Approach Summary
Team Prime Coders

Overview

We framed product price prediction as a supervised regression problem on a rich, multi-modal tabular dataset. The final system combines deep learning models trained on learned text/image representations (Stella, SigLIP, Qwen, Gemma) and structured features with a tree-based booster (XGBoost). Predictions are produced on the log-price scale and mapped back via expm1. We report and optimize Symmetric Mean Absolute Percentage Error (SMAPE) on the original price scale.

Features and embeddings

- Structured ("manual") features: all non-embedding, non-target columns (including engineered and TF-IDF features `tfidf_*`).
- Vision embeddings: `siglip_*` (SigLIP image representations) and `stella_*` (text-image aligned textual representations).
- LLM text embeddings:
    - `qwen_*`: Qwen sentence embeddings merged by `sample_id`.
    - `gemma_*`: EmbeddingGemma sentence embeddings (trained and merged analogously).

We clip extreme values to [-1e5, 1e5] and scale each feature group independently with StandardScaler.

Models

- Deep-Wide MLPs (PyTorch):
    - Architecture: three branches (manual, embedding A, embedding B) each with Linear → ReLU → BatchNorm1d → Dropout, concatenated into a final MLP head.
    - Variants trained on:
        - Stella + SigLIP + manual/TF-IDF
        - Qwen + SigLIP + manual/TF-IDF
        - Gemma + SigLIP + manual/TF-IDF
    - Loss: HuberLoss on log-price; mixed precision (autocast + GradScaler) and gradient clipping used for stability.
- XGBoost: gradient-boosted trees on the clipped float32 feature matrix (log-target). We trained both squared-error and Pseudo-Huber objectives and selected the best via validation SMAPE.

Hyperparameter optimization

- We use Optuna to search MLP and XGBoost hyperparameters.
  - MLP search space: batch_size (256–1024), lr (1e-4–5e-3, log-uniform), weight_decay (1e-6–1e-3), dropout (0.1–0.3), hidden sizes for each branch and the fusion layer, and Huber delta.
  - XGBoost search space: max_depth, learning_rate, subsample, colsample_bytree, min_child_weight, L2 (lambda) and L1 (alpha) regularization; tree_method=gpu_hist and gpu_predictor for speed.
- Objective: minimize validation SMAPE computed after mapping predictions back to the price space via expm1.
- Early stopping on validation and model checkpointing were used to select the best trials.

Training and preprocessing

- Targets are transformed with log1p during training; predictions are inverted with expm1 for evaluation and submission.
- Per-group StandardScaler is fit on the training split for MLPs and applied consistently to validation. For XGBoost, we train on clipped float32 features without scaling.
- Robustness measures: NaN checks, replacement (0.0) in inference paths, gradient clipping (max_norm=5.0), and outlier clipping of inputs.

Ensembling

- At inference, we average model outputs on the log scale and then apply expm1.
- Typical ensemble composition:
  - 2× Deep-Wide MLPs (Stella + SigLIP)
  - 1× Deep-Wide MLP (Qwen + SigLIP)
  - 1× XGBoost model
- We also experimented with adding the Gemma+SigLIP branch; when unstable, we fall back to the 4-model ensemble.

Reproducibility and artifacts

- Training scripts: `MLP/stella+siglip+optimization.py`, `MLP/qwen+siglip+optimization.py`, `MLP/gemma+siglip+optimization.py`.
- Tree models: `Trees/tree2.py` (squared error) and `Trees/tree3.py` (Pseudo-Huber, float32).
- Saved models used in evaluation:
  - `MLP/optuna_best_48_46_siglip_stella.pt`, `MLP/optuna_best_48_22_siglip_stella.pt`
  - `MLP/optuna_best_siglip_qwen.pt`
  - `Trees/best_xgb_model_regression.json` or `Trees/best_xgb_huber_float32_model.json`
- Evaluation/ensemble scripts: `Trees/evaluate4.py` (no Gemma), `Trees/evaluate3.py` (with Gemma, architecture auto-infer and diagnostics).

This pipeline balances robustness and performance by blending complementary inductive biases: representation learning via Deep-Wide MLPs and non-linear partitioning via XGBoost, with careful preprocessing, clipping, and SMAPE-driven hyperparameter search.