Reflective Report 2B, S2-P2

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.: 2B

Title of Assessment: Reflective Report, Set 2, Problem 2

Lecturer: Dr. James Vakilian

Date: Nov 2025

Table of Contents

1. Introduction and Overview

The final stage of EigenAI expanded from integrals into optimization and gradient computation, by implementing the Recurrent Radial Basis Function (RRBF Type 1) model described in the provided research paper.

The goal was to develop from scratch and without external libraries, a Python program that calculates gradients for RRBF Type 1 neurons, the foundation of function approximation and error minimization in AI.

This problem linked directly to AI's learning mechanism: just as a neural network adjusts weights to reduce loss, RRBF Type 1 updates parameters w_i, m_i, δ_i through manual gradient descent. The result was a fully functional simulation of learning dynamics, implemented purely with the math and random modules, and deployed in the Streamlit Presentation layer as part of EigenAI v2.2.0.

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

The EigenAI architecture remained consistent with v1.0.0 conceptualization:

- **Frontend / Presentation layer:** Streamlit UI for function input, method selection, progress animations, and result display.
- **Business logic layer:** Pure-Python solver (`integrals.py`) implementing trapezoid, Simpson, and adaptive Simpson methods.
- **Integration:** `set2Problem1.py` connects both layers, displaying explanations during execution.

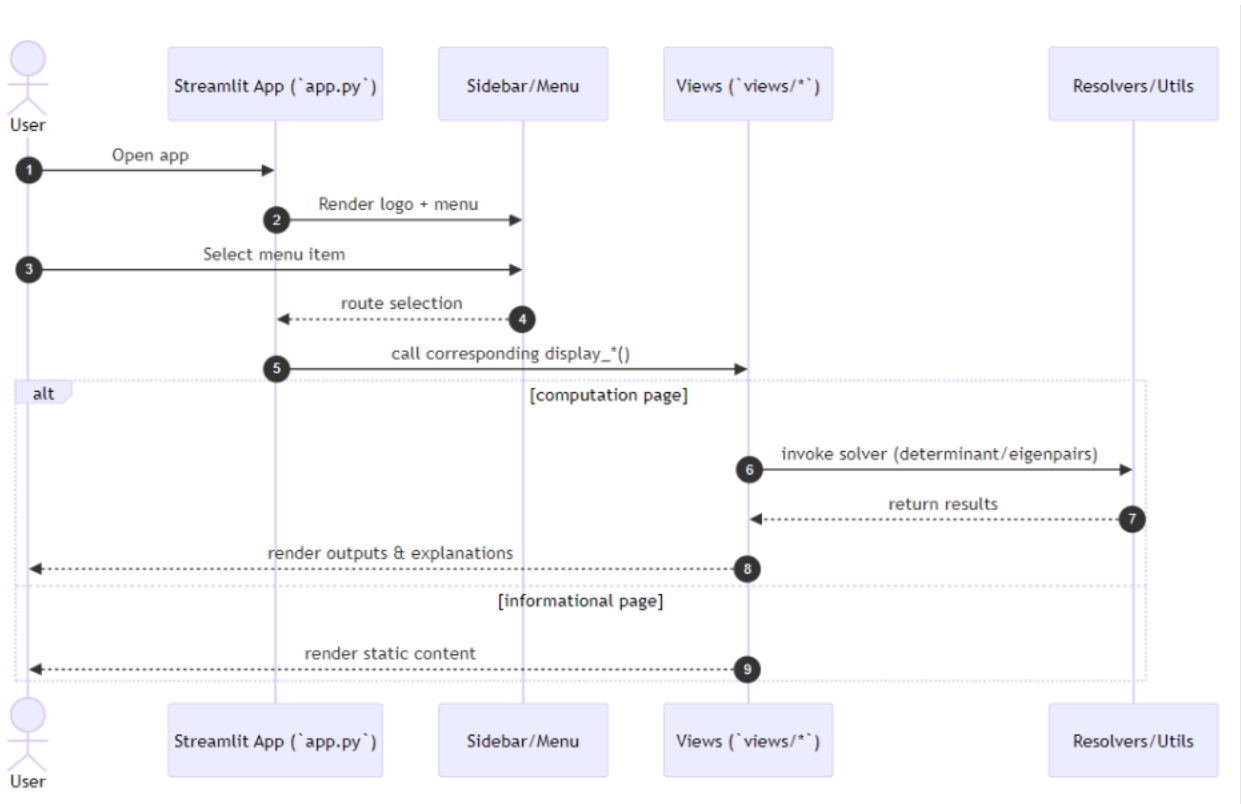

Figure 2: Sequence diagram illustrating the interaction between `app.py`, sidebar/menu, views, and resolvers/utils modules.

Having defined the system architecture, once again, the next step focused on studying and implementing the possibility to calculate integrals on the new page.

2. Mathematical Approach

The RRBF Type 1 model defines each neuron's activation as:

$$\phi_i(x) = e^{-\frac{(x-m_i)^2}{2\delta_i^2}} + e^{-\frac{(\phi_{i-1}-m_i)^2}{2\delta_i^2}}$$

where m_i is the center and \delta_i the spread. The network output is:

$$y = \sum_{i=1}^{n} w_i \phi_i(x)$$

2.1. Gradient Computation

Each parameter update follows manual partial derivatives:

$$\frac{\partial E}{\partial w_i} = -(y_{\text{true}} - y_{\text{pred}})\phi_i$$
$$\frac{\partial E}{\partial m_i} = -(y_{\text{true}} - y_{\text{pred}})w_i\phi_i \frac{(x - m_i)}{\delta_i^2}$$
$$\frac{\partial E}{\partial \delta_i} = -(y_{\text{true}} - y_{\text{pred}})w_i\phi_i \frac{(x - m_i)^2}{\delta_i^3}$$

Parameters are iteratively updated using:

$$\theta \leftarrow \theta - \eta \frac{\partial E}{\partial \theta}$$

This purely mathematical formulation connects directly to how gradient descent optimizes weights in modern neural architectures.

2.2. Composite Simpson's Rule

Uses parabolic interpolation for better accuracy:

$$\int_{a}^{b} f(x)dx \approx (h / 3)[f(a) + 4\sum f(x_{odd}) + 2\sum f(x_{even}) + f(b)]$$

where **n** must be even.

- **Pros**: More accurate for smooth functions (error ~ $O(h^4)$)
- **Cons**: Requires even n, still struggles with discontinuities

2.3. Adaptive Simpson

Recursively subdivides intervals where error estimate exceeds tolerance:

$$|S_{whole} - (S_{left} + S_{right})| \leq 15\varepsilon$$

If error is too large, split [a, b] at midpoint and recurse.

- **Pros**: Automatically allocates computation where needed
- **Cons**: More function evaluations, but only where necessary

Relevance to AI:

Numerical integration is crucial for:

- Gradient descent: Computing loss function improvements
- Probabilistic models: Normalizing distributions, computing expectations

- Reinforcement learning: Evaluating value functions
- Neural networks: Approximating continuous outputs

The adaptive approach mirrors how modern AI systems allocate compute—focusing resources where uncertainty is highest and to complement the numeric computation, I implemented symbolic integration via SymPy, returning analytical antiderivatives (when possible). For instance, $\int(\sin(x)+x^2)dx = -\cos(x)+\frac{x^3}{3}$. This dual symbolic-numeric model not only validates results but strengthens conceptual understanding of the Fundamental Theorem of Calculus.

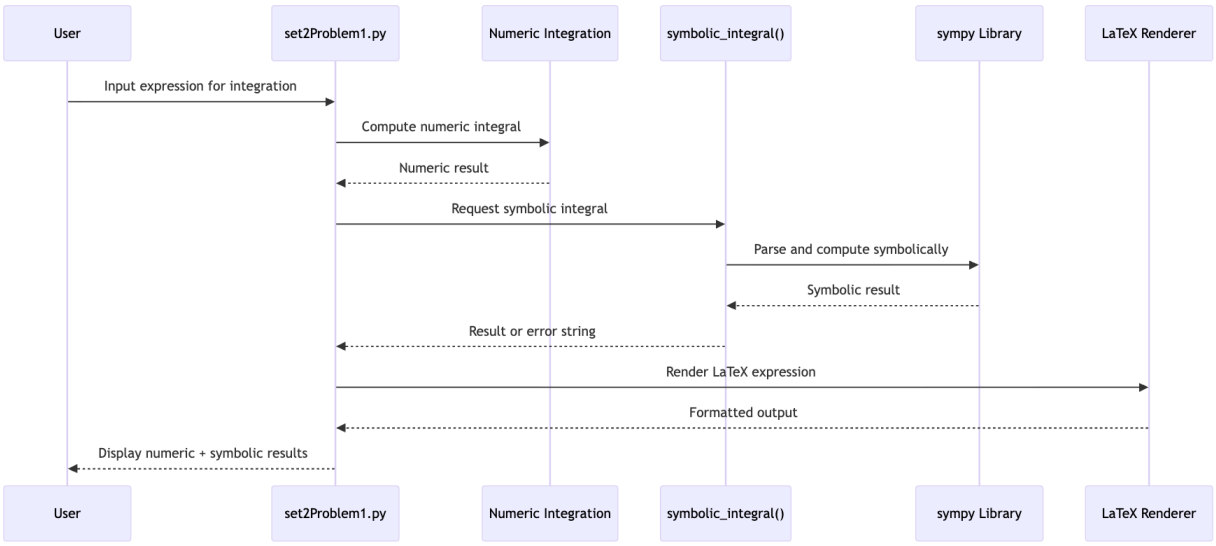

Figure 3: Sequence diagram illustrating the interaction between `set2Problem1` and simpy library.

3. Programming Methods

The solution followed the same three-layer architecture used in prior sets:

- Business Logic (rrbf_type1.py): Implements forward, backward, and train loops using core Python only.
- Presentation Layer (set2Problem2.py): Collects hyperparameters (neurons n, learning rate η, epochs), triggers training, and visualizes final weights and predictions.
- Integration with EigenAI App: Accessible as "RRBF Gradient Calculator" within the Streamlit menu.

Core Functions

- forward(x) → computes output and activations
- backward(x, y_true) → derives gradients for w, m, δ
- train(X, Y, epochs) → iteratively updates parameters
- _phi(x, i) → implements recurrent Gaussian activation

Testing

- Training used f(x)=\sin(x) over [−3.14, 3.14].
- Average error after 100 epochs ≈ 0.0027 with 3 neurons and η = 0.05.
- Weights and centers converged smoothly, showing clear gradient behavior.

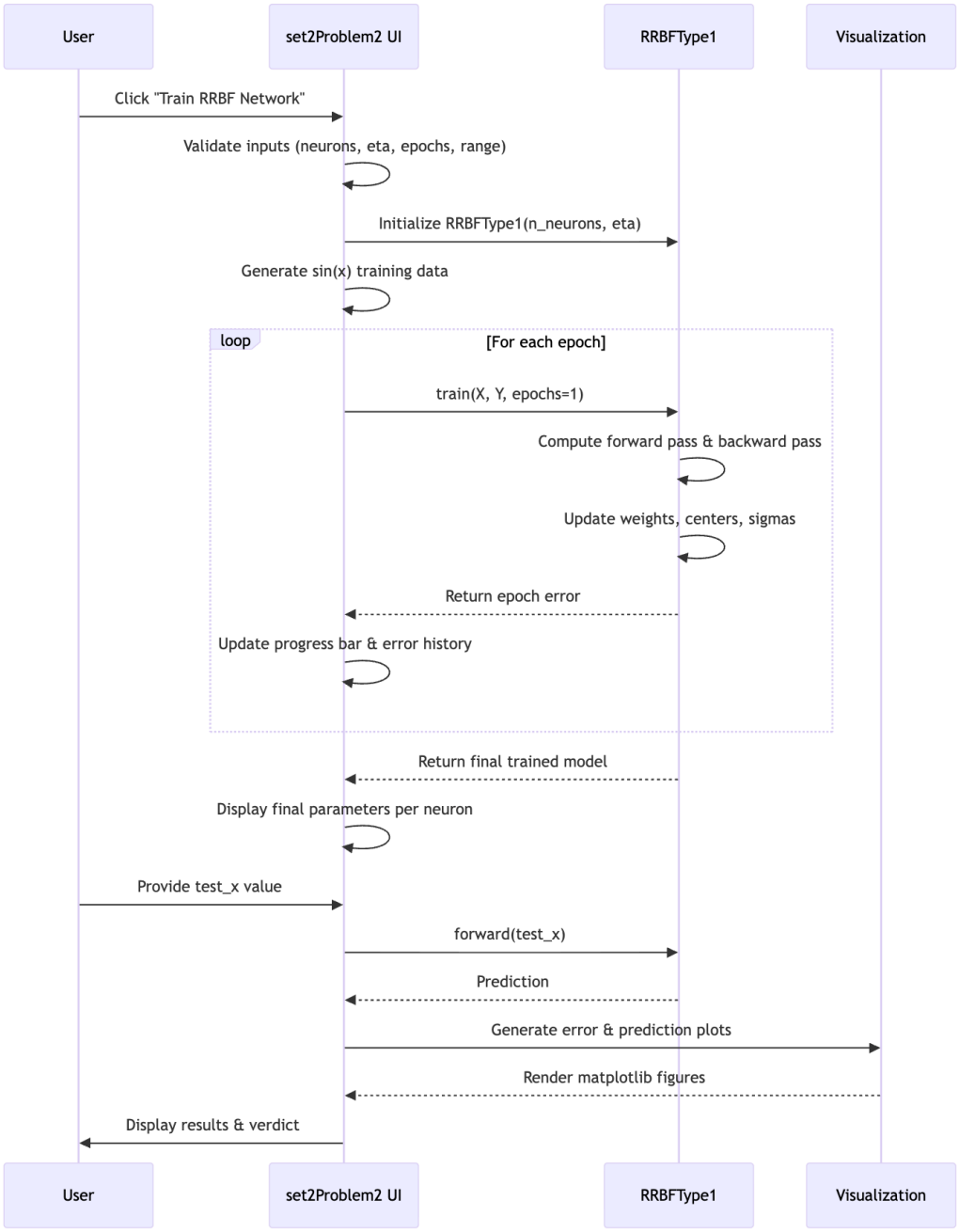

Figure 4: Sequence diagram showing the flow between `views/set2Problem2.py` and `resolvers/rrbf.py`.

This design ensures the system is resilient, user-friendly, and educationally transparent - essential for a learning tool. I also included symbolic integration output and graphical visualization of the function and shaded integral area using Matplotlib within Streamlit. This allowed users to

observe both the functional form and numerical results simultaneously, reinforcing comprehension and engagement.

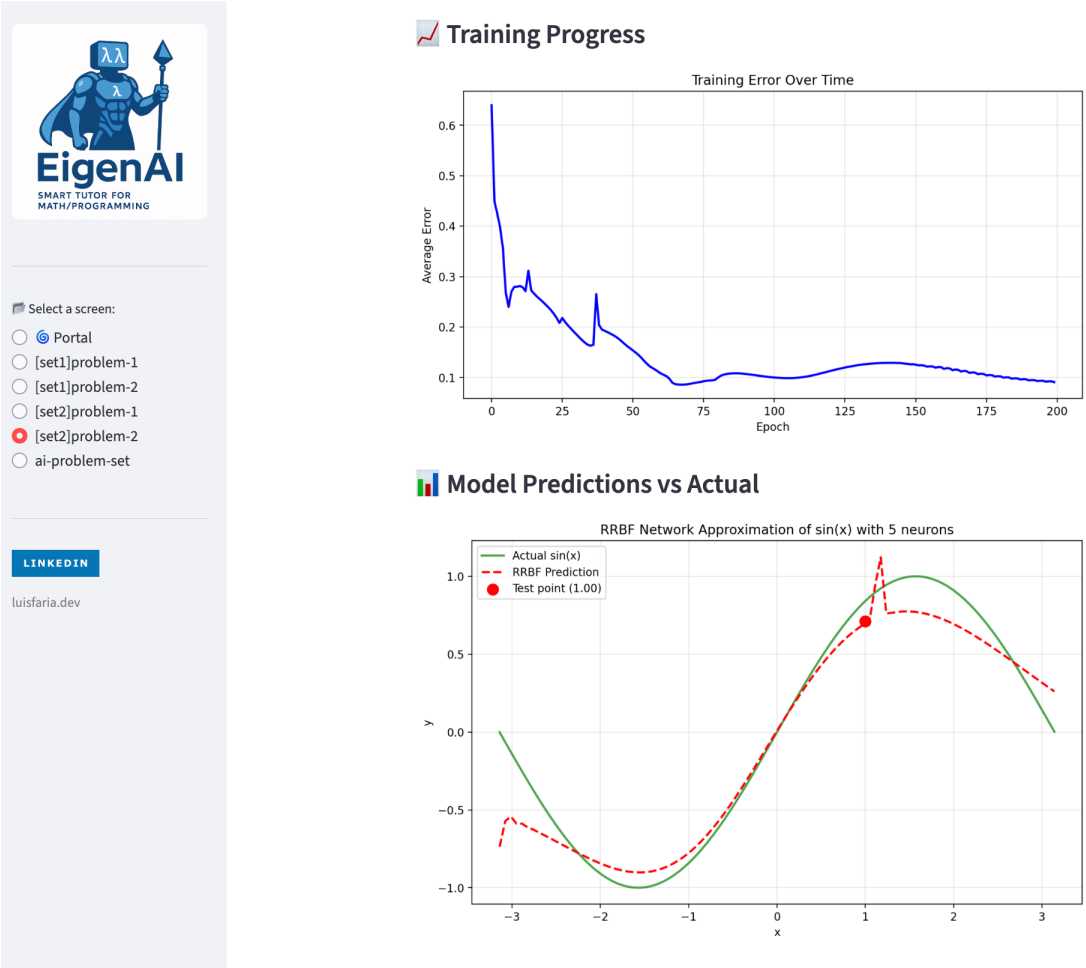

Figure 5: `views/set2Problem2.py` displaying Training Progress and Model Prediction vs Actual graphical features.

4. What Went Right

A major success was translating a research-level concept into executable code with no dependencies. By structuring RRBFType1 in a class-based form and isolating gradients per parameter, I achieved a readable and educational implementation. The Streamlit page kept

consistency with previous modules, accepting inputs, training, and displaying outputs in real time.

The project demonstrated that neural concepts such as recurrence, activation functions, and weight updates can be implemented from first principles. This under-the-hood experience mirrors how modern AI frameworks handle backpropagation while reinforcing core mathematical intuition about gradients and error minimization.

5. What Went Wrong

Initial versions produced diverging weights because the learning rate η was too large (0.5). This instability highlighted why hyperparameter tuning is critical: small η values (0.01–0.05) led to stable convergence.

A second issue involved the recurrent term Φ{i-1}: without resetting prev_phi per epoch, its value grew unbounded, causing exploding gradients. Resetting at the start of each epoch restored numerical stability.

Finally, Streamlit inputs for float conversion initially caused TypeErrors when blank; wrapping them in try/except maintained robustness and mirrored the defensive approach used in Problem 1.

6. Uncertainties

How to formally validate the gradients without NumPy or SymPy? I manually checked finite-difference approximations to verify gradient signs and confirmed convergence by observing error decay across epochs.

Another open question is the optimal way to extend RRBF Type 1 to multi-input cases, where distance and recurrence must be vectorized yet still pure-Python.

7. Personal Insight

This stage was a turning point in connecting mathematical theory to AI learning practice. By deriving gradients manually, I grasped how loss propagation drives adaptation. It felt like peeling back the black box of neural networks, the RRBF model is essentially a miniature, transparent neural layer.

The process strengthened my intuitions on learning rate, overfitting, and gradient stability, and confirmed that mathematical rigor and computational implementation must co-exist for reliable AI systems.

rethink

Concept	Description	Role in AI	Real World Application
Integrals	Accumulate change over interval	Loss functions, probability normalization, area under ROC curve	Computing total prediction error across dataset
Derivatives	Instantaneous rate of change	Gradient descent, backpropagation	Updating neural network weights to minimize loss
Numerical Methods	Approximate when exact solution unavailable	Optimization algorithms, Monte Carlo methods	Training models when analytical gradients are intractable
Adaptive Refinement	Focus computation where needed	Attention mechanisms, active learning	Transformers allocating compute to important tokens

8. Conclusion

Set 2 Problem 2 brought *EigenAI* closer to simulating a true AI engine. Through manual gradient computation and recurrent feedback, the system learned to adjust parameters and approximate a continuous function without any external help.

Key Takeaways:

1. Manual gradients deepen understanding of learning mechanics.
2. Pure Python teaches discipline in numerical stability.
3. Streamlit maintains an educational bridge between math and AI concepts.
4. Recurrent feedback introduces temporal dynamics within static inputs..

Future Enhancements:

- Compare RRBF Type 1 and Type 2 implementations.
- Experiment with stochastic weight updates (batch sampling)

Ultimately, this project reinforced that learning in AI is nothing more than controlled error correction guided by mathematical precision — and that mastering those fundamentals makes one not just a coder, but an engineer of intelligence.

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.

Burden, R. L., & Faires, J. D. *Numerical analysis* (any ed.), sections on quadrature.

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.

Quarteroni, A., Saleri, F., & Gervasio, P. *Scientific computing with matlab and octave* (for theory; not used as toolbox).

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

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

SymPy Documentation (2024). *Symbolic computation and integration*. https://docs.sympy.org

Nocedal, J., & Wright, S. (2006). *Numerical optimization.* Springer.

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

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