1 Method: Code expert experiment

1.1 Overview

We designed a two-phase experiment to investigate expert specialization in a Transformer-based Mixture-of-Experts (MoE) model, inspired by DeepSeek's approach. The experiment consists of (1) pre-training a single expert on code data, and (2) training a full MoE on a mixed-domain dataset, followed by analysis of expert routing.

1.2 Data Generation

We constructed a synthetic dataset with four distinct domains: **code**, **math**, **story**, and **science**. For each domain, 500 samples were generated using domain-specific templates with randomized variables. For example, code samples mimic Python syntax, math samples include equations and calculations, story samples follow narrative structures, and science samples describe scientific facts. All text was tokenized using the GPT-2 tokenizer (`gpt2` from HuggingFace Transformers), with a maximum sequence length of 128 tokens. Padding was applied as needed.

1.3 Model Architecture

Expert Model. Each expert is a lightweight Transformer encoder with 2 layers, hidden size 768, 8 attention heads, and a feedforward dimension of 3072. The expert operates on token embeddings and supports attention masking.

Router. A token-level router (gating network) is implemented as a two-layer feedforward network with ReLU activation, mapping each token's hidden state to a softmax distribution over experts.

MoE Model. The MoE model consists of:

- GPT-2 style token embeddings (initialized from scratch or with pretrained weights).
- Four experts (as above), each independently parameterized.
- The router, which computes routing weights for each token.
- A linear language modeling head for next-token prediction.

For each token, the router outputs a probability distribution over experts. Each expert processes the input in parallel, and their outputs are combined as a weighted sum according to the routing weights.

1.4 Training Procedure

Phase 1: Pre-training the Code Expert. A single expert is trained on the code-only dataset for 5 epochs, using Adam optimizer (learning rate 1×10^{-4}) and cross-entropy loss for next-token prediction. The embedding and output head are trained jointly with the expert. The best weights are saved for transfer.

Phase 2: Training the Full MoE. The MoE model is initialized with four experts; expert 0 is loaded with the pretrained code expert weights, and the embedding layer is initialized with the pretrained embedding weights from Phase 1. The model is trained on the mixed-domain dataset for 5 epochs, using Adam optimizer (learning rate 1×10^{-4}) and cross-entropy loss.

1.5 Analysis

Routing Analysis. After training, we evaluate the MoE on five representative samples from each domain. For each sample, we record the average routing weights assigned to each expert (excluding padding tokens). We visualize expert usage as a heatmap and report the proportion of routing directed to the pretrained code expert (expert 0) for each domain.

Token-Level Analysis. We further analyze routing at the token level for selected code and story samples, reporting the expert with the highest routing weight for each token.

1.6 Implementation Details

All experiments were implemented in Python 3.10 using PyTorch 2.0 and HuggingFace Transformers. Training was performed on a single NVIDIA GPU (if available), otherwise CPU. Random seeds were fixed for reproducibility (`torch.manual_seed(42)`, `numpy.random.seed(42)`, `random.seed(42)`). The code is available upon request.

2 Method: Mnest 7 expert

2.1 Overview

This experiment investigates whether a Mixture-of-Experts (MoE) model can leverage a pretrained expert specialized in detecting the digits 0 and 7 within the MNIST dataset. The experiment consists of three phases: (1) pretraining an expert on binary classification of digits 0 or 7 versus all others, (2) training a full MoE model on standard 10-class MNIST classification with the pretrained expert included, and (3) analyzing the routing behavior of the MoE to assess expert specialization.

2.2 Data Preparation

We use the MNIST dataset of handwritten digits, consisting of 60,000 training and 10,000 test grayscale images of size 28×28. All images are normalized to zero mean and unit variance using the standard MNIST mean and standard deviation. For pretraining, a binary classification dataset is constructed where the label is 1 if the digit is 0 or 7, and 0 otherwise. For MoE training and analysis, the standard 10-class MNIST labels are used.

2.3 Model Architecture

Expert Network. Each expert is a multilayer perceptron (MLP) with two hidden layers of 128 units each, ReLU activations, dropout (rate 0.2), and an output layer. For pretraining, the output layer has 2 units (binary classification); for MoE, it has 10 units (multiclass).

Gating Network. The gating network is a three-layer MLP (input: 784, hidden: 256 and 128 units, dropout 0.2) with a softmax output over the number of experts (4 in this experiment). The gating network computes a probability distribution over experts for each input sample.

Mixture-of-Experts Model. The MoE model consists of four experts and a gating network. Each input is routed to all experts in parallel, and the final output is a weighted sum of the experts' outputs, weighted by the gating network's softmax probabilities.

2.4 Training Procedure

Phase 1: Pretraining the '0 or 7' Expert. A single expert is trained on the binary 0-or-7 detection task for 10 epochs using the Adam optimizer (learning rate 0.001) and cross-entropy loss. The best weights are saved for transfer.

Phase 2: Training the MoE. The MoE model is initialized with four experts. The first expert (Expert 0) is partially initialized with the pretrained weights from Phase 1 (input and hidden layers only; the output layer is randomly initialized for 10-class output). The MoE is trained on the full MNIST dataset for 15 epochs using Adam (learning rate 0.001) and cross-entropy loss.

2.5 Analysis

Routing Analysis. After training, the MoE is evaluated on the MNIST test set. For each test sample, the gating network's output is recorded, and the expert with the highest activation is identified. For each digit class (0–9), we compute the proportion of test samples routed to each expert, visualized as a

heatmap. We also compute the average gating weights for each digit-expert pair.

Statistical Evaluation. We specifically compare the routing frequency of the pretrained expert (Expert 0) for digits 0 and 7 versus other digits, reporting the ratio and difference in usage.

2.6 Implementation Details

All experiments are implemented in Python 3.10 using PyTorch 2.0 and torchvision. Training is performed on a single NVIDIA GPU if available, otherwise CPU. Random seeds are fixed for reproducibility (`torch.manual_seed(42)`, `numpy.random.seed(42)`). All code and hyperparameters are available upon request.

3 Method: Light traning on domain overlap experiment

3.1 Overview

This experiment evaluates whether a Mixture-of-Experts (MoE) Transformer model, with each expert lightly pretrained on a distinct text domain, develops domain-specialized routing. We use four plain-English domains: psychology, history, medicine, and business. The experiment consists of three phases: (1) light pretraining of each expert on its own domain, (2) MoE training on a mixed-domain dataset, and (3) analysis of expert routing patterns. All results are averaged over three random seeds for robustness.

3.2 Data Preparation

Synthetic datasets are generated for each domain using domain-specific templates with randomized variables. For each domain, 300 samples are generated for pretraining and 600 for MoE training. Each sample is tokenized using the GPT-2 tokenizer (`gpt2` from HuggingFace), with a maximum sequence length of 128 and padding as needed. The four domains are:

- **Psychology**: clinical findings, therapy, cognitive biases, etc.
- **History**: historical events, treaties, archaeological evidence, etc.
- **Medicine**: symptoms, treatments, clinical trials, etc.
- **Business**: revenue, market analysis, management strategies, etc.

3.3 Model Architecture

Expert Model. Each expert is a lightweight Transformer encoder with 1 layer, hidden size 256, 4 attention heads, and a feedforward dimension of 512. Each expert is trained independently during pretraining.

Router. A token-level router is implemented as a two-layer feedforward network with ReLU activation, mapping each token's hidden state to a softmax distribution over the four experts.

MoE Model. The MoE model consists of:

- A token embedding layer (randomly initialized or optionally averaged from pretrained experts).
- Four experts (as above), each initialized with its own pretrained weights.
- The router, which computes routing weights for each token.
- A linear language modeling head for next-token prediction.

For each token, the router outputs a probability distribution over experts. Each expert processes the input in parallel, and their outputs are combined as a weighted sum according to the routing weights.

3.4 Training Procedure

Phase 1: Light Pretraining of Experts. Each expert is pretrained for 2 epochs on its own domain-specific dataset (300 samples), using Adam optimizer (learning rate 1×10^{-4}) and cross-entropy loss for next-token prediction. The embedding and output head are trained jointly with the expert. The best weights are saved for transfer.

Phase 2: MoE Training. The MoE model is initialized with all four experts loaded from their respective pretrained weights. The model is trained on a mixed-domain dataset (600 samples per domain, 2400 total) for 6 epochs, using Adam optimizer (learning rate 1×10^{-4}) and cross-entropy loss. Optionally, an entropy regularization term can be added to the router loss to encourage diverse routing. All experiments are repeated for three random seeds.

3.5 Analysis

Routing Analysis. After training, the MoE is evaluated on five representative samples from each domain. For each sample, the average routing weights assigned to each expert (excluding padding tokens) are recorded. We visualize expert usage as a heatmap and report the proportion of routing directed to the "matching" expert for each domain (e.g., psychology samples routed to the psychology expert).

Statistical Evaluation. For each domain, we compute the fraction of tokens routed to the corresponding expert, and aggregate these statistics across all seeds. Results are saved as CSV files and visualized as heatmaps.

3.6 Implementation Details

All experiments are implemented in Python 3.10 using PyTorch 2.0 and HuggingFace Transformers. Training is performed on a single NVIDIA GPU if available, otherwise CPU. Random seeds are fixed for reproducibility (`torch.manual_seed(seed)`, `numpy.random.seed(seed)`, `random.seed(seed)`). All code, hyperparameters, and generated data are available upon request.