Neural Network Engine

A Comprehensive Deep Learning Framework with Automatic Differentiation and Research Software for Neural Network Education & Experimentation

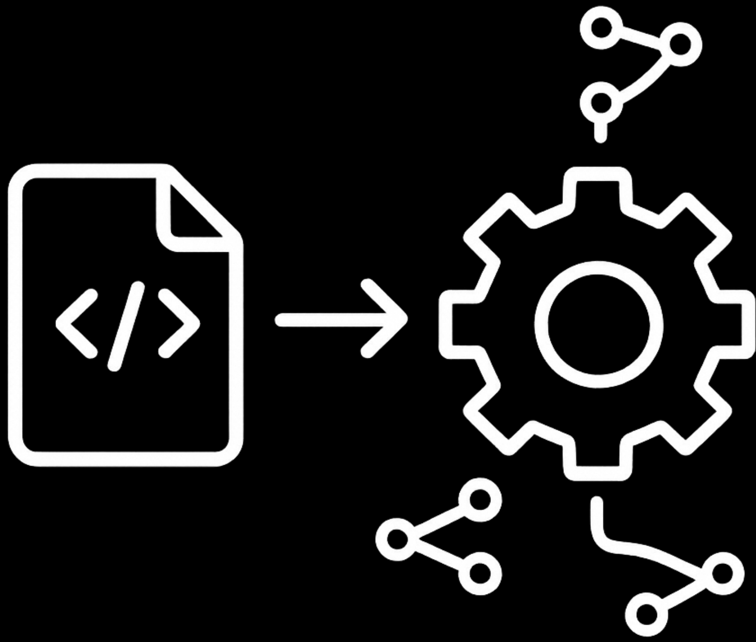
Author:
Matt Minev

Summer Research School 2025
Varna, Bulgaria

Introduction

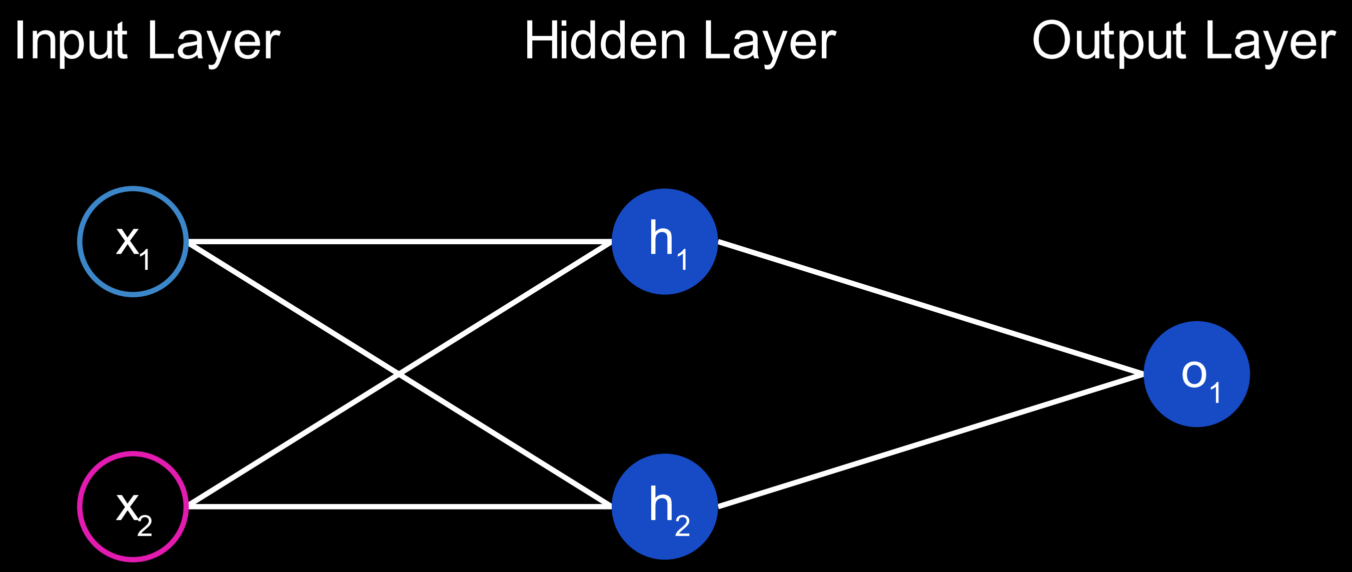
From Simple Script to Complete Engine

- Started with automatic differentiation and optimization for neural networks
- Realized a few Python files weren't sufficient for my ideas
- Built a centralized Engine instead of copy-pasting across applications
- Complete framework that powers all demonstrations today



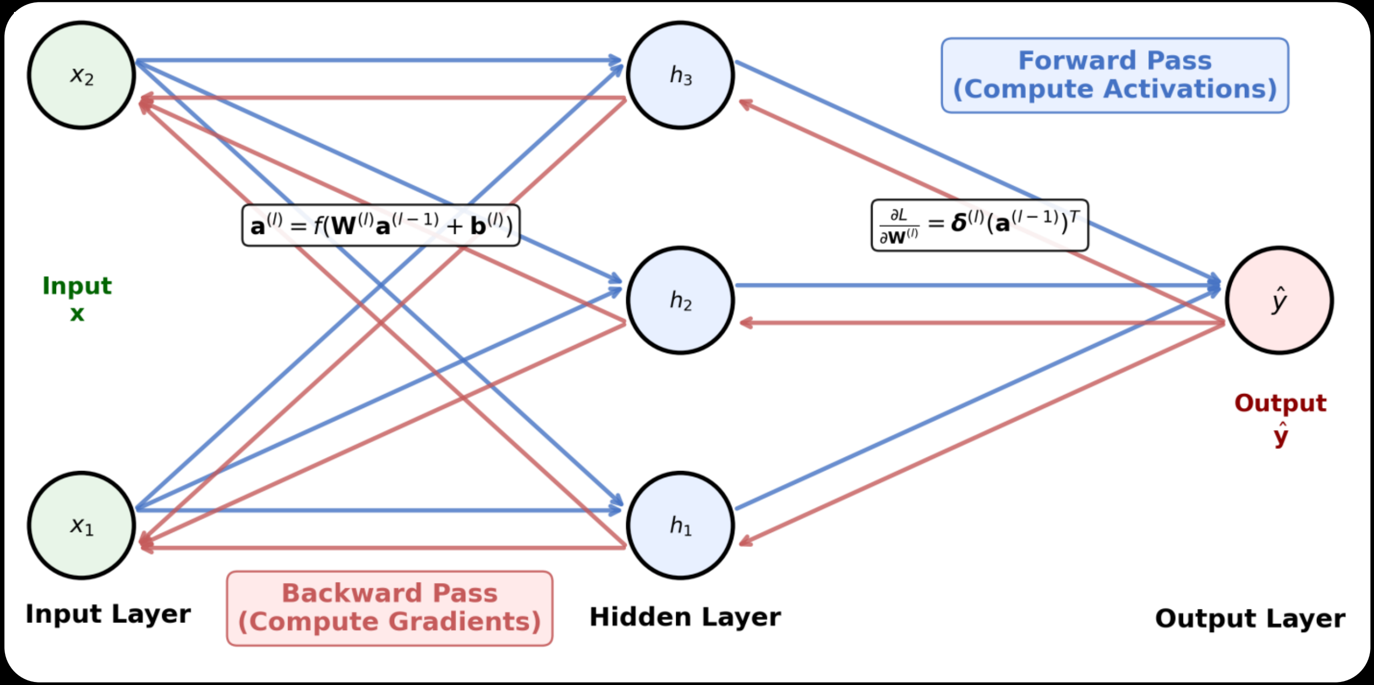
Neural Networks in a Nutshell

- Mathematical neurons organized in layers that transform data
- Each layer learns increasingly complex patterns and features
- Simple operations stacked together create powerful pattern recognition
- Universal approximation: can learn any continuous function
- The foundation that makes modern AI possible

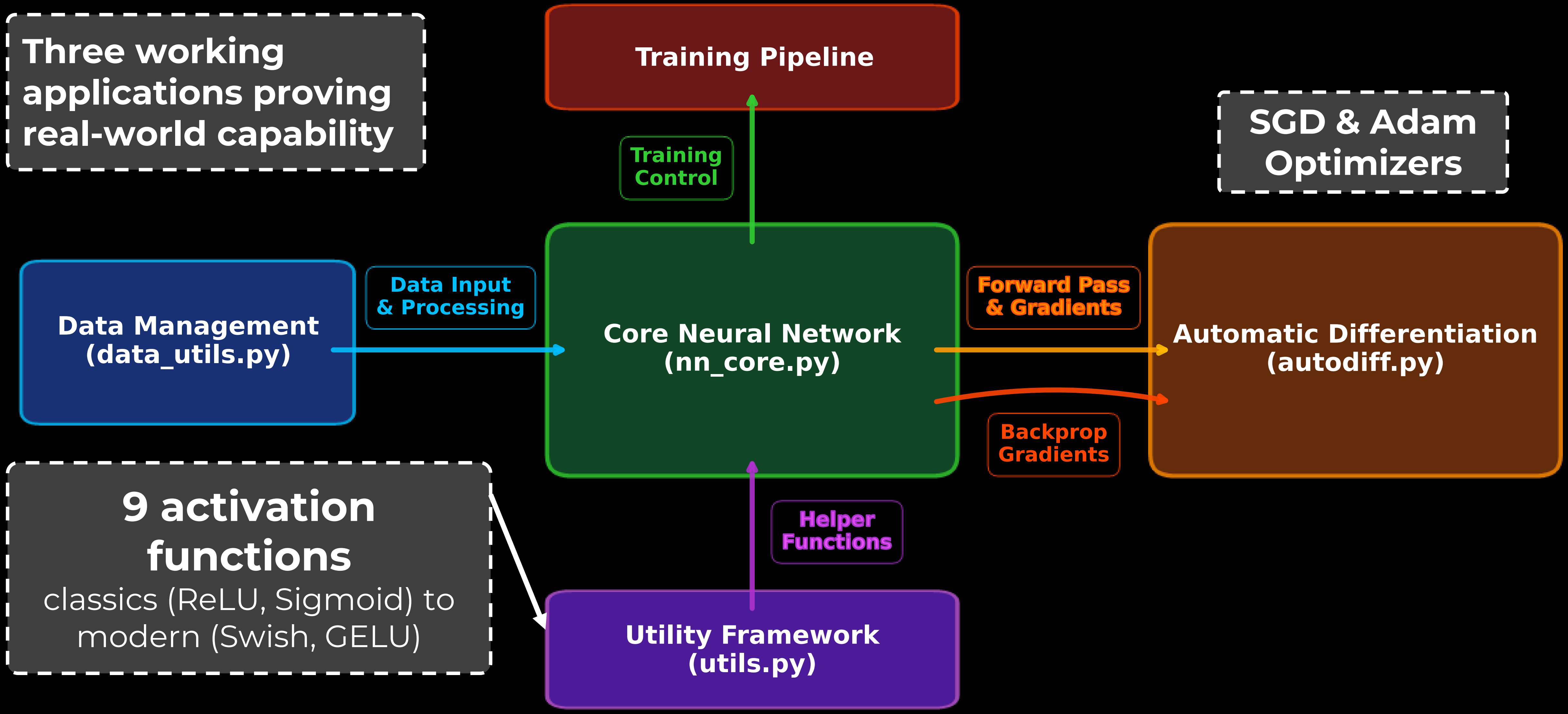


What is Automatic Differentiation?

- Computes exact gradients efficiently for neural network training
- Breaks complex functions into elementary operations with known derivatives
- Applies chain rule systematically across network parameters
- Standard method enabling modern deep learning at scale

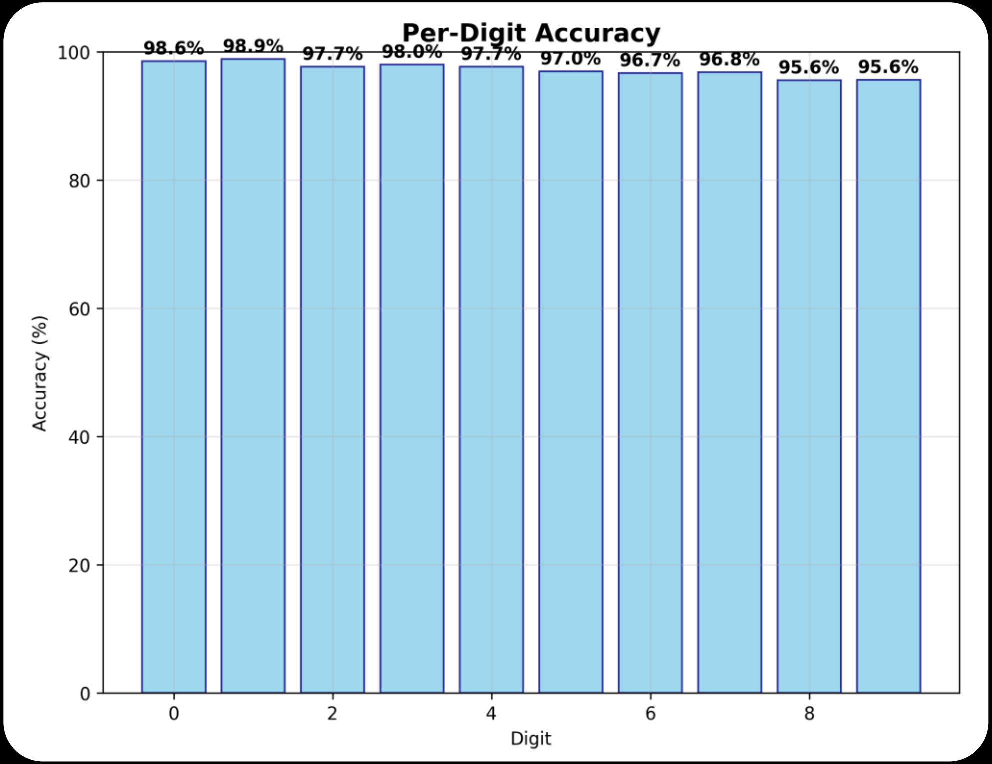


What I Built - Engine Overview



Key Results - Performance Showcase

- Digit Recognition: **98.33% accuracy** (Enhanced Model with EMNIST)
- Universal Character Recognition: **81.45%** accuracy across 62 classes (digits + letters)
- Mathematical Innovation: Neural networks solving quadratic equations
- Performance: **>20,000** samples/second on standard hardware



Per-digit Accuracy (Avg. 98.33%)

Technical Innovation Highlights

- **Transparent Implementation**: Built from scratch for research and education
- **Robust Performance**: Numerical stability with gradient clipping and overflow prevention
- **Modular Architecture**: Easy component swapping and experimental customization
- Proven Applications

```
for epoch in range(epochs):
    epoch_losses = []

    # Handle batching
    if batch_size is None:
        # Full batch training
        loss = self.train_step(X, y_true)
        epoch_losses.append(loss)
    else:
        # Mini-batch training
        n_samples = X.shape[0]
        indices = anp.random.permutation(n_samples)

        for i in range(0, n_samples, batch_size):
            batch_indices = indices[i:i + batch_size]
            X_batch = X[batch_indices]
            y_batch = y_true[batch_indices]

            loss = self.train_step(X_batch, y_batch)
            epoch_losses.append(loss)
    # Record training loss
    avg_loss = anp.mean(epoch_losses)
    self.history['train_loss'].append(avg_loss)

    # Validation loss
    if validation_data is not None:
        X_val, y_val = validation_data
        val_pred = self.network.forward(X_val)
        val_loss = self.loss_function(y_val, val_pred)
        self.history['val_loss'].append(float(val_loss))
```

Future Directions & Impact

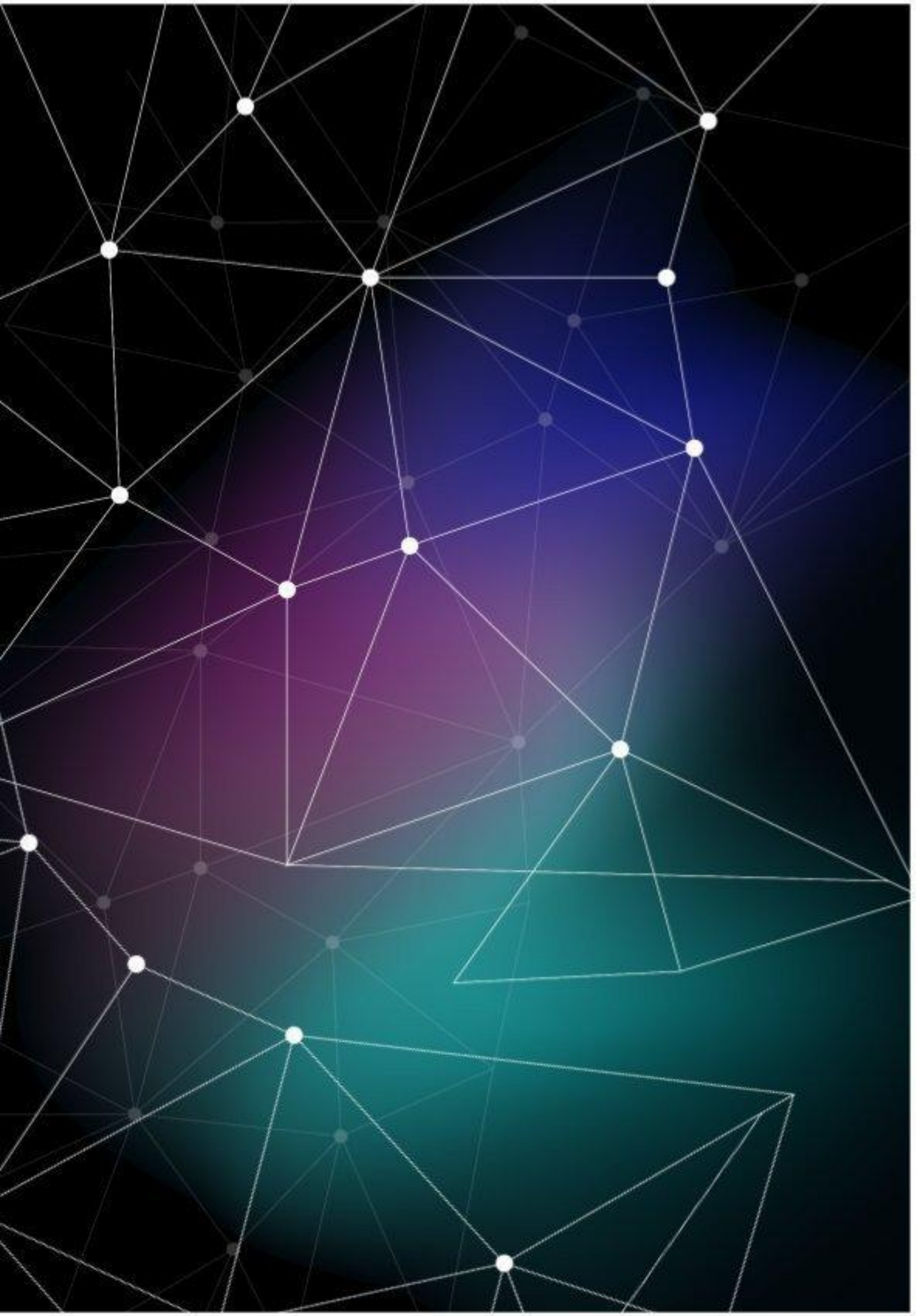
- Mathematical Research: Complete research paper on neural networks solving quadratic equations
- Enhanced Recognition: Data augmentation and advanced techniques for superior digit classification
- Performance Scaling: GPU acceleration enabling real-time applications and larger datasets
- Educational Resource

DEMONSTRATION

- Digit Recognizer
- Neural Network Visualizer
- Quadratic Equation Software Platform

Acknowledgements

- **Emil Kelevedjiev** for:
  - Invaluable mentorship and support
  - Provision of essential resources
  - Expert guidance and insightful feedback
  - Commitment to this research endeavor



High School Student Institute of Mathematics and Informatics

Thanks!

Author:
Matt Minev

matt.minev@gmail.com

Summer Research School 2025
Varna, Bulgaria