# **Project Title: Product Price Prediction System**

## **Problem Statement & Solution Architecture**

### **1. Project Vision & Objective**

To build an intelligent system that accurately predicts product prices based on product names and descriptions using fine-tuned language models. The system enables businesses and consumers to get instant price estimates for various products without manual market research.

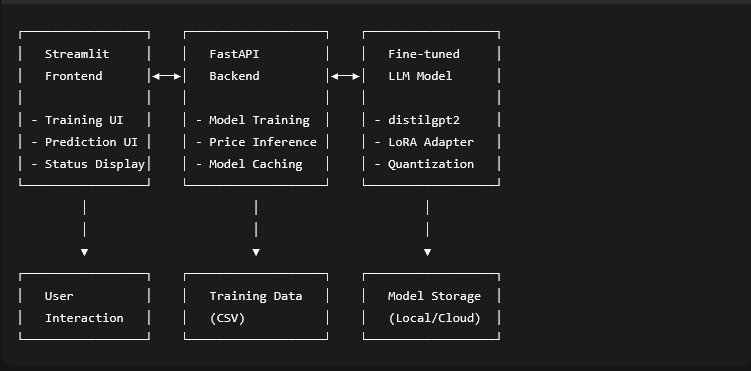
### **2. Core Problem**

Determining appropriate pricing for products is challenging due to market volatility, competition, and numerous product variations. Manual price research is time-consuming and inconsistent across different platforms and regions.

This application solves this by:

1. **Automating Price Prediction**: Using fine-tuned language models to analyze product attributes and generate accurate price estimates
2. **Streamlined Training Pipeline**: Providing an automated workflow to train custom models on product datasets
3. **Real-time Inference**: Offering instant price predictions through a user-friendly web interface
4. **Scalable Architecture**: Supporting both training and inference through separate, optimized components

### **3. High-Level Architecture & Workflow**



### **4. Step-by-Step Data Flow**

#### **Training Workflow:**

1. **User Initiates Training**: User clicks "Start Training" button in Streamlit frontend
2. **Backend Receives Request**: FastAPI receives POST request to /train endpoint
3. **Background Processing**: Training task is queued in background using BackgroundTasks
4. **Data Preparation**:
   * Loads product dataset from CSV file
   * Cleans and validates product names, descriptions, and prices
   * Creates formatted prompts for training
5. **Model Fine-tuning**:
   * Loads base distilgpt2 model with 4-bit quantization
   * Applies LoRA (Low-Rank Adaptation) for efficient training
   * Trains model on product-price mapping data
6. **Model Saving**: Saves fine-tuned adapter to persistent storage

#### **Prediction Workflow:**

1. **User Input**: User enters product name and description in Streamlit UI
2. **Request Sent**: Frontend sends POST request to /predict endpoint with product data
3. **Model Loading**:
   * Backend checks if model is cached in memory
   * If not cached, loads base model and LoRA adapter
   * Configures tokenizer and model for inference
4. **Prompt Construction**: Creates structured prompt matching training format
5. **Inference**:
   * Tokenizes input prompt
   * Generates price prediction using constrained text generation
   * Extracts numerical price from model output using regex patterns
6. **Response Processing**: Validates and formats predicted price
7. **Result Display**: Returns predicted price to frontend for user display

### **5. Technical Components**

#### **Frontend (Streamlit):**

* **Training Interface**: Start/stop model training with progress indicators
* **Prediction Interface**: Input forms for product details and price display
* **Status Monitoring**: Real-time backend connectivity checks
* **Debug Information**: Detailed logging for troubleshooting

#### **Backend (FastAPI):**

* **REST API**: RESTful endpoints for training and prediction
* **Model Management**: Caching and lifecycle management of ML models
* **Background Processing**: Asynchronous training job execution
* **Error Handling**: Comprehensive error handling and logging

#### **ML Pipeline:**

* **Base Model**: distilgpt2 (lightweight GPT-2 variant)
* **Fine-tuning Method**: QLoRA (Quantized Low-Rank Adaptation)
* **Training Optimization**: 4-bit quantization, gradient checkpointing
* **Prompt Engineering**: Structured prompt format for price learning
* **Price Extraction**: Multi-pattern regex parsing with validation

### **6. Key Features**

1. **Efficient Fine-tuning**: Uses QLoRA for parameter-efficient adaptation
2. **Model Caching**: Reduces inference latency by caching loaded models
3. **Background Training**: Non-blocking training process
4. **Robust Price Parsing**: Multiple fallback strategies for price extraction
5. **Comprehensive Logging**: Detailed logs for debugging and monitoring
6. **Error Resilience**: Graceful handling of model and connection failures

### **7. Performance Optimizations**

* **Quantization**: 4-bit model weights for reduced memory usage
* **LoRA**: Targeted parameter updates instead of full fine-tuning
* **Model Caching**: Avoids repeated model loading
* **Batch Processing**: Optimized data loading for training
* **Hardware Awareness**: Automatic CUDA detection and configuration