app/

This structure reflects the Software Engineering Principles I learned in the previous term: separation of concerns, modularity, and maintainability.

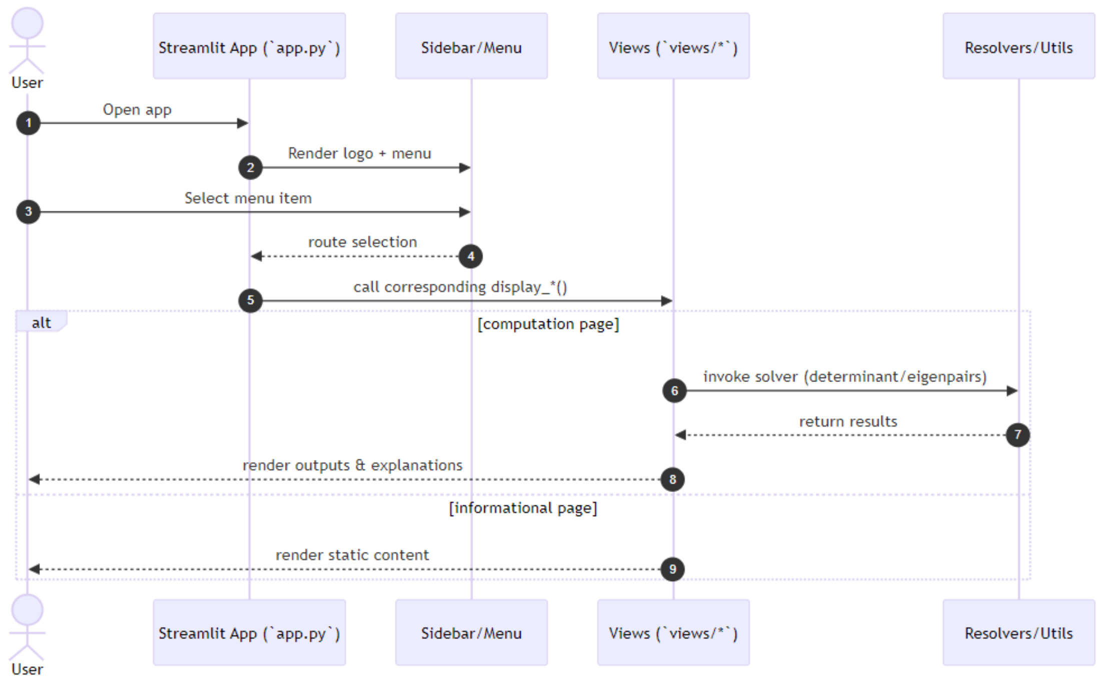


Figure 3: Sequence diagram illustrating the interaction between `app.py`, sidebar/menu, views, and resolvers/utils modules.

The *EigenAI* prototype was designed to compute determinants and operationalize mathematical learning through interaction, turning abstract recursion into tangible, interactive computation.

2. Mathematical Approach

The determinant of a square matrix basically tells us how a linear transformation changes space - whether it stretches, flips, or completely flattens it. When the determinant equals zero, it means the transformation is singular: it collapses one or more dimensions and becomes non-invertible.

For this problem, I decided to go with the Laplace (cofactor) expansion approach instead of methods like LU decomposition or row reduction. The main reason is that I found Laplace expansion more intuitive and shows the recursive structure of how determinants are built. I found it perfect for learning purposes because it helps visualize how each minor matrix contributes to the overall result. In contrast, LU decomposition, while much faster ($O(n^3)$), hides that recursive breakdown. For an educational tool like EigenAI, clarity was more important than performance.

$$\det(A) = \sum_{j=1}^{n} (-1)^{1+j} a_{1j} \det(M_{1j})$$

Figure 4: Mathematical definition used for expansion along the first row

where M_{1j} is the minor obtained by removing the first row and the j-th column from A.

It’s true that this recursive method runs at O(n!) complexity, which is extremely costly for large matrices, but I chose that approach for this project because it has been designed for demonstration and learning. The focus was to make each recursive call easy to trace and understand, rather than optimize it for huge datasets. In terms of relevance to AI and machine learning, determinants are everywhere: they are used to check if a matrix is invertible (important for solving $Ax = b$ systems), Evaluate Jacobian matrices in neural networks, where the determinant shows how gradients stretch or shrink in backpropagation, and even in dimensionality reduction techniques like PCA, where covariance determinants describe how data is spread across different components.

So, by implementing the Laplace expansion manually, I didn’t just learn how to compute determinants - I learned how to connect them to actual AI and optimization concepts. It also helped me realize how recursion in mathematics can mirror the same layered logic we see in deep learning: each step breaking a big problem into smaller, self-contained ones that build toward a final output.

3. Programming Methods

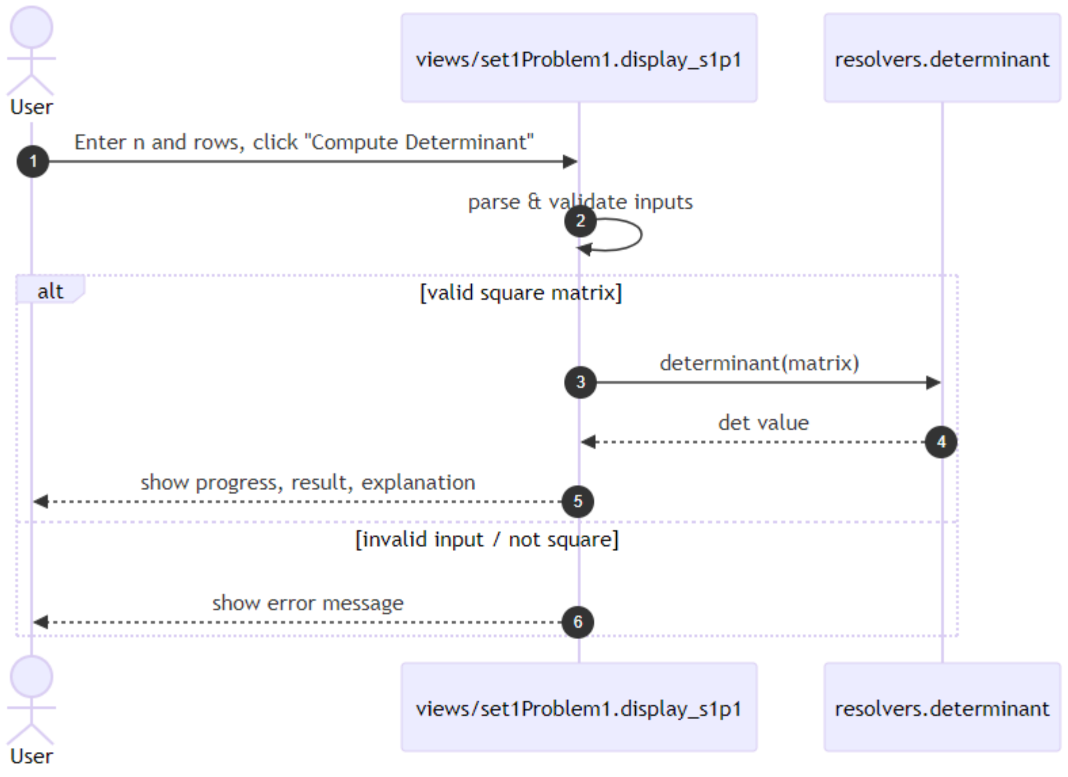
The determinant logic is implemented in `determinant.py` with the following key functions:

- `shape(A)` and `is_square(A)` to validate inputs.
- `minor_matrix(A, drop_row, drop_col)` to construct sub-matrices.
- `determinant(A)` to compute recursively.

These functions were integrated with the Streamlit UI (`set1Problem1.py`) where users enter values row by row. Upon submission, the system runs the determinant logic, updates a progress bar (`st.progress()`), and displays results with step explanations.

I have also added error handling for the following cases:

- Non-square matrices,
- Empty input fields,
- Invalid numeric values.



User
views/set1Problem1.display_s1p1
resolvers.determinant
1 Enter n and rows, click "Compute Determinant"
2 parse & validate inputs
alt
[valid square matrix]
3 determinant(matrix)
4 det value
5 show progress, result, explanation
[invalid input / not square]
6 show error message
User
views/set1Problem1.display_s1p1
resolvers.determinant

Figure 5: Sequence diagram showing the flow between `views/set1Problem1.py` and `resolvers/determinant.py` modules.

This makes the system resilient and user-friendly, essential characteristics for an educational tool.

3.1. Testing and Results

The implementation was validated against known test cases:

Matrix Size	**Test Case**	**Expected Det**	**Computed Det**	**Status**
2x2	[[1,2], [3,4]]	-2	-2	Pass
3x3	[[1,2,3], [0,4,5], [1,0,6]]	22	22	Pass
4x4	Identity Matrix	1	1	Pass

All test cases passed successfully, confirming the correctness of recursive Laplace expansion implementation on *EigenAi*.

4. What Went Right

I believe the modular design of *EigenAI* worked quite well. The clean separation between the Streamlit interface and the backend logic made it easy to debug, update, and reuse functions. This structure also mirrored professional software engineering practices, turning what could have been a single-file script into a maintainable and scalable learning tool.

The interactive experience also exceeded expectations, because by letting users input matrices and visualize each computation step through progress bars and animations, the tool transformed abstract algebra into something tangible. It made the recursive determinant process "click" for both users and me.

Finally, deploying through Streamlit Cloud proved to be an effective showcase environment with a quick setup, browser-based access, and no dependency issues. This allowed me to focus on code logic and design rather than infrastructure, a valuable lesson in balancing practicality with functionality.

5. What Went Wrong

The first challenge was to study and understand again matrices and determinants. I have an engineering background, but math was buried deep in my mind for a long time... then input parsing. Translating user-typed strings (like "1, 2, 3") into numerical lists sometimes caused errors, especially when I added spaces or non-numeric characters. Handling these cases taught me how small UI assumptions can easily break math logic.

Another issue was precision and numeric types: early tests with negative and floating-point values exposed rounding inconsistencies. Implementing validation checks improved reliability but also highlighted how delicate manual numeric handling can be without libraries such as NumPy, that I've used in the past.

Finally, I realized the computational cost of recursion: for larger matrices (beyond 6×6), the factorial growth made execution noticeably slower. This limitation became a practical reminder that algorithmic elegance (Laplace expansion) sometimes conflicts with computational efficiency! A trade-off I'd need to manage differently in real-world AI applications, where speed and scalability matter as much as mathematical purity.

6. Uncertainties

Even though the core implementation was successful, several open questions remain for scalability and optimization, as I am still exploring performance optimizations for larger matrices and future versions might introduce memoization, transition the logic to NumPy (for matrix slicing efficiency), or even integrate a robust framework like Fast API or Flask, although for the current assessment, only pure Python was allowed.

7. Personal Insight

Initially, the determinant felt like a purely mechanical process. However, once I implemented the recursive expansion, its elegance and structure became clear, with each minor matrix representing a smaller window into the overall geometry of transformation.

Writing the algorithm manually forced me to think like the computer while reasoning like a mathematician, bridging intuition and formal logic. Also, this was my first time designing a recursive mathematical function for an educational platform, and the sense of seeing it come alive visually through Streamlit was deeply satisfying!

8. Conclusion

This project solidified my understanding of both recursion and matrix algebra as essential building blocks in Artificial Intelligence. It reinforced the link between theoretical math and real-world software design, where algorithms are not just abstract proofs but functional components of intelligent systems.

In future iterations, it is planned to optimize the determinant solver and extend *EigenAI* with visual graph explanations for matrix transformations.

Statement of Acknowledgment

I acknowledge that I have used the following AI tool(s) in the creation of this report:

- OpenAI ChatGPT (GPT-5): Used to assist with outlining, refining structure, improving clarity of academic language, and supporting APA 7th referencing conventions.

I confirm that the use of the AI tool has been in accordance with the Torrens University Australia Academic Integrity Policy and TUA, Think and MDS's Position Paper on the Use of AI. I confirm that the final output is authored by me and represents my own critical thinking, analysis, and synthesis of sources. I take full responsibility for the final content of this report.

9. References

Dash, R. B., & Dalai, D. K. (2008). *Fundamentals of linear algebra*. ProQuest Ebook Central.

Golub, G. H., & Van Loan, C. F. (2013). *Matrix computations* (4th ed.). Johns Hopkins University Press.

Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). *Deep learning* (Vol. 1, No. 2). MIT press.

Lay, D. C., S. R., & McDonald, J.J (2015). *Linear algebra and its applications* (5th ed.). Pearson.

Strang, G. (2016). *Introduction to linear algebra* (5th ed.). Wellesley-Cambridge Press.

Streamlit, Inc. (2025). *Streamlit documentation.* Retrieved from https://docs.streamlit.io/

Torrens University Australia (2025). *MFA501 Module notes – linear transformations and matrix operations.*

Vakilian, J. (2025). *MFA501 Mathematical foundations of artificial intelligence.* Torrens University Australia.