# High-Velocity Multimodal Pricing Blueprint (48-Hour Execution Plan)

## I. Executive Strategy: The High-Impact Framework

The success of a time-constrained machine learning challenge, particularly one involving disparate data modalities like text and imagery, hinges on maximizing feature quality while minimizing computational overhead associated with deep learning training. The proposed strategy is centered on a high-throughput, GPU-accelerated pipeline that leverages transfer learning for feature generation and an efficient Gradient Boosting Decision Tree (GBDT) model for rapid regression.

### A. Non-Negotiable Strategic Pillars

The blueprint rests on three critical strategic pillars designed to ensure maximum accuracy within the 48-hour timeframe.

#### 1. Transfer Learning for Feature Extraction

Training deep multimodal models end-to-end is computationally prohibitive within a two-day window. Therefore, the strategy mandates the use of pre-trained models exclusively as high-quality feature extractors. For text, lightweight, performant transformer architectures like MiniLM are favored over larger BERT variants, as they provide robust semantic representations with faster inference times.[1] For visual features, Contrastive Language-Image Pre-training (CLIP) models are selected because they are uniquely suited to extracting features relevant to catalog content, providing dense visual vectors (e.g., 512 dimensions) that capture high-level semantic meaning.[3] Bypassing fine-tuning allows the process to

immediately generate a dense, information-rich tabular dataset suitable for rapid training.

### 2. Target Transformation for Metric Alignment

The evaluation metric, Symmetric Mean Absolute Percentage Error (SMAPE), heavily penalizes large relative errors. Directly optimizing a GBDT model, which typically uses a Mean Squared Error (MSE) objective (L2 Loss) by default for regression [5], on raw price data leads to poor performance on skewed distributions. To mitigate variance and align the optimization objective more closely with percentage errors, a logarithmic transformation is mandatory. The target variable is transformed as . This standard practice in pricing challenges converts the skewed target distribution into a more Gaussian distribution, allowing the LightGBM model to converge faster and achieve lower RMSE on the log scale, which translates to better SMAPE performance upon inverse transformation.[6]

### 3. LightGBM GPU Acceleration for High-Dimensional Data

The combined feature set (text and image embeddings) will result in a high-dimensional tabular matrix (approximately 896 dimensions). LightGBM is chosen over other boosting frameworks due to its superior speed and memory efficiency achieved through histogram-based learning.[7] Critically, configuring LightGBM to utilize the available GPU hardware (device_type='cuda') ensures that training and validation processes, which occur on Day 2, are not bottlenecked by CPU processing, allowing for rapid iteration and completion of the training phase.[8]

## B. The 2-Day High-Velocity Action Plan

The execution schedule is designed to front-load I/O and feature extraction, which constitute the primary potential bottlenecks, ensuring Day 2 is reserved strictly for modeling and submission finalization.

Table: 48-Hour High-Velocity Action Plan

| Stage | Task Description | Estimated Hours | Day | Key Deliverables |
|---|---|---|---|---|
| **P0: Setup** | Environment config, library installs (PyTorch, HF, LightGBM). | 0.5h | Day 1 | Executable environment. |
| **D1.1: Data Ingest & Target Prep** | Load CSVs, initial text cleaning, apply transformation . | 1.5h | Day 1 | Cleaned dataframes, Log-Target series . |
| **D1.2: Image Download (Parallel)** | Setup parallel file download utility (ThreadPoolExecutor). Run bulk image download to local disk/cache. | 4.0h | Day 1 | Local repository of product images. |
| **D1.3: Text Features** | Load fast BERT model (MiniLM), encode text content in batches, extract mean-pooled embeddings. | 3.0h | Day 1 | X_text_embeddings.npy. |
| **D1.4: Image Features** | Load CLIP Vision model, process downloaded images in GPU batches, | 5.0h | Day 1 | X_image_embeddings.npy. |

| | | | | |
|---|---|---|---|---|
| | extract and save embeddings. | | | |
| **D1.5: Feature Fusion & Persistence** | Concatenate text and image features. Save all feature matrices for rapid Day 2 reload. | 1.0h | Day 1 | Final feature matrices (High-D tabular data). |
| **Total Day 1 Work** | | **15.0h** | | |
| **D2.1: Model Prep & Metric** | Load features, define custom NumPy SMAPE function, finalize GPU LightGBM config. | 1.0h | Day 2 | Custom SMAPE function, Config dict. |
| **D2.2: Training & Validation** | Train LightGBM model using GPU, RMSE objective, and SMAPE evaluation with early stopping. | 3.0h | Day 2 | Trained LightGBM model artifact. |
| **D2.3: Prediction & Inverse Transform** | Generate test predictions. Apply np.exp() inverse transform to yield final prices. | 1.0h | Day 2 | Final prediction vector. |

| | | | | |
|---|---|---|---|---|
| **D2.4: Submission Finalization** | Format predictions into required test_out.csv (ID, Price). | 0.5h | Day 2 | Final submission file. |
| **D2.5: Optional Optimization** | Shallow Hyperparameter search or simple two-model ensembling. | 2.5h | Day 2 | Performance uplift/backup submission. |
| **Total Day 2 Work** | | **8.0h** | | |

# II. Day 1: High-Velocity Data Acquisition and Feature Engineering

## A. Data Preprocessing and I/O Bottleneck Mitigation

Data ingress begins with loading the train.csv and test.csv files. The initial data preparation involves minimal text cleaning, primarily focused on filling any potential missing values in the catalog_content columns with a standardized token (e.g., ``) to prevent downstream transformer model errors. Crucially, the target variable  (price) is immediately transformed using .

The most significant time constraint on Day 1 is the acquisition of image data from external URLs (image_link). Fetching thousands of images sequentially suffers severely from network latency. To overcome this critical I/O bottleneck, parallel processing is implemented.[10] Python's concurrent.futures.ThreadPoolExecutor is utilized with the requests library to execute multiple download tasks concurrently across several threads.[10] This mechanism ensures that while one thread is waiting for a network response, others are actively initiating or completing downloads, dramatically reducing the overall wall clock time for data

acquisition.[10] The images are downloaded, opened using libraries like Pillow to confirm integrity, and saved locally, typically keyed by product ID, which simplifies subsequent feature extraction.

## B. Text Feature Extraction

The text feature extraction utilizes a fast transformer model, such as MiniLM, which provides a balance between rapid processing and high-quality semantic information. The text input, consisting of the concatenated title + description + quantity, is tokenized using the Hugging Face AutoTokenizer.

The tokenized data is processed in batches on the available GPU hardware. Since LightGBM requires fixed-size feature vectors, the dynamic sequence of token embeddings produced by the transformer is converted into a single, dense vector through **Mean Pooling**.[1] Mean pooling involves calculating the arithmetic average of all token embeddings for a given sequence, excluding contributions from padding tokens via the attention mask.[2] This process yields robust, fixed-size text embeddings (e.g., 384 dimensions), which are subsequently saved as a NumPy array for efficient loading on Day 2.

## C. Image Feature Extraction

The image feature extraction pipeline is designed around the powerful CLIP model, specifically the Vision Transformer (ViT) architecture, which produces highly generalized 512-dimensional visual features.[3] This model is ideal for general computer vision tasks and transfers well to e-commerce product recognition.

The process involves iterating through the list of locally saved image files. Each image must be loaded (e.g., using PIL/Pillow for standardized image handling [11]), standardized, and prepared for the CLIP model using CLIPProcessorFast. This processor handles essential preprocessing steps like resizing and normalization automatically, ensuring the image data conforms to the model's input requirements.[3] Encoding is performed in GPU-accelerated batches to maximize speed. A critical element of this stage is robust error handling: if an image file is missing, corrupt, or failed to download, the pipeline must insert a zero vector of the corresponding embedding dimension (e.g., 512 zeros) into the feature matrix. This ensures that the final feature matrices remain perfectly aligned with the product IDs, preventing alignment errors during model training.

### D. Feature Fusion and Data Persistence

Upon completion of the text and image extraction phases, the resulting dense feature matrices ( and ) are fused using simple vector concatenation (np.hstack). This results in the final, unified high-dimensional tabular feature set (, 896 dimensions).

This simple concatenation approach is highly effective for GBDT models, as LightGBM can efficiently traverse and select features from high-dimensional space.[13] To safeguard the 15 hours of Day 1 work against notebook timeouts or environment resets, the final feature matrices (, ) and the log-transformed target vector () are persistently saved to disk (e.g., using NumPy's .npy format).

## III. Day 2: Multimodal Training, SMAPE, and Submission Protocol

Day 2 focuses entirely on loading the pre-computed features and executing the model training and prediction stages with precision.

### A. Defining the Custom SMAPE Metric

The LightGBM model will be trained using the default RMSE objective (regression_l2) on the log-transformed target, which provides a fast and numerically stable training process.[5] However, the competitive metric is SMAPE. To ensure the model training tracks the required metric and utilizes effective early stopping, a custom evaluation function (feval) is necessary.[14]

#### 1. SMAPE Implementation Nuances

The mathematical definition of the Symmetric Mean Absolute Percentage Error (SMAPE) is critical. In its unscaled form (not multiplied by 100), it is defined by:

$ \text{SMAPE} = \frac{1}{N} \sum_{i=1}^{N} \frac{|F_i - A_i|}{(|A_i| + |F_i|) / 2} $
Where  is the actual value and  is the forecast value.[15] The Python implementation requires including a small numerical epsilon () in the denominator to prevent division by zero errors, especially if prices could be zero.

## 2. The Log-Scale Conversion Requirement

A significant pitfall in competition pipelines utilizing log transformation is calculating the final metric on the wrong scale. Since LightGBM predicts the log-transformed target, the input y_pred passed into the custom evaluation function is on the  scale. The true SMAPE, however, must be calculated on the raw, linear price scale.

Therefore, the custom LightGBM feval wrapper must perform a *two-step conversion* internally:

1. Access the true labels (y_true = train_data.get_label()).[16]
2. Inverse transform both the true labels and the predictions back to the linear price scale using .[6]
3. Calculate the SMAPE using these inverse-transformed, non-negative actual prices.

Python

```
import numpy as np
import lightgbm as lgb

def smape_numpy(y_true, y_pred):
    # Implementation of SMAPE calculation
    EPSILON = 1e-6
    numerator = np.abs(y_pred - y_true)
    denominator = (np.abs(y_true) + np.abs(y_pred)) / 2.0
    return np.mean(numerator / (denominator + EPSILON))

def lgbm_smape_eval(y_pred, train_data):
    # Retrieve log-transformed true labels
    y_true = train_data.get_label()

    # Inverse Transform
    y_true_actual = np.exp(y_true) - 1
```

```
    y_pred_actual = np.exp(y_pred) - 1

    # Clamp to ensure non-negative prices for SMAPE
    y_true_actual = np.maximum(0, y_true_actual)
    y_pred_actual = np.maximum(0, y_pred_actual)

    score = smape_numpy(y_true_actual, y_pred_actual)

    # Return LightGBM standard output (metric_name, metric_value, is_higher_better)
    return 'SMAPE', score, False
```

## B. LightGBM GPU Training Blueprint

The final model training employs LGBMRegressor configured for maximum speed and efficiency in the Google Colab GPU environment.

Table: LightGBM GPU Optimized Configuration

| Parameter | Value | Rationale |
|---|---|---|
| objective | 'regression_l2' | Standard training objective for fast gradient calculation. [5] |
| device_type | 'cuda' | Mandatory GPU acceleration, using optimized 32-bit floats by default. [8] |
| n_estimators | 2000 | Sufficient maximum iterations, controlled by early stopping. [8] |
| learning_rate | 0.04 | A moderate setting to ensure quick convergence without overshooting the minimum. [17] |

| | | |
|---|---|---|
| num_leaves | 63 | Maximum number of leaves, balancing complexity with rapid tree growth. [8] |
| max_bin | 63 | Recommended smaller bin size for better GPU speed-up. [8] |
| n_jobs | -1 | Utilizes all available CPU cores for non-GPU operations. [8] |

The model is trained using an appropriate train/validation split (e.g., 90% train, 10% validation). The custom lgbm_smape_eval function is passed to the eval_metric parameter, allowing the model to track real competition performance during training. Early stopping (e.g., 100 rounds) is implemented to prevent overfitting and rapidly determine the optimal number of boosting iterations, crucial for meeting the submission deadline.[18]

### C. Prediction and Submission Protocol

Once trained, the model generates predictions on the test set features (), resulting in log-transformed price forecasts. The crucial final step is the inverse transformation.

The raw log predictions are converted back to the linear price scale: final_predictions = np.exp(log_predictions) - 1.[6] It is imperative to clamp all resulting predictions to zero or above (np.maximum(0, final_predictions)) to ensure no nonsensical negative prices enter the submission file.

The final predictions are then formatted into the required submission file, typically test_out.csv.[19] This file must contain the product identifier (ID) and the final predicted price column, correctly named (e.g., 'price').[20] Indexing must be disabled during the CSV export to prevent extraneous columns, and the order of predictions must strictly match the order of IDs in the original test set data.

## IV. Optional Performance Boosters and Final

# Verification

## A. Rapid Optimization Strategies (D2.5)

If time remains on Day 2, rapid performance enhancements can be deployed:

1. **Shallow Hyperparameter Optimization:** Instead of deep grid searches, a shallow randomized search focusing on key parameters like learning_rate, num_leaves, and regularization parameters like feature_fraction can quickly identify improved parameter combinations.[13] Tuning these few parameters provides a high return on investment in a short time.
2. **Statistical Feature Augmentation:** The dense 896-dimensional embeddings can be augmented with basic statistical features (e.g., L2 norm, mean, standard deviation across dimensions). These features offer LightGBM scalar input that it can easily interpret and utilize alongside the high-dimensional vectors, often providing a small but reliable performance boost.
3. **Simple Model Ensembling:** Training a second, near-identical LightGBM model using only a different random seed, and then averaging the inverse-transformed predictions, is a highly effective way to reduce prediction variance without significant computational cost.

## B. Final Verification Checklists

The rigorous adherence to this plan requires the use of definitive checklists to ensure no step is overlooked prior to submission.

### What to do on Day 1 checklist

1. [X] Install PyTorch, Hugging Face, LightGBM (with GPU support).
2. [X] Load CSVs and ensure the  transformation is applied to .
3. [X] **Run Parallel Download Utility (D1.2).** Monitor for I/O errors and ensure local image storage is successful.

4. [X] Run Text Feature Extraction (MiniLM) using GPU batching. Save X_text_embeddings.npy.
5. [X] Run Image Feature Extraction (CLIP ViT) using GPU batching and robust error handling. Save X_image_embeddings.npy.
6. [X] Fuse  and  matrices. Save the final high-dimensional training and test feature matrices.

### What to do on Day 2 checklist

1. [X] Load all persistent feature matrices and .
2. [X] Implement and test the custom lgbm_smape_eval function (confirming inverse transform *inside* the function).
3. [X] Configure LightGBM with device_type='cuda' and optimized parameters (e.g., max_bin=63).
4. [X] Run primary training loop (D2.2) with early stopping based on the custom SMAPE metric.
5. [X] Generate test predictions and apply the final inverse transform: np.exp(log_predictions) - 1.
6. [X] Format and generate test_out.csv (ID, price).

### How to verify and finalize submission

1. **Format Check:** Confirm test_out.csv contains the correct number of rows (matching the test set size) and exactly two columns: ID and price.
2. **Scale Check:** Verify that all predicted values in the CSV are non-negative, linear prices (i.e., the inverse transform was successfully applied).
3. **ID Alignment:** Ensure the prediction order perfectly corresponds to the ID order established in the original test set data.[20]
4. **Final Model Selection:** Use the model that achieved the best SMAPE on the validation set during the early stopping phase, incorporating any simple ensembling results (D2.5) if applicable.

# V. Conclusion

The time-optimized multimodal pricing solution outlined above prioritizes speed and feature richness through the strategic use of transfer learning and GPU acceleration. By dedicating Day 1 to mitigating the I/O and feature extraction bottlenecks via parallel processing and compact transformer models, the pipeline ensures that high-quality, fused features are ready for rapid model training on Day 2. The critical step of using  transformation combined with a custom SMAPE evaluation metric ensures that the LightGBM GPU regressor optimizes against the competition standard, leading to the most robust and competitive result achievable within the strict 48-hour constraint.

**Works cited**


1. jianjieluo/BERTFeatureExtraction: BERT feature extraction based on state-of-the-art pre-trained models - GitHub, accessed on October 11, 2025, https://github.com/jianjieluo/BERTFeatureExtraction
2. Exploring BERT: Feature extraction & Fine-tuning | by Mouna Labiadh | DataNess.AI, accessed on October 11, 2025, https://medium.com/dataness-ai/exploring-bert-feature-extraction-fine-tuning-6d6ad7b829e7
3. CLIP - Hugging Face, accessed on October 11, 2025, https://huggingface.co/docs/transformers/en/model_doc/clip
4. Image Feature Extraction - Hugging Face, accessed on October 11, 2025, https://huggingface.co/docs/transformers/main/tasks/image_feature_extraction
5. LightGBM hyperparameters - Amazon SageMaker AI - AWS Documentation, accessed on October 11, 2025, https://docs.aws.amazon.com/sagemaker/latest/dg/lightgbm-hyperparameters.html
6. New to Kaggle? Here's How you can Get Started with Kaggle Competitions, accessed on October 11, 2025, https://www.analyticsvidhya.com/blog/2020/06/get-started-kaggle-competitions/
7. Simple_LightGBM_Integration.ipynb - Google Colab, accessed on October 11, 2025, https://colab.research.google.com/github/wandb/examples/blob/master/colabs/boosting/Simple_LightGBM_Integration.ipynb
8. Parameters — LightGBM 4.6.0.99 documentation - Read the Docs, accessed on October 11, 2025, https://lightgbm.readthedocs.io/en/latest/Parameters.html
9. LightGBM GPU baseline model for Google Colab - Kaggle, accessed on October 11, 2025, https://www.kaggle.com/code/slavikonnikov/lightgbm-gpu-baseline-model-for-google-colab
10. How to Download Files From URLs With Python – Real Python, accessed on October 11, 2025, https://realpython.com/python-download-file-from-url/
11. Image Processing With the Python Pillow Library, accessed on October 11, 2025, https://realpython.com/image-processing-with-the-python-pillow-library/
12. Feature Extractor - Hugging Face, accessed on October 11, 2025,

https://huggingface.co/docs/transformers/main_classes/feature_extractor
13. A LightGBM-Based Power Grid Frequency Prediction Method with Dynamic Significance–Correlation Feature Weighting - MDPI, accessed on October 11, 2025, https://www.mdpi.com/1996-1073/18/13/3308
14. lightgbm.LGBMClassifier — LightGBM 4.6.0.99 documentation - Read the Docs, accessed on October 11, 2025, https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMClassifier.html
15. Python Numpy functions for most common forecasting metrics - GitHub Gist, accessed on October 11, 2025, https://gist.github.com/bshishov/5dc237f59f019b26145648e2124ca1c9
16. How to handle error in Lightgbm with custom loss? - Stack Overflow, accessed on October 11, 2025, https://stackoverflow.com/questions/63919897/how-to-handle-error-in-lightgbm-with-custom-loss
17. Tune a LightGBM model - Amazon SageMaker AI, accessed on October 11, 2025, https://docs.aws.amazon.com/sagemaker/latest/dg/lightgbm-tuning.html
18. LightGBM Classification Project.ipynb - Colab, accessed on October 11, 2025, https://colab.research.google.com/github/pb111/Data-Science-Portfolio-in-Python/blob/master/LightGBM_Classification_Project.ipynb
19. MNIST Tutorial Machine Learning Challenge - Kaggle, accessed on October 11, 2025, https://www.kaggle.com/c/mnist-tutorial-machine-learning-challenge/data
20. How Should I submit my dataset on the house prediction competition, I got this error frequently | Kaggle, accessed on October 11, 2025, https://www.kaggle.com/discussions/questions-and-answers/327786