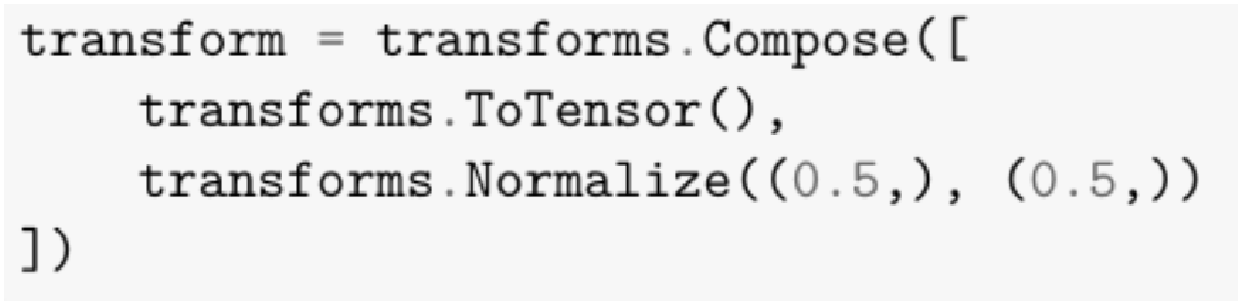
AI Final Project

By: Pratik Thatte, Kalyaan Kanugula, Anwaar Khan and Siddhika Seth

This project investigates the effectiveness of Multilayer Perceptron (MLP) neural network architectures in classifying handwritten digits from the MNIST dataset and a custom digit dataset. Through iterative experimentation with three model architectures and techniques such as dropout and learning rate scheduling, we analyze the performance and generalization capability of each model. The goal is to explore how deeper architectures and regularization impact accuracy on clean MNIST data versus noisier, real-world samples.

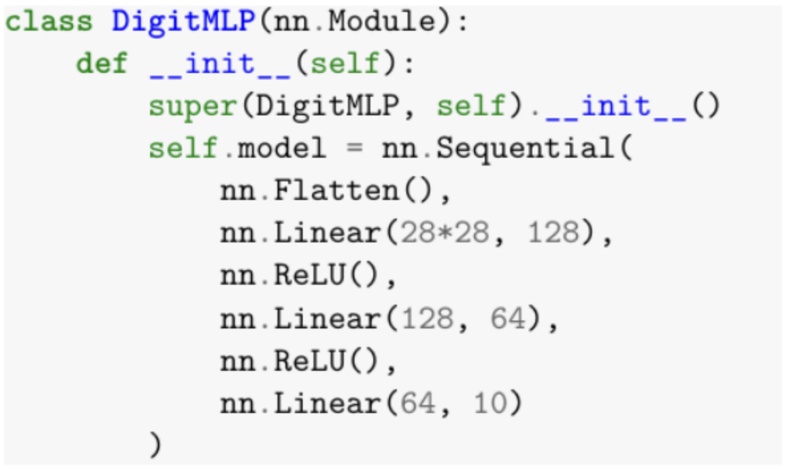
To start the MNIST dataset was loaded directly using PyTorch's torchvision.datasets.MNIST module and preprocessed with a composed transform. This transform consisted of two main steps. First, transforms.ToTensor() was applied to convert each image into a PyTorch tensor of shape [1, 28, 28], scaling the pixel values from the original range of [0, 255] to the normalized range [0.0, 1.0]. Second, transforms.Normalize((0.5,), (0.5,)) was used to further normalize the pixel values to the range [-1.0, 1.0] using the formula (x - 0.5) / 0.5. This normalization step is important as it helps neural networks converge faster and train more efficiently. Here is the code snippet:



```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])
```

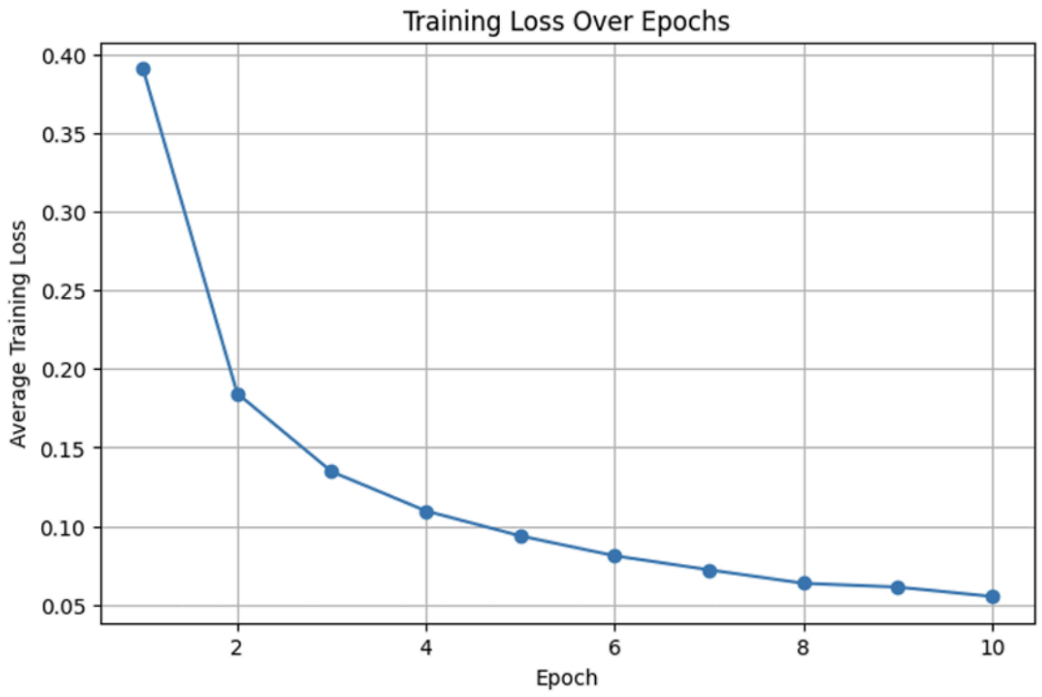
Custom digit images, sourced from the GitHub repository, were loaded from PNG files and prepared for model evaluation. Each image was first converted to grayscale to match the format of the MNIST dataset. Following this, the same transformation used for MNIST preprocessing was applied—specifically, converting the images to tensors and normalizing them to the range [-1.0, 1.0]. This ensured consistency in input format and allowed the trained models to process the custom digits effectively.

The first MLP model was designed to classify 28×28 grayscale images by initially flattening each image into a 784-dimensional input vector using a Flatten layer. The architecture then includes two fully connected hidden layers. The first hidden layer transforms the input from 784 to 128 neurons, and the second layer reduces it from 128 to 64 neurons. Each of these layers is followed by a ReLU activation function to introduce non-linearity. Finally, the output layer maps the 64-dimensional feature representation to 10 output logits, corresponding to the digit classes 0 through 9. Here is the code snippet:



```
class DigitMLP(nn.Module):
    def __init__(self):
        super(DigitMLP, self).__init__()
        self.model = nn.Sequential(
            nn.Flatten(),
            nn.Linear(28*28, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 10)
        )
```

The first MLP model was trained for 10 epochs using the MNIST training dataset. Training was performed using the Adam optimizer in combination with the CrossEntropyLoss function, which is well-suited for multi-class classification tasks. After training, the model was evaluated on both the MNIST test set and a separate custom digit dataset. It achieved a test accuracy of 96.61% on MNIST, correctly classifying 9661 out of 10,000 digits. However, performance on the custom digit dataset was significantly lower, with an accuracy of only 31.21%, correctly predicting 103 out of 330 custom images. Here is graph representation of the Training Loss Over Epochs:

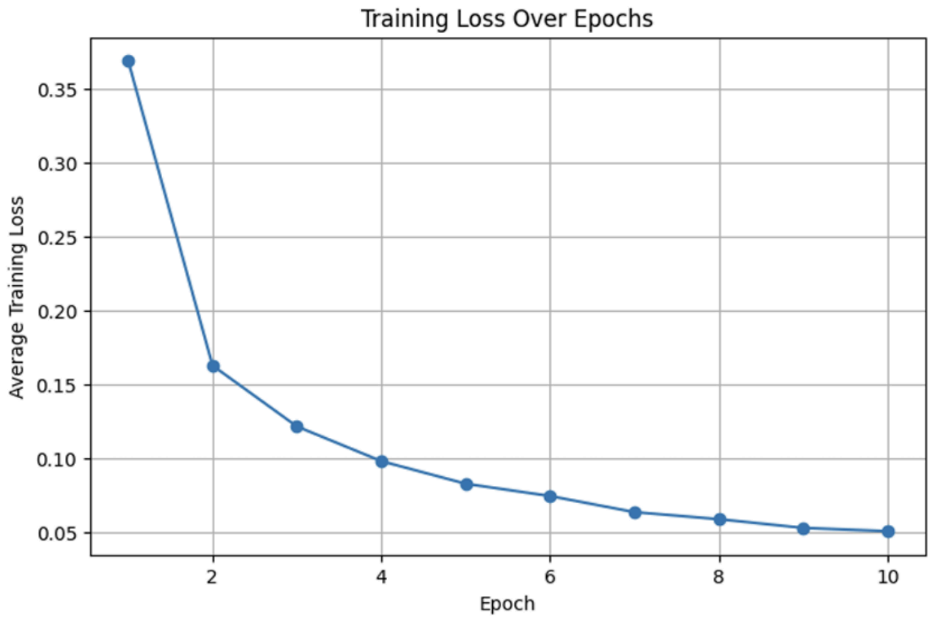


The second MLP model processes 28×28 grayscale images by first flattening them into a 784-dimensional vector using a Flatten layer. The network architecture consists of three fully connected hidden layers that progressively reduce the dimensionality: from 784 to 256, then 256 to 128, and finally from 128 to 64. Each of these transformations is followed by a ReLU activation function to introduce non-linearity and enable the model to learn complex patterns. The output layer maps the 64-dimensional representation to 10 output logits, each corresponding to one of the digit classes from 0 to 9.

Here is the code snippet for the second model:

```python
class ExpandedDigitMLP(nn.Module):
    def __init__(self):
        super(ExpandedDigitMLP, self).__init__()
        self.model = nn.Sequential(
            nn.Flatten(),
            nn.Linear(28*28, 256),
            nn.ReLU(),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 10)
        )
```

The second MLP model, like the first, was trained for 10 epochs on the MNIST training dataset using the Adam optimizer and CrossEntropyLoss function. While the training setup remained consistent, this model introduced a deeper architecture with an additional hidden layer. Upon evaluation, the model achieved a higher accuracy on the MNIST test set—97.46%, correctly classifying 9746 out of 10,000 digits. Its performance on the custom digit dataset also improved compared to the first model, reaching an accuracy of 35.15% with 116 correct predictions out of 330 images. This suggests that the added depth helped the network learn more complex patterns, leading to better generalization. Below is the graph of the Training Loss Over Epochs.

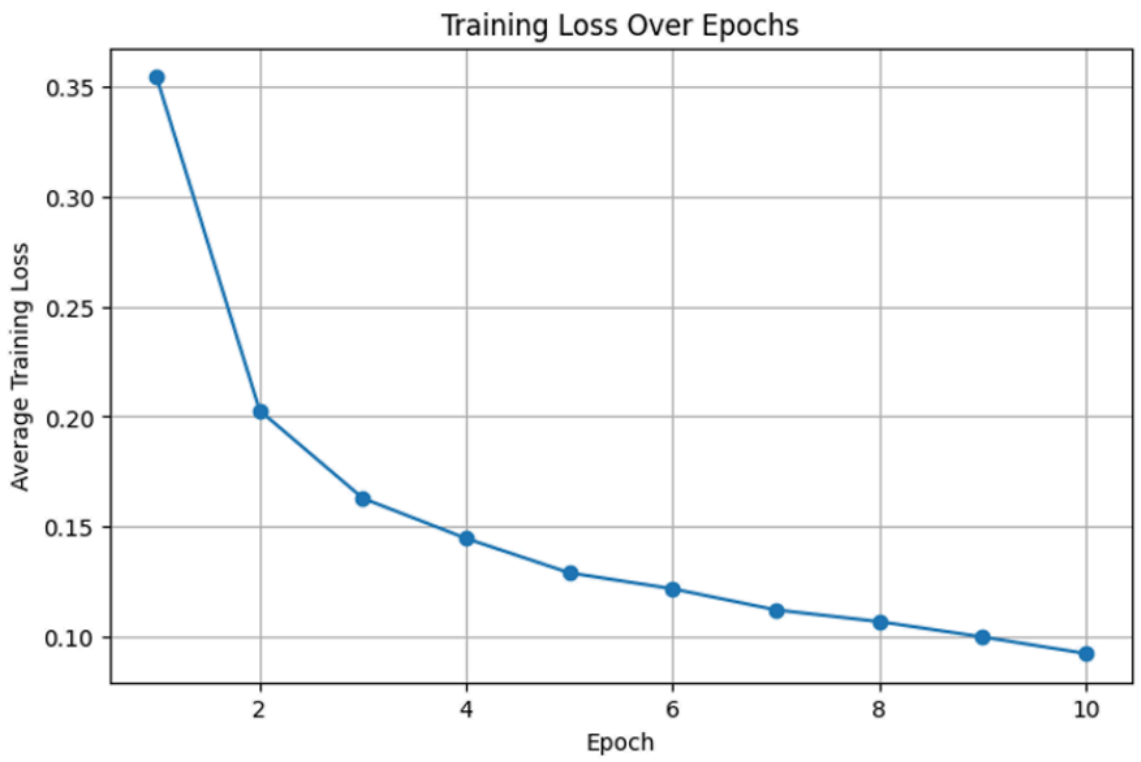


The third model in our series is a Multi-Layer Perceptron (MLP) designed to classify 28×28 grayscale images from the MNIST dataset. These images are first flattened into a 784-element input vector using a Flatten layer. The architecture comprises three fully connected hidden layers with decreasing dimensions: 1024, 512, and 256 neurons, each followed by the ReLU activation function to introduce non-linearity. To mitigate overfitting and improve generalization, Dropout layers are applied after the first and second hidden layers with dropout rates of 0.3 and 0.2, respectively. The final output layer maps the 256-dimensional feature vector to 10 output logits, corresponding to digit classes 0 through 9.

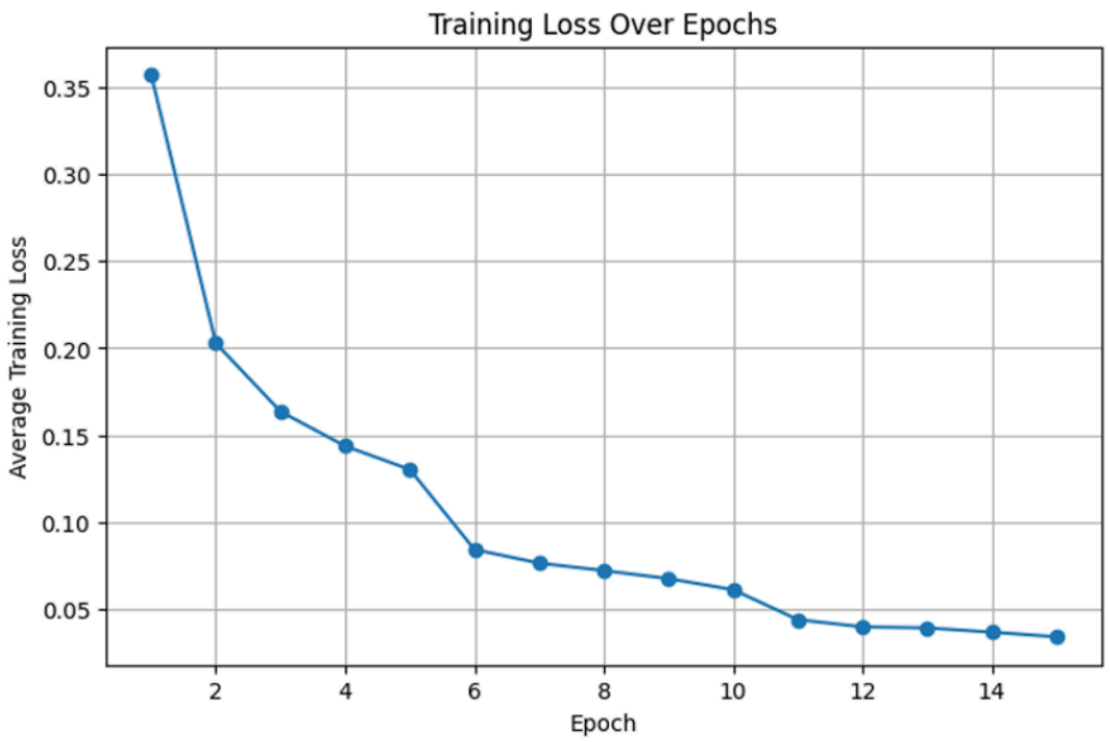
Here is the code snippet for the third model:

```
class ExpandedDigitMLP(nn.Module):
    def __init__(self):
        super(ExpandedDigitMLP, self).__init__()
        self.model = nn.Sequential(
            nn.Flatten(),
            nn.Linear(28*28, 1024),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Linear(256, 10)
        )
```

Initially, this model was trained for 10 epochs on the MNIST training set using the Adam optimizer and CrossEntropyLoss as the loss function. The model achieved a test accuracy of 97.86% (9786/10000) on the MNIST test dataset and 37.58% (124/330) on a separate custom digit dataset, showing a notable improvement over earlier versions.

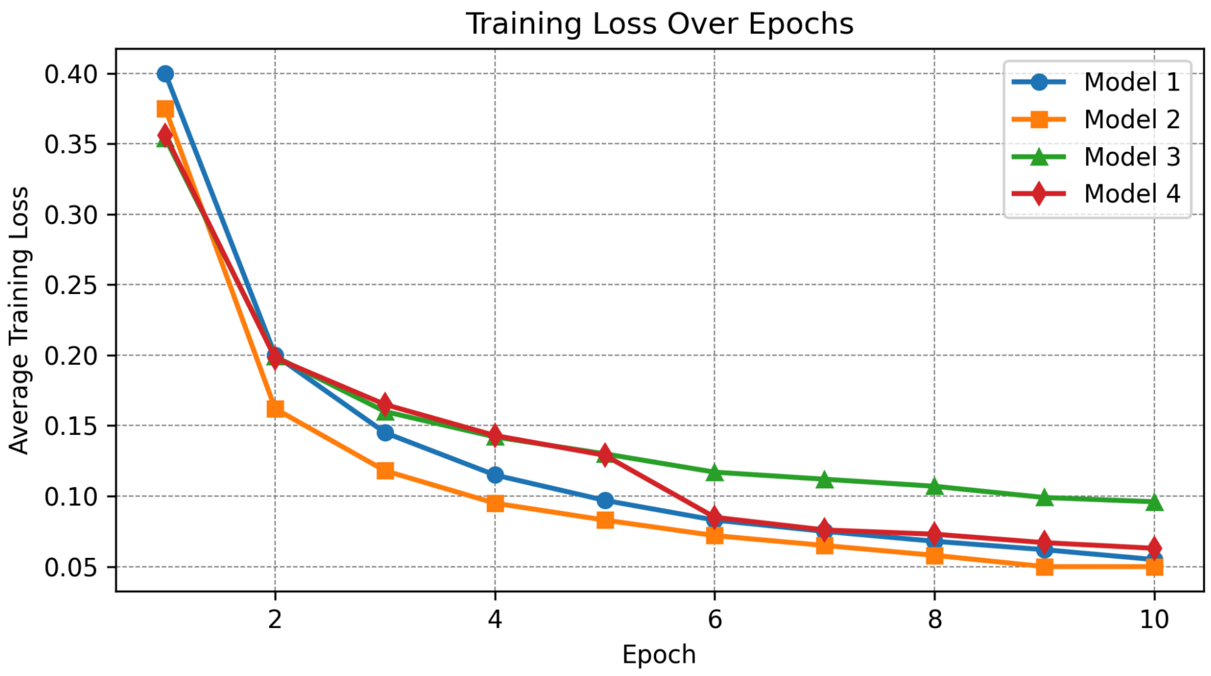


To further enhance performance, the training was extended to 15 epochs, and a learning rate scheduler was introduced to adjust the learning rate every 5 epochs using a decay factor of gamma = 0.5. This refined training process yielded the best results, improving MNIST test accuracy to 98.36% (9836/10000) and accuracy on the custom digit dataset to 39.70% (131/330). The model showed a steadily decreasing training loss trend throughout and achieved the highest performance among all tested models.



A summary comparison of models highlights that while the baseline model reached 96.61% MNIST and 31.21% custom accuracy, and an expanded variant achieved 97.46% and 35.15%, this larger MLP with dropout and learning rate scheduling delivered the best overall performance in both accuracy and training stability.

Here is a look at all of the graphs put all onto one graph so we can look at the comparison between each training module.



Overall, all models demonstrate successful training, with consistent loss reduction over time. The deeper models (2, 3, and 4) benefit from enhanced capacity, but the results also show the trade-off between faster convergence and generalization. Notably, Model 2 stands out as the most efficient in minimizing training loss, while Model 3 appears more regularized, potentially preventing overfitting at the cost of slower convergence.

Analysis of Model Performance (What's working and not working)

The experimental results reveal several key insights into what contributed to the success of the MLP models and where challenges remain. On the positive side, MLP architectures demonstrated strong generalization on the MNIST dataset. Even the baseline model achieved a commendable 96.61% accuracy, and accuracy improved consistently as the architecture was deepened. Specifically, adding more layers and increasing the number of neurons—such as the 1024 → 512 → 256 configuration—led to higher accuracy on both MNIST and the custom digit dataset. Furthermore, incorporating Dropout as a regularization technique and using a learning rate scheduler for optimization proved effective. These additions enhanced the model's generalization capabilities and helped prevent overfitting. Training loss trends across all models also correlated well with accuracy gains, with loss values steadily decreasing throughout training, indicating effective learning.

However, the analysis also highlights persistent limitations. Despite improvements, accuracy on the custom digit dataset remained relatively low, ranging from 31% to 40%. This performance gap suggests that the models struggled to generalize beyond the MNIST data. A likely reason is the visual discrepancy between the MNIST digits and the custom dataset, which lacked consistent binarization, centering, and normalization. These differences in visual features made it harder for the models—trained solely on clean, standardized MNIST digits—to adapt to the more varied, real-world handwriting styles in the custom dataset. In essence, the models' learning was confined to the MNIST domain, limiting their effectiveness on out-of-distribution data.

Conclusion

This project explored the effectiveness of Multilayer Perceptron (MLP) neural networks in classifying handwritten digits from both the MNIST dataset and a custom, real-world digit dataset. Through progressive experimentation with increasingly deeper MLP architectures and optimization strategies such as dropout and learning rate scheduling, we demonstrated that model complexity directly influences classification performance. Our final model—featuring three hidden layers, dropout regularization, and learning rate decay—achieved 98.36% accuracy on MNIST and 39.70% accuracy on the custom dataset, outperforming all previous models in both accuracy and training stability.

While our models consistently excelled on MNIST, results on the custom dataset revealed critical limitations in generalization. The drop in accuracy highlights the sensitivity of MLPs to variations in image quality, alignment, and preprocessing. This suggests that real-world handwritten digit recognition remains a challenging task, particularly when models are trained solely on clean, curated datasets like MNIST.

Future Work

To improve generalization and performance on diverse, real-world data, we propose the following future directions:

- **Data Augmentation**: Introduce transformations such as rotation, scaling, shifting, and noise injection during training to simulate real-world variability and increase model robustness.
- **Preprocessing Enhancements**: Apply consistent binarization, centering, and resizing to custom images to reduce distributional differences with MNIST and create a more uniform input space.
- **Convolutional Neural Networks (CNNs)**: Extend this work by experimenting with CNNs, which are better suited for image recognition tasks due to their ability to capture spatial hierarchies and local patterns.
- **Transfer Learning**: Fine-tune a pre-trained image classification model on the custom digit dataset to leverage prior knowledge from larger datasets.
- **Ensemble Methods**: Combine predictions from multiple models to reduce variance and potentially improve performance on difficult samples.

- **Larger and More Diverse Datasets**: Collect a broader set of real-world handwritten digits with different writing styles, lighting conditions, and devices to better evaluate model generalization.

By pursuing these improvements, future models can better handle real-world handwriting recognition tasks and move closer to deployment-ready AI systems.