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

## **1.Issues Faced & Resolution Log**

### **● Inconsistent Prompt Formatting Between Training and Inference**

**Problem**: The model received differently formatted prompts during training vs inference, confusing its understanding.

**Root Cause**: Training used a simplified format while inference added extra formatting, creating a distribution shift.

**Solution**:

* **Unified Prompt Format**: Standardized on clear instruction format:  
  text

**Consistent Tokenization**: Ensured same tokenization parameters (max\_length=256, same padding) across both phases

### **● Poor Model Performance with Limited Data**

**Problem**: With only 100 training samples, the model failed to learn meaningful price-feature relationships.

**Root Cause**:

1. **Too Many Feature Variations**: RAM options from 2GB-16GB, storage from 32GB-1024GB created too many combinations
2. **Unclear Pricing Logic**: Complex, non-linear pricing formulas confused the model
3. **Excessive Random Variation**: ±5% price variation introduced noise

**Solution**:

* **Focused Feature Set**: Reduced to 2-3 clear options per category:  
  python

ram\_options = [8, 16] # Only 2 distinct options

storage\_options = [256, 512] # Only 2 distinct options

**Linear Pricing Formulas**: Implemented clear, linear relationships:

**Minimal Variation**: Reduced random variation to ±2% for clearer patterns

### **● CUDA vs CPU Compatibility Issues**

**Problem**: The system threw warnings about "8-bit optimizer not available" when running on CPU, and mixed precision issues.

**Root Cause**: The training configuration assumed CUDA availability with specific optimizations that weren't CPU-compatible.

**Solution**:

* **Conditional Configuration**: Implemented device-aware settings:  
  python

**Fallback Optimizers**: Used standard AdamW instead of 8-bit optimizers for CPU compatibility

### **● Slow Training on Consumer Hardware**

**Problem**: Training took excessively long on CPU and consumed high memory, making iteration cycles slow.

**Root Cause**: Using full sequence lengths and large batch sizes unsuitable for CPU training.

**Solution**:

* **Optimized Training Parameters**:

max\_length=256 # Reduced sequence length

per\_device\_batch\_size=2 # Smaller batches for CPU

gradient\_accumulation\_steps=1 # No accumulation for speed

num\_train\_epochs=10 # Fewer epochs for small dataset

**Disabled Expensive Features**: Turned off gradient checkpointing and reduced logging frequency

### **● Unrealistic Price Distributions**

**Problem**: Generated prices didn't follow realistic market distributions, confusing the model's learning.

**Root Cause**: Synthetic data generation created uniformly distributed prices without considering real-world price clustering.

**Solution**:

* **Market-Realistic Pricing**: Implemented tiered pricing:
  + Entry-level: $400-600 (basic specs)
  + Mid-range: $600-900 (balanced specs)
  + Premium: $900-1200 (high-end specs)
* **Clear Upgrade Pricing**: Established consistent upgrade costs ($50 per RAM tier, etc.)

## **2. Key Lessons Learned**

### **● Data Quality Over Quantity**

For small-scale fine-tuning, 100 well-designed samples with clear, repetitive patterns outperform 1000 noisy samples. The key is creating data that teaches specific relationships through repetition.

### **● Prompt Consistency is Critical**

Even small differences between training and inference prompts can completely derail model performance. Maintaining identical formatting across all phases is non-negotiable.

### **● Start Simple, Then Scale**

Beginning with minimal feature variations (2 options per feature) establishes clear learning patterns before introducing complexity.

### **● Price Extraction Requires Defense-in-Depth**

Relying on a single regex pattern for price extraction is fragile. Multiple fallback strategies with sanity checks are essential for production reliability.

### **● Clear Error Messages Accelerate Debugging**

Specific, actionable error messages for common issues (model incompatibility, missing files) dramatically reduce debugging time.

### **● Hardware-Aware Configuration**

Assuming GPU availability creates deployment issues. Conditional configuration based on available hardware ensures broader compatibility.

### **● Pattern Repetition Beats Data Volume**

For small datasets, repeating clear patterns multiple times is more effective than having many slightly different examples.

### **● Validation Throughout the Pipeline**

Regular validation of data generation, model outputs, and price extraction at each stage prevents compounding errors.