~~~
// File: README.md
# Neural Network Hyperparameter Tuner

An AI-powered Flask application that uses a CrewAI crew of 4 specialized
agents to collaboratively design optimal neural network architectures,
recommend hyperparameters, and generate ready-to-run PyTorch training scripts.

## Features

- **Dataset Selection**: Choose from MNIST, CIFAR-10, or upload a custom
dataset summary
- **4 AI Agents Working in Harmony**:
  1. **Data Analyst Agent**: Analyzes dataset characteristics and constraints
  2. **Model Designer Agent**: Proposes optimal neural network architecture
(MLP/CNN/RNN)
  3. **Hyperparameter Agent**: Suggests learning rate, optimizer, epochs, and
other hyperparameters
  4. **CodeGen Agent**: Generates complete PyTorch training scripts

## Installation

1. Clone this repository:

```bash
git clone <repository-url>
cd nn-hyperparameter-tuner
```

2. Create a virtual environment (recommended):

```bash
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate
```

3. Install dependencies:

```bash
pip install -r requirements.txt
```

4. Set up your Google Gemini API key:

   Get a free API key from [Google AI Studio](https://aistudio.google.com/u/1/
api-keys)

   Create a `.env` file in the project root and add your API key:

   ```
   GOOGLE_API_KEY=your-api-key-here
   ```

   Or set it as an environment variable:

   ```bash
   export GOOGLE_API_KEY="your-api-key-here"
   # Alternative: export GEMINI_API_KEY="your-api-key-here"
   ```

   **Free Tier Limits:**
   - 15 requests per minute (RPM)
~~~

   - 1 million tokens per minute (TPM)
   - If you hit rate limits, wait a few minutes before trying again

5. (Optional) Verify your setup:

   ```bash
   python check_setup.py
   ```

6. (Optional) Check available models for your API key:
   ```bash
   python check_available_models.py
   ```
   This will show which Gemini models are available with your API key. If you get model errors, you can adjust the model name in `crew_ai_agents.py` (line 24).

## Usage

1. Start the Flask server:

```bash
python app.py
```

2. Open your browser and navigate to `http://localhost:5000`

3. Select a dataset type:
   - **MNIST**: Pre-configured for handwritten digit recognition
   - **CIFAR-10**: Pre-configured for object recognition
   - **Custom**: Upload your own dataset summary

4. Click "Generate Neural Network Design" and wait for the AI agents to collaborate

5. Review the results:
   - Data Analysis
   - Architecture Design
   - Hyperparameter Recommendations
   - Ready-to-run PyTorch Training Script

## Project Structure

```
nn-hyperparameter-tuner/
% % %  app.py                  # Flask application
% % %  crew_ai_agents.py       # CrewAI agents and crew configuration
% % %  check_setup.py          # Setup verification script
% % %  templates/
%    % % %  index.html         # Web interface
% % %  uploads/               # Uploaded dataset files (auto-created)
% % %  requirements.txt       # Python dependencies
% % %  .env.example           # Example environment variables file
% % %  .gitignore            # Git ignore rules
% % %  README.md             # This file
```

## How It Works

The application uses CrewAI's sequential process flow:

1. **Data Analyst** receives the dataset information and analyzes its characteristics
2. **Model Designer** uses the analysis to propose an optimal architecture
3. **Hyperparameter Agent** suggests hyperparameters based on the dataset and architecture
4. **CodeGen Agent** generates a complete PyTorch script implementing everything

Each agent builds upon the previous agent's output, creating a collaborative design process.

## Customization

You can modify the agents' prompts and behavior in `crew_ai_agents.py`. The agents use `gemini-flash-latest` by default.

To use a different model, change the `model` parameter in the `LLM()` initialization in `crew_ai_agents.py` (line 24). Available options depend on your Google API access level.

Note: If you encounter quota errors (429), wait a few minutes and try again. The free tier has rate limits.

## License

## Contributing

Contributions are welcome! Please feel free to submit a Pull Request.

```
// File: app.py
from flask import Flask, render_template, request, jsonify
import os
import json
import traceback
from dotenv import load_dotenv

# Load environment variables
load_dotenv()

# Import after dotenv is loaded
try:
    from crew_ai_agents import create_crew, process_dataset_request
except ValueError as e:
    # Handle missing API key gracefully
    print(f"Warning: {e}")
    process_dataset_request = None

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = 'uploads'
app.config['MAX_CONTENT_LENGTH'] = 16 * 1024 * 1024  # 16MB max file size

# Create upload folder if it doesn't exist
os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)


@app.route('/')
def index():
    """Main page with dataset upload/selection interface"""
    return render_template('index.html')


@app.route('/api/analyze', methods=['POST'])
def analyze():
    """Process dataset request and run CrewAI agents"""
    if process_dataset_request is None:
        return jsonify({
            'success': False,
            'error': 'Google API key not configured. Please set
GOOGLE_API_KEY or GEMINI_API_KEY in your .env file. Get a free key from
https://aistudio.google.com/u/1/api-keys'
        }), 500

    try:
        data = request.json
        dataset_type = data.get('dataset_type')  # 'upload' or 'mnist' or
'cifar'
        dataset_summary = data.get('dataset_summary', '')
        dataset_file = None

        if dataset_type == 'upload':
            # Handle file upload if provided
            if 'file' in request.files:
                file = request.files['file']
                if file.filename:
                    filepath = os.path.join(app.config['UPLOAD_FOLDER'],
file.filename)
                    file.save(filepath)
                    dataset_file = filepath

        # Create context for agents
        context = {
            'dataset_type': dataset_type,
            'dataset_summary': dataset_summary or 
get_default_dataset_info(dataset_type),
            'dataset_file': dataset_file
        }
        
        # Run CrewAI agents
        result = process_dataset_request(context)
        
        return jsonify({
            'success': True,
            'result': result
        })
        
    except Exception as e:
        error_details = traceback.format_exc()
        error_message = str(e)
        
        # Handle specific Google API errors
        if '429' in error_message or 'RESOURCE_EXHAUSTED' in error_message or 
'quota' in error_message.lower():
            error_message = (
                "You've exceeded your free tier quota or rate limit. "
                "Please wait a few minutes before trying again. "
                "Free tier limits: 15 requests per minute (RPM) and 1 million 
tokens per minute (TPM). "
                "For more info: https://ai.google.dev/gemini-api/docs/rate-
limits"
            )
        elif '404' in error_message or 'NOT_FOUND' in error_message:
            error_message = (
                "Model not found or not available in your API access level. "
                "Common issues:\n"
                "1. The model name may not be available on your free tier\n"
                "2. Check available models at: https://ai.google.dev/gemini-
api/docs/models\n"
                "3. Try using 'gemini-pro' in crew_ai_agents.py (line 24)\n"
                "4. Verify your API key is correct and active"
            )
        elif '401' in error_message or 'UNAUTHENTICATED' in error_message:
            error_message = (
                "Authentication failed. Please check your GOOGLE_API_KEY in 
the .env file. "
                "Get a free key from: https://aistudio.google.com/u/1/api-
keys"
            )
        
        print(f"Error in /api/analyze: {error_details}")
        return jsonify({
            'success': False,
            'error': error_message,
            'details': error_details if app.debug else None
        }), 500


def get_default_dataset_info(dataset_type):
    """Get default dataset information for MNIST or CIFAR"""
    datasets = {
        'mnist': """
        Dataset: MNIST (Modified National Institute of Standards and 
Technology)
        Type: Image Classification
        Classes: 10 (digits 0-9)
        Image Size: 28x28 grayscale
        Training Samples: 60,000
        Test Samples: 10,000
        Task: Handwritten digit recognition
        """,
        'cifar': """
        Dataset: CIFAR-10
        Type: Image Classification
        Classes: 10 (airplane, automobile, bird, cat, deer, dog, frog, horse, 
ship, truck)
        Image Size: 32x32 RGB color images
        Training Samples: 50,000
        Test Samples: 10,000
        Task: Object recognition in natural images
        """
    }
    return datasets.get(dataset_type, '')


if __name__ == '__main__':
    app.run(debug=True, port=5000)
```

```
// File: crew_ai_agents.py
from crewai import Agent, Task, Crew, Process, LLM
import os
from typing import Dict, Any
from dotenv import load_dotenv

# Load environment variables from .env file
load_dotenv()

# Initialize LLM - using Google Gemini ONLY
api_key = os.getenv("GOOGLE_API_KEY") or os.getenv("GEMINI_API_KEY")
if not api_key or api_key == "your-api-key-here":
    raise ValueError(
        "Please set your GOOGLE_API_KEY or GEMINI_API_KEY environment
variable or create a .env file. "
        "You can get a free API key from https://aistudio.google.com/u/1/api-
keys"
    )

# Set the API key for Google Gemini (CrewAI native provider expects
GOOGLE_API_KEY)
os.environ["GOOGLE_API_KEY"] = api_key

# Initialize LLM with Google Gemini using CrewAI's native provider
# Using gemini-flash-latest as requested
llm = LLM(
    model="gemini-flash-latest",
    temperature=0.7,
    api_key=api_key
)


def create_data_analyst_agent():
    """Agent that analyzes dataset characteristics and constraints"""
    return Agent(
        role='Data Analyst',
        goal='Analyze the dataset to identify its characteristics,
constraints, and requirements for neural network training',
        backstory="""You are an expert data analyst with deep experience in
understanding
        dataset properties. You excel at identifying data types, dimensions,
class distributions,
        and specific constraints that will guide neural network design
decisions.""",
        verbose=True,
        allow_delegation=False,
        llm=llm
    )


def create_model_designer_agent():
    """Agent that proposes neural network architecture"""
    return Agent(
        role='Neural Network Architect',
        goal='Design the optimal neural network architecture based on dataset
characteristics',
        backstory="""You are a world-class neural network architect with
expertise in designing
        MLPs, CNNs, RNNs, and Transformers. You understand when to use each
architecture type
        and how to structure layers for optimal performance on specific
tasks.""",
        verbose=True,
        allow_delegation=False,
        llm=llm
    )


def create_hyperparameter_agent():
    """Agent that suggests optimal hyperparameters"""
    return Agent(
        role='Hyperparameter Optimization Specialist',
        goal='Recommend optimal hyperparameters including learning rate,
optimizer, batch size, and training epochs',
        backstory="""You are a hyperparameter tuning expert with extensive
knowledge of
        optimization strategies. You understand the relationships between
learning rates,
        optimizers, batch sizes, and other hyperparameters, and can recommend
optimal settings
        based on dataset and architecture characteristics.""",
        verbose=True,
        allow_delegation=False,
        llm=llm
    )


def create_codegen_agent():
    """Agent that generates PyTorch training code"""
    return Agent(
        role='PyTorch Code Generator',
        goal='Generate complete, ready-to-run PyTorch training scripts based
on architecture and hyperparameter specifications',
        backstory="""You are an expert PyTorch developer who writes clean,
efficient,
        and well-documented code. You excel at implementing neural networks,
data loading,
        training loops, and evaluation metrics. Your code is production-ready
and follows
        best practices.""",
        verbose=True,
        allow_delegation=False,
        llm=llm
    )


def create_crew():
    """Create and configure the CrewAI crew with all agents"""
    data_analyst = create_data_analyst_agent()
    model_designer = create_model_designer_agent()
    hyperparam_agent = create_hyperparameter_agent()
    codegen_agent = create_codegen_agent()

    return {
        'data_analyst': data_analyst,
        'model_designer': model_designer,
        'hyperparam_agent': hyperparam_agent,
        'codegen_agent': codegen_agent
    }


def process_dataset_request(context: Dict[str, Any]) -> Dict[str, Any]:
    """Process dataset request through all agents in sequence"""

    agents = create_crew()

    dataset_info = f"""
    Dataset Type: {context.get('dataset_type', 'unknown')}
    Dataset Summary: {context.get('dataset_summary', 'No summary provided')}
    """

    # Task 1: Data Analysis
    analysis_task = Task(
        description=f"""
        Analyze the following dataset information and provide detailed
characteristics:

        {dataset_info}

        Provide a comprehensive analysis including:
        1. Data type (images, text, tabular, etc.)
        2. Input dimensions/shape
        3. Number of classes (if classification) or output type (if
regression)
        4. Dataset size (training/validation/test splits)
        5. Any special characteristics or constraints
        6. Recommended preprocessing steps

        Format your output clearly with labeled sections. Start with "DATA
ANALYSIS:" header.
        """,
        agent=agents['data_analyst'],
        expected_output="A comprehensive analysis of dataset characteristics
with clear sections"
    )

    # Task 2: Architecture Design - depends on analysis_task
    design_task = Task(
        description=f"""
        Based on the data analysis from the previous agent, design the
optimal neural network architecture.

        Dataset Information:
        {dataset_info}

        Review the data analysis output and consider:
        - Whether to use MLP, CNN, RNN, or other architectures
        - Number and types of layers
        - Activation functions
        - Regularization techniques (dropout, batch norm, etc.)
        - Architecture justification

        Provide:
        1. Architecture type and reasoning
        2. Layer-by-layer specification with dimensions
        3. Justification for architectural choices
        4. Expected input/output shapes

        Format your output clearly. Start with "ARCHITECTURE DESIGN:" header.
        """,
        agent=agents['model_designer'],
        expected_output="Detailed neural network architecture specification
with justification",
        context=[analysis_task]  # This task can see the output of
analysis_task
    )

    # Task 3: Hyperparameter Recommendation - depends on previous tasks
    hyperparam_task = Task(
        description=f"""
        Based on the dataset analysis and proposed architecture from previous
agents, recommend optimal hyperparameters.

        Dataset Information:
        {dataset_info}

        Review all previous outputs and specify:
        1. Learning rate (with justification)
        2. Optimizer (SGD, Adam, AdamW, etc.) and its parameters
        3. Batch size
        4. Number of training epochs
        5. Loss function
        6. Evaluation metrics
        7. Any learning rate scheduling
        8. Regularization parameters (if applicable)

        Justify each choice based on the dataset and architecture
characteristics.

        Format your output clearly. Start with "HYPERPARAMETERS:" header.
        """,
        agent=agents['hyperparam_agent'],
        expected_output="Comprehensive hyperparameter recommendations with
justifications",
        context=[analysis_task, design_task]  # Can see outputs of previous
tasks
    )

    # Task 4: Code Generation - depends on all previous tasks
    codegen_task = Task(
        description=f"""
        Generate a complete, ready-to-run PyTorch training script based on
all previous agents' outputs.

        Dataset Information:
        {dataset_info}

        Review the data analysis, architecture design, and hyperparameter
recommendations.

        The script must:
        1. Implement the proposed architecture exactly as specified
        2. Use the recommended hyperparameters
        3. Include data loading (support MNIST and CIFAR-10 from torchvision,
or custom dataset loading)
        4. Implement training loop with proper logging
        5. Include validation/testing
        6. Save/load model checkpoints
        7. Have clear comments and documentation
        8. Be production-ready and executable

        The code should be complete and executable. Include all necessary
imports and helper functions.
        Wrap your code output in ```python code blocks.
```

```
        Start with "TRAINING SCRIPT:" header.
        """,
        agent=agents['codegen_agent'],
        expected_output="Complete PyTorch training script ready to run,
wrapped in code blocks",
        context=[analysis_task, design_task, hyperparam_task]  # Can see all
previous outputs
    )

    # Create crew with sequential execution
    # Configure LLM at crew level for better compatibility
    crew = Crew(
        agents=[
            agents['data_analyst'],
            agents['model_designer'],
            agents['hyperparam_agent'],
            agents['codegen_agent']
        ],
        tasks=[
            analysis_task,
            design_task,
            hyperparam_task,
            codegen_task
        ],
        process=Process.sequential,  # Run tasks sequentially so each can use
previous output
        verbose=True
    )

    # Execute the crew
    result = crew.kickoff(inputs=context)

    # Extract results from task outputs
    full_output = str(result)

    # Try to extract structured sections from the output
    data_analysis = extract_section_by_header(full_output, "DATA ANALYSIS:")
    architecture = extract_section_by_header(full_output, "ARCHITECTURE
DESIGN:")
    hyperparameters = extract_section_by_header(full_output,
"HYPERPARAMETERS:")
    training_script = extract_code_block(full_output)

    # If extraction failed, try to get from individual task outputs
    if not data_analysis and hasattr(result, 'tasks_output'):
        try:
            data_analysis = str(result.tasks_output[0].raw) if
len(result.tasks_output) > 0 else full_output
        except:
            pass

    if not architecture and hasattr(result, 'tasks_output'):
        try:
            architecture = str(result.tasks_output[1].raw) if
len(result.tasks_output) > 1 else ""
        except:
            pass

    if not hyperparameters and hasattr(result, 'tasks_output'):
        try:
            hyperparameters = str(result.tasks_output[2].raw) if
len(result.tasks_output) > 2 else ""
        except:
            pass

    if not training_script and hasattr(result, 'tasks_output'):
        try:
            training_script = str(result.tasks_output[3].raw) if
len(result.tasks_output) > 3 else ""
        except:
            pass

    return {
        'data_analysis': data_analysis or full_output,
        'architecture': architecture or full_output,
        'hyperparameters': hyperparameters or full_output,
        'training_script': training_script or full_output,
        'full_output': full_output
    }


def extract_section_by_header(text: str, header: str) -> str:
    """Extract section content by header"""
    if not isinstance(text, str):
        text = str(text)

    header_index = text.find(header)
    if header_index == -1:
        return ""

    # Find the start of content after header
    content_start = header_index + len(header)

    # Find the next header or end of text
    next_headers = [
        "ARCHITECTURE DESIGN:",
        "HYPERPARAMETERS:",
        "TRAINING SCRIPT:",
        "DATA ANALYSIS:"
    ]

    next_header_pos = len(text)
    for h in next_headers:
        if h != header:
            pos = text.find(h, content_start)
            if pos != -1 and pos < next_header_pos:
                next_header_pos = pos

    section_content = text[content_start:next_header_pos].strip()
    return section_content


def extract_code_block(text: str) -> str:
    """Extract Python code block from text"""
    if not isinstance(text, str):
        text = str(text)

    # Look for code blocks
    import re
    pattern = r'```(?:python)?\s*\n(.*?)```'
    matches = re.findall(pattern, text, re.DOTALL)
```

```
    if matches:
        # Return the largest code block (likely the training script)
        return max(matches, key=len).strip()

    # If no code block found, look for "TRAINING SCRIPT:" section
    return extract_section_by_header(text, "TRAINING SCRIPT:")


def extract_section(raw_output: str, section: str) -> str:
    """Extract specific sections from crew output (legacy function)"""
    if isinstance(raw_output, str):
        return raw_output
    return str(raw_output)
```

```
// File: requirements.txt
flask
crewai[google-genai]
python-dotenv
werkzeug
```

```
// File: templates/index.html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Neural Network Hyperparameter Tuner</title>
    <!-- Markdown renderer -->
    <script src="https://cdn.jsdelivr.net/npm/marked/marked.min.js"></script>
    <!-- Syntax highlighting -->
    <link
      rel="stylesheet"
      href="https://cdnjs.cloudflare.com/ajax/libs/highlight.js/11.9.0/styles/
github-dark.min.css"
    />
    <script src="https://cdnjs.cloudflare.com/ajax/libs/highlight.js/11.9.0/
highlight.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/highlight.js/11.9.0/
languages/python.min.js"></script>
    <style>
      * {
        margin: 0;
        padding: 0;
        box-sizing: border-box;
      }

      body {
        font-family:
          -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, Oxygen,
Ubuntu,
          Cantarell, sans-serif;
        background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
        min-height: 100vh;
        padding: 20px;
      }

      .container {
        max-width: 1200px;
        margin: 0 auto;
        background: white;
        border-radius: 20px;
        box-shadow: 0 20px 60px rgba(0, 0, 0, 0.3);
        overflow: hidden;
      }

      .header {
        background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
        color: white;
        padding: 40px;
        text-align: center;
      }

      .header h1 {
        font-size: 2.5em;
        margin-bottom: 10px;
      }

      .header p {
        font-size: 1.1em;
        opacity: 0.9;
      }
```

```
.content {
  padding: 40px;
}

.dataset-selection {
  margin-bottom: 30px;
}

.dataset-options {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
  gap: 20px;
  margin-top: 20px;
}

.dataset-card {
  border: 2px solid #e0e0e0;
  border-radius: 12px;
  padding: 20px;
  cursor: pointer;
  transition: all 0.3s ease;
  text-align: center;
}

.dataset-card:hover {
  border-color: #667eea;
  transform: translateY(-5px);
  box-shadow: 0 10px 25px rgba(102, 126, 234, 0.2);
}

.dataset-card.selected {
  border-color: #667eea;
  background: #f0f4ff;
}

.dataset-card h3 {
  color: #667eea;
  margin-bottom: 10px;
}

.dataset-card p {
  color: #666;
  font-size: 0.9em;
}

.upload-section {
  margin-top: 30px;
  padding: 20px;
  border: 2px dashed #ccc;
  border-radius: 12px;
  text-align: center;
}

.upload-section.active {
  border-color: #667eea;
  background: #f0f4ff;
}

textarea {
  width: 100%;
  min-height: 150px;
  padding: 15px;
  border: 2px solid #e0e0e0;
  border-radius: 8px;
  font-family: "Courier New", monospace;
  font-size: 0.9em;
  resize: vertical;
  margin-top: 10px;
}

textarea:focus {
  outline: none;
  border-color: #667eea;
}

.submit-btn {
  background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
  color: white;
  border: none;
  padding: 15px 40px;
  font-size: 1.1em;
  border-radius: 8px;
  cursor: pointer;
  width: 100%;
  margin-top: 20px;
  transition: transform 0.2s ease;
}

.submit-btn:hover {
  transform: scale(1.02);
}

.submit-btn:disabled {
  opacity: 0.6;
  cursor: not-allowed;
}

.loading {
  display: none;
  text-align: center;
  padding: 40px;
}

.loading.active {
  display: block;
}

.spinner {
  border: 4px solid #f3f3f3;
  border-top: 4px solid #667eea;
  border-radius: 50%;
  width: 50px;
  height: 50px;
  animation: spin 1s linear infinite;
  margin: 0 auto 20px;
}

@keyframes spin {
  0% {
    transform: rotate(0deg);
  }
  100% {
    transform: rotate(360deg);
  }
}

.results {
  display: none;
  margin-top: 30px;
}

.results.active {
  display: block;
}

.result-section {
  background: #f8f9fa;
  border-radius: 12px;
  padding: 25px;
  margin-bottom: 20px;
}

.result-section h3 {
  color: #667eea;
  margin-bottom: 15px;
  display: flex;
  align-items: center;
  gap: 10px;
}

.result-content {
  background: white;
  padding: 20px;
  border-radius: 8px;
  border-left: 4px solid #667eea;
  max-height: 600px;
  overflow-y: auto;
  font-size: 0.95em;
  line-height: 1.7;
}

/* Markdown content styling */
.result-content h1,
.result-content h2,
.result-content h3,
.result-content h4 {
  margin-top: 1.5em;
  margin-bottom: 0.75em;
  color: #333;
  font-weight: 600;
}

.result-content h1 {
  font-size: 1.5em;
  border-bottom: 2px solid #e0e0e0;
  padding-bottom: 0.5em;
}
.result-content h2 {
  font-size: 1.3em;
}
.result-content h3 {
  font-size: 1.1em;
```

```css
}

.result-content p {
  margin-bottom: 1em;
  color: #444;
  line-height: 1.7;
}

.result-content ul,
.result-content ol {
  margin-left: 2em;
  margin-bottom: 1em;
  color: #444;
  line-height: 1.8;
}

.result-content li {
  margin-bottom: 0.5em;
}

.result-content strong {
  font-weight: 600;
  color: #333;
}

.result-content code:not(pre code) {
  background: #f4f4f4;
  padding: 2px 6px;
  border-radius: 3px;
  font-family: "Monaco", "Courier New", monospace;
  font-size: 0.9em;
  color: #e83e8c;
}

.result-content pre {
  background: #1e1e1e;
  padding: 0;
  border-radius: 8px;
  margin: 1em 0;
  overflow: hidden;
  position: relative;
}

.result-content pre:hover .copy-btn-small {
  opacity: 1;
}

.result-content pre code {
  background: transparent;
  padding: 1.2em;
  color: inherit;
  display: block;
  overflow-x: auto;
  font-size: 0.85em;
  line-height: 1.5;
}

.copy-btn-small {
  position: absolute;
  top: 8px;
  right: 8px;
  background: rgba(102, 126, 234, 0.9);
  color: white;
  border: none;
  padding: 6px 12px;
  border-radius: 4px;
  cursor: pointer;
  font-size: 0.75em;
  opacity: 0;
  transition: opacity 0.3s ease;
  z-index: 10;
}

.copy-btn-small:hover {
  background: #5568d3;
}

.copy-btn-small.copied {
  background: #28a745;
  opacity: 1;
}

.result-content blockquote {
  border-left: 4px solid #667eea;
  padding-left: 1em;
  margin: 1em 0;
  color: #666;
  font-style: italic;
}

.result-content table {
  width: 100%;
  border-collapse: collapse;
  margin: 1em 0;
}

.result-content table th,
.result-content table td {
  border: 1px solid #ddd;
  padding: 8px 12px;
  text-align: left;
}

.result-content table th {
  background: #f8f9fa;
  font-weight: 600;
}

.code-block-wrapper {
  position: relative;
  margin: 1em 0;
}

.code-block {
  background: #1e1e1e;
  color: #d4d4d4;
  padding: 1.2em;
  border-radius: 8px;
  overflow-x: auto;
  font-family: "Monaco", "Courier New", monospace;
  font-size: 0.85em;
  line-height: 1.6;
  margin: 0;
  position: relative;
}

.copy-btn {
  position: absolute;
  top: 10px;
  right: 10px;
  background: #667eea;
  color: white;
  border: none;
  padding: 8px 16px;
  border-radius: 6px;
  cursor: pointer;
  font-size: 0.85em;
  transition: all 0.3s ease;
  z-index: 10;
  display: flex;
  align-items: center;
  gap: 6px;
}

.copy-btn:hover {
  background: #5568d3;
  transform: translateY(-2px);
  box-shadow: 0 4px 8px rgba(102, 126, 234, 0.3);
}

.copy-btn.copied {
  background: #28a745;
}

.copy-btn svg {
  width: 16px;
  height: 16px;
}

.error {
  background: #fee;
  color: #c33;
  padding: 15px;
  border-radius: 8px;
  margin-top: 20px;
  border-left: 4px solid #c33;
}

.agents-info {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
  gap: 15px;
  margin-top: 20px;
}

.agent-badge {
  background: #f0f4ff;
  padding: 10px;
  border-radius: 8px;
  text-align: center;
  font-size: 0.9em;
  color: #667eea;
  border: 1px solid #667eea;
```

```
      }
    </style>
  </head>
  <body>
    <div class="container">
      <div class="header">
        <h1>Ø>Ýà Neural Network Hyperparameter Tuner</h1>
        <p>AI-Powered Architecture & Hyperparameter Recommendation System</p>
      </div>

      <div class="content">
        <div class="dataset-selection">
          <h2>Select Dataset</h2>
          <div class="dataset-options">
            <div class="dataset-card" data-type="mnist">
              <h3>Ø=ÜÊ MNIST</h3>
              <p>
                Handwritten digit recognition<br />28x28 grayscale images<br /
>10
                classes
              </p>
            </div>
            <div class="dataset-card" data-type="cifar">
              <h3>Ø=Ý¼þ  CIFAR-10</h3>
              <p>Object recognition<br />32x32 RGB images<br />10 classes</p>
            </div>
            <div class="dataset-card" data-type="upload">
              <h3>Ø=ÜÁ Custom Dataset</h3>
              <p>Upload your own dataset<br />summary or description</p>
            </div>
          </div>

          <div class="upload-section" id="uploadSection" style="display:
none">
            <h3>Dataset Summary</h3>
            <textarea
              id="datasetSummary"
              placeholder="Describe your dataset here:
- Data type (images, text, tabular)
- Input dimensions/shape
- Number of classes
- Dataset size
- Task type (classification, regression, etc.)"
            ></textarea>
          </div>

          <div class="agents-info">
            <div class="agent-badge">Ø=ÜÊ Data Analyst</div>
            <div class="agent-badge">Ø<ß×þ  Model Designer</div>
            <div class="agent-badge">&™þ  Hyperparameter Agent</div>
            <div class="agent-badge">Ø=Ü» Code Generator</div>
          </div>

          <button class="submit-btn" id="submitBtn"
onclick="analyzeDataset()">
            Ø=Þ€ Generate Neural Network Design
          </button>
        </div>

        <div class="loading" id="loading">
          <div class="spinner"></div>
          <h3>AI Agents are working...</h3>
          <p>
            This may take a minute. Our crew of 4 agents is analyzing your
            dataset and designing the optimal neural network.
          </p>
        </div>

        <div class="error" id="error" style="display: none"></div>

        <div class="results" id="results">
          <div class="result-section">
            <h3>Ø=ÜÊ Data Analysis</h3>
            <div class="result-content" id="dataAnalysis"></div>
          </div>

          <div class="result-section">
            <h3>Ø<ß×þ  Architecture Design</h3>
            <div class="result-content" id="architecture"></div>
          </div>

          <div class="result-section">
            <h3>&™þ  Hyperparameters</h3>
            <div class="result-content" id="hyperparameters"></div>
          </div>

          <div class="result-section">
            <h3>Ø=Ü» Training Script</h3>
            <div class="code-block-wrapper">
              <button
                class="copy-btn"
                onclick="copyCode('trainingScript')"
                title="Copy code"
              >
                <svg fill="none" stroke="currentColor" viewBox="0 0 24 24">
                  <path
                    stroke-linecap="round"
                    stroke-linejoin="round"
                    stroke-width="2"
                    d="M8 16H6a2 2 0 01-2-2V6a2 2 0 012-2h8a2 2 0 012 2v2m-6
12h8a2 2 0 002-2v-8a2 2 0 00-2-2h-8a2 2 0 00-2 2v8a2 2 0 002 2z"
                  ></path>
                </svg>
                <span>Copy</span>
              </button>
              <pre class="code-block" id="trainingScript"></pre>
            </div>
          </div>
        </div>
      </div>
    </div>

    <script>
      let selectedDataset = null;

      // Handle dataset card selection
      document.querySelectorAll(".dataset-card").forEach((card) => {
        card.addEventListener("click", () => {
          document
            .querySelectorAll(".dataset-card")
            .forEach((c) => c.classList.remove("selected"));
          card.classList.add("selected");
          selectedDataset = card.dataset.type;

          const uploadSection = document.getElementById("uploadSection");
          if (selectedDataset === "upload") {
            uploadSection.style.display = "block";
          } else {
            uploadSection.style.display = "none";
          }
        });
      });

      async function analyzeDataset() {
        if (!selectedDataset) {
          alert("Please select a dataset type");
          return;
        }

        const datasetSummary =
document.getElementById("datasetSummary").value;

        if (selectedDataset === "upload" && !datasetSummary.trim()) {
          alert("Please provide a dataset summary");
          return;
        }

        // Show loading, hide results and error
        document.getElementById("loading").classList.add("active");
        document.getElementById("results").classList.remove("active");
        document.getElementById("error").style.display = "none";
        document.getElementById("submitBtn").disabled = true;

        try {
          const response = await fetch("/api/analyze", {
            method: "POST",
            headers: {
              "Content-Type": "application/json",
            },
            body: JSON.stringify({
              dataset_type: selectedDataset,
              dataset_summary: datasetSummary,
            }),
          });

          const data = await response.json();

          if (data.success) {
            // Render markdown content
            renderMarkdown(
              "dataAnalysis",
              data.result.data_analysis || data.result.full_output,
            );
            renderMarkdown(
              "architecture",
              data.result.architecture || "See full output",
            );
            renderMarkdown(
              "hyperparameters",
              data.result.hyperparameters || "See full output",
            );

            // Render code with syntax highlighting
            renderCode(
              "trainingScript",
              data.result.training_script || data.result.full_output,
            );

            document.getElementById("results").classList.add("active");
          } else {
            throw new Error(data.error || "Unknown error occurred");
          }
        } catch (error) {
          document.getElementById("error").textContent =
            `Error: ${error.message}`;
          document.getElementById("error").style.display = "block";
        } finally {
          document.getElementById("loading").classList.remove("active");
          document.getElementById("submitBtn").disabled = false;
        }
      }

      // Configure marked for better markdown rendering
      marked.setOptions({
        breaks: true,
        gfm: true,
        headerIds: false,
        mangle: false,
      });

      function renderMarkdown(elementId, content) {
        const element = document.getElementById(elementId);
        if (!content || content.trim() === "") {
          element.innerHTML =
            '<p style="color: #999; font-style: italic;">No content
available</p>';
          return;
        }

        // Clean up the content - remove markdown headers if present
        let cleanedContent = content.trim();

        // Convert to HTML using marked
        try {
          const html = marked.parse(cleanedContent);
          element.innerHTML = html;

          // Highlight any code blocks in the markdown and add copy buttons
          element.querySelectorAll("pre code").forEach((block) => {
            hljs.highlightElement(block);
            // Add copy button to parent pre element if not already present
            const pre = block.parentElement;
            if (pre && !pre.querySelector(".copy-btn-small")) {
              const copyBtn = document.createElement("button");
              copyBtn.className = "copy-btn-small";
              copyBtn.textContent = "Copy";
              copyBtn.onclick = (e) => {
                e.stopPropagation();
                copyCodeFromPre(pre);
              };
              pre.appendChild(copyBtn);
            }
          });
        } catch (e) {
```

```
      console.error("Error rendering markdown:", e);
      element.innerHTML = "<pre>" + escapeHtml(cleanedContent) + "</pre>";
    }
  }

  function renderCode(elementId, content) {
    const element = document.getElementById(elementId);
    if (!content || content.trim() === "") {
      element.textContent = "# No code available";
      return;
    }

    // Clean up code - remove markdown code fences if present
    let cleanedCode = content.trim();

    // Remove ```python or ``` if at the start/end
    cleanedCode = cleanedCode.replace(/^```(?:python)?\s*\n?/i, "");
    cleanedCode = cleanedCode.replace(/\n?```\s*$/, "");

    // Create code element
    const codeElement = document.createElement("code");
    codeElement.className = "language-python";
    codeElement.textContent = cleanedCode;

    element.innerHTML = "";
    element.appendChild(codeElement);

    // Highlight the code
    hljs.highlightElement(codeElement);
  }

  function copyCode(elementId) {
    const element = document.getElementById(elementId);
    const codeElement = element.querySelector("code") || element;
    const text = codeElement.textContent || codeElement.innerText;

    navigator.clipboard
      .writeText(text)
      .then(() => {
        const btn = event.target.closest(".copy-btn");
        const originalText = btn.querySelector("span").textContent;
        btn.classList.add("copied");
        btn.querySelector("span").textContent = "Copied!";

        setTimeout(() => {
          btn.classList.remove("copied");
          btn.querySelector("span").textContent = originalText;
        }, 2000);
      })
      .catch((err) => {
        console.error("Failed to copy:", err);
        alert("Failed to copy code. Please select and copy manually.");
      });
  }

  function copyCodeFromPre(preElement) {
    const codeElement = preElement.querySelector("code");
    const text = codeElement
      ? codeElement.textContent
      : preElement.textContent;
    const btn = preElement.querySelector(".copy-btn-small");
```

```
        navigator.clipboard
          .writeText(text)
          .then(() => {
            const originalText = btn.textContent;
            btn.classList.add("copied");
            btn.textContent = "Copied!";

            setTimeout(() => {
              btn.classList.remove("copied");
              btn.textContent = originalText;
            }, 2000);
          })
          .catch((err) => {
            console.error("Failed to copy:", err);
            alert("Failed to copy code. Please select and copy manually.");
          });
      }

      function escapeHtml(text) {
        const div = document.createElement("div");
        div.textContent = text;
        return div.innerHTML;
      }
    </script>
  </body>
</html>
```