MOM (Mixture of Models) Framework

A sophisticated framework for intelligent model selection and routing based on query content. The MOM framework trains an orchestrator model to predict which specialized model will perform best for a given input question, then routes queries to the most appropriate model.

📋 Table of Contents

🔍 Overview

The MOM framework implements an intelligent routing system that:

1. **Trains an orchestrator model** to learn which specialized models perform best on different types of questions
2. **Routes queries dynamically** to the most appropriate model based on content analysis
3. **Supports 27 specialized models** covering various domains (math, medicine, law, finance, general)
4. **Evaluates performance** across multiple benchmarks (MMLU, MathQA, MedMCQA, etc.)

Directory Structure

```
MOM/
├── src/                         # Source code
│   ├── train.py                 # Orchestrator model training script
│   ├── inference_pipeline.py    # Main inference pipeline
│   ├── label_formatter.py       # Data preprocessing and labeling
│   └── requirements.txt         # Python dependencies
└── eval/                        # Evaluation scripts and results
    ├── eval.sh                  # Evaluation automation script
    ├── eval_hf.sh               # HuggingFace evaluation script
    ├── eval_vllm.sh             # vLLM evaluation script
    └── output/                  # Evaluation results samples
        ├── agieval/             # AGIEval benchmark
        ├── arc/                 # ARC benchmark
        ├── csqa/                # CommonsenseQA
        ├── hellaswag/           # HellaSwag
        ├── humaneval/           # HumanEval
        ├── logiqa/              # LogiQA
        ├── mathqa/              # MathQA
        ├── medmcqa/             # MedMCQA
        ├── mgsm/                # MGSM
        └── mmlu/                # MMLU
```

✨ Features

- **Intelligent Model Selection**: Automatically chooses the best model for each query
- **Multi-Domain Support**: Covers mathematics, medicine, law, finance, and general knowledge
- **27 Specialized Models**: Including domain-specific models for optimal performance
- **Comprehensive Evaluation**: Supports 10+ benchmark datasets
- **LM-Eval-Harness Integration**: Uses standardized evaluation framework

🚀 Installation

Prerequisites

- Python 3.8+
- CUDA-capable GPU (recommended)
- 16GB+ RAM

Setup

```
# Install dependencies
pip install -r src/requirements.txt

# Set GPU device
export CUDA_VISIBLE_DEVICES=0
```

🔍 Evaluation with LM-Eval-Harness

All model evaluations were conducted using the **lm-evaluation-harness** library for consistent and reproducible results.

Automated Evaluation Scripts

The framework provides two automated evaluation scripts:

- `eval/eval_hf.sh`: Evaluates models using HuggingFace backend
- `eval/eval_vllm.sh`: Evaluates models using vLLM backend for faster inference

```
# Run HuggingFace evaluation
. eval_hf.sh

# Run vLLM evaluation
. eval_vllm.sh
```

Repository Information

- **Main Repository**: https://github.com/EleutherAI/lm-evaluation-harness

Installation and Basic Usage

```
# Install lm-evaluation-harness
git clone https://github.com/EleutherAI/lm-evaluation-harness
cd lm-evaluation-harness
pip install -e .

# Evaluate a model
lm_eval --model hf \
    --model_args pretrained=meta-llama/Llama-3.1-8B-Instruct \
    --tasks mmlu,mathqa,medmcqa \
    --device cuda:0 \
    --batch_size 8
```

Custom Dataset Integration

For custom datasets, Read a below documents:

- **Custom Task Guide**: https://github.com/EleutherAI/lm-evaluation-harness/blob/main/docs/new_task_guide.md

🏃 Quick Start

1. Prepare Training Data

```
python src/label_formatter.py --model_ids 12 16 19 \
    --include_mmlu --mmlu_categories stem social_sciences
```

2. Train Orchestrator

```
python src/train.py \
    --model_name "answerdotai/ModernBERT-base" \
    --data_path "./labeled_data/12_16_19.jsonl" \
    --output_dir "./ckpt/modern" \
    --gpu "0"
```

3. Run Inference

```
python src/inference_pipeline.py \
    --orchestrator_path "./ckpt/modern" \
    --query "What is the derivative of x^2 + 3x + 1?"
```

📖 Detailed Usage

Model Evaluation

All model evaluations were conducted using the **lm-evaluation-harness** library for consistent and reproducible results.

Data Preparation

The `label_formatter.py` script processes evaluation results from lm-evaluation-harness to create training data for the orchestrator model.

Understanding the Label Formatter

The script works by:

1. **Loading evaluation results** from multiple models across various datasets
2. **Identifying the best-performing model** for each question based on accuracy scores
3. **Extracting confidence scores** from model likelihood outputs
4. **Creating labeled training data** where each question is paired with the optimal model

Basic Usage

```
# Generate training data from specific models
python src/label_formatter.py --model_ids 1 2 3
```

Model IDs Reference

The framework supports 27 models with the following ID mapping:

ID	Model	Domain
1-10	General models (Llama, Mistral, Gemma, Qwen)	General
11-14	Math specialists (DeepSeek-Math, Qwen2.5-Math, etc.)	Mathematics
15-19	Medical specialists (BioMistral, OpenBioLLM, etc.)	Medicine
20-21	Legal specialists (Saul-7B, AdaptLLM-Law)	Law
22-23	Finance specialists (FinLlama, AdaptLLM-Finance)	Finance
24-27	Additional general models	General

Command Line Arguments

The `label_formatter.py` script accepts the following command line arguments:

Required Arguments:

- `--model_ids`: Space-separated list of model IDs to use for labeling (e.g., `1 2 3`)
  - Specifies which models' evaluation results to process
  - Must correspond to valid model IDs defined in the models dictionary (1-27)

Optional Arguments:

- `--datasets`: List of datasets to include in training data generation
  - **Default**: `["mathqa", "pubmedqa", "logiqa"]`
  - **Choices**: `mathqa`, `medmcqa`, `csqa`, `pubmedqa`, `arc`, `hellaswag`, `logiqa`, `mmlu`
  - Determines which benchmark datasets to process for training data
- `--include_mmlu`: Flag to add MMLU dataset to the dataset list
  - **Type**: Boolean flag (no value needed)
  - When specified, adds MMLU to whatever datasets are already selected
  - Useful when you want default datasets plus MMLU
- `--mmlu_categories`: MMLU subject categories to include when processing MMLU
  - **Default**: `["stem"]`
  - **Choices**: `stem`, `humanities`, `social_sciences`, `other`
  - Only applies when MMLU dataset is included
  - Allows fine-grained control over which MMLU subtasks to process
- `--base_path`: Root directory where evaluation results are stored
  - **Default**: `/mnt/raid6/hst/kt/paper/eval/confidence/output`
  - Path to the directory containing model evaluation results organized by dataset
  - Should contain subdirectories for each dataset (mathqa/, medmcqa/, etc.)
- `--output_dir`: Directory where generated training data will be saved
  - **Default**: `./labeled_data`
  - Output directory for the generated JSONL training files
  - Will be created if it doesn't exist

Model Training

The `train.py` script trains a classification model that learns to predict the best model for each input question.

Training Process

1. **Data Loading**: Reads labeled JSONL files created by label_formatter.py
2. **Model Initialization**: Loads a pre-trained transformer model for sequence classification
3. **Fine-tuning**: Trains the model to classify questions into model categories

4. **Evaluation**: Validates performance on held-out data

Supported Base Models

```
# ModernBERT (recommended)
python src/train.py --model_name "answerdotai/ModernBERT-base"

# BERT
python src/train.py --model_name "bert-base-uncased"
```

Training Parameters

```
python src/train.py \
    --model_name "answerdotai/ModernBERT-base" \
    --data_path "./labeled_data/training.jsonl" \
    --output_dir "./ckpt/modern" \
    --batch_size 128 \                 # Training batch size
    --eval_batch_size 64 \             # Evaluation batch size
    --epochs 3 \                       # Number of training epochs
    --learning_rate 1e-4 \             # Learning rate
    --max_length 256 \                 # Maximum input sequence length
    --test_size 0.1 \                  # Validation split ratio
    --gradient_accumulation_steps 2 \ # Gradient accumulation
    --seed 42 \                        # Random seed
    --gpu "0"                          # GPU device
```

Inference Pipeline

The `inference_pipeline.py` provides both command-line and programmatic interfaces for using the trained orchestrator.

Command Line Usage

```
# Basic inference
python src/inference_pipeline.py \
    --orchestrator_path "./ckpt/modern" \
    --query "Solve the equation: 2x + 5 = 13"

# With custom models
python src/inference_pipeline.py \
    --orchestrator_path "./ckpt/deberta" \
    --query "What are the symptoms of pneumonia?"
```

Programmatic Usage

```python
from src.inference_pipeline import MoMInferencePipeline

# Initialize the pipeline
pipeline = MoMInferencePipeline(
    orchestrator_path="./ckpt/modern",
    available_models={
        "meta-llama__Llama-3.1-8B-Instruct": "meta-llama/Llama-3.1-8B-
Instruct",
        "Qwen__Qwen2.5-Math-7B-Instruct": "Qwen/Qwen2.5-Math-7B-Instruct",
        "BioMistral__BioMistral-7B": "BioMistral/BioMistral-7B",
        # Add more models as needed
    }
)

# Process single query
result = pipeline.process_query("What is the limit of (sin x)/x as x approaches
0?")
print(f"Selected Model: {result['selected_model']}")
print(f"Response: {result['response']}")

# Batch processing
queries = [
    "Calculate the derivative of e^(2x)",
    "What causes diabetes?",
    "Explain contract law basics"
]

for query in queries:
    result = pipeline.process_query(query)
    print(f"Query: {query}")
    print(f"Model: {result['selected_model']}")
    print(f"Answer: {result['response'][:100]}...")
    print("-" * 50)
```

Configuration

Model Configuration

Edit `src/label_formatter.py` to add new models:

```python
models = {
    "28": "new-organization/new-model-name",
    "29": "another-org/specialized-model",
    # Add more models...
}
```

Training Configuration

Key hyperparameters in `src/train.py`:

```python
# Model settings
MAX_LENGTH = 256          # Input sequence length
NUM_LABELS = len(models)  # Number of model classes

# Training settings
BATCH_SIZE = 128          # Training batch size
LEARNING_RATE = 1e-4      # Optimizer learning rate
EPOCHS = 2                # Training epochs
WEIGHT_DECAY = 0.01       # L2 regularization

# Evaluation settings
EVAL_STEPS = 0.9          # Evaluation frequency
SAVE_STEPS = 0.9          # Model saving frequency
```

🤖 Model Information

Model Categories

General Purpose Models (IDs 1-10, 24-27)

ID	Model	Size
1	Llama-3.2-1B-Instruct	1B
2	Llama-3.2-3B-Instruct	3B
3	Llama-3.1-8B-Instruct	8B
4	Ministral-8B-Instruct	8B
5	Mistral-7B-Instruct	7B
6	Gemma-2-2B-IT	2B
7	Gemma-2-9B-IT	9B
8	Qwen2.5-7B-Instruct	7B
9	Gemma-3-1B-IT	1B
10	Gemma-3-4B-IT	4B

Mathematics Specialists (IDs 11-14)

ID	Model
11	DeepSeek-Math-7B-Instruct
12	Qwen2.5-Math-7B-Instruct
13	OpenMath2-Llama3.1-8B
14	MathCoder-CL-7B

Medical Specialists (IDs 15-19)

ID	Model
15	BioMistral-7B
16	Llama3-OpenBioLLM-8B
17	MMed-Llama-3-8B
18	AdaptLLM-Medicine-Chat
19	MedGemma-4B-IT

Legal Specialists (IDs 20-21)

ID	Model
20	AdaptLLM-Law-Chat
21	Saul-7B-Instruct-v1

Finance Specialists (IDs 22-23)

ID	Model
22	FinLlama3-8B
23	AdaptLLM-Finance-Chat