Predicting Kitchen Order Preparation Time

Jayaprakash Sundararaj, hi@jayaprakash.net

September 2026

1. Executive Summary

Headline result: MAE 246.4 seconds on a temporal holdout (8,553 orders from Oct 23–28, trained on 34,212 orders from Oct 1–23). Against per-kitchen median: 18.4% reduction (302.0 s). Against global median: 31.3% reduction (358.6 s).

Estimated ceiling: Even with perfect information (kitchen, exact basket, time of day), predictions cannot go below ≈218 s due to inherent kitchen variability. This leaves under 30 seconds of potential improvement — nearly all remaining error is unavoidable operational variance, not a model limitation.

Findings:

- Data violates the IID assumption: predictions depend on prior kitchen activity and historical patterns.

- Inherent trade-off between early prediction (risk of cold food) and late prediction (courier idle time).
- Simple features (kitchen busyness, recent order volume) with lightweight models achieve competitive performance.
- On baskets dominated by rare items, LightGBM (299.7 s) is beaten by the kitchen-median (294.4 s) — the only slice where it loses.
- Per kitchen, LightGBM beats the kitchen-median in 43 of the 46 holdout kitchens; the three exceptions are specific and diagnosable (two thin cells, one cold-start kitchen) rather than a diffuse weakness (§7.3).
- NOTE: Transformer fine-tuning excels on rare items and high-variance kitchen behaviors where simple models struggle.
- NOTE: Dataset size and complexity limit effectiveness of larger models (DNN, LSTM); overparameterization leads to overfitting.
- NOTE: The real-time preparation time of last few orders would be helpful. It is not used because of it will be missing in the TEST data.

1.1. Quick Stats

Metric	Value
Total dataset	42,934 orders (after dedup)
Training split	34,212 orders (Oct 1–23)
Holdout (temporal)	8,553 orders (Oct 23–28)
Kitchens	46 unique
Days of data	28
Features	57 tabular + optional text
Best model	LightGBM + text (SVD 64)
Best MAE	246.4 s
Bias	+24.0 s (under predicts)

2. Problem Statement

The task is to predict how long a kitchen will take to prepare an order in real time, given a JSON object with order metadata (kitchen ID, basket items, timestamp, subtotal), and output a scalar prediction of `prep_time_seconds` representing the actual cooking duration end-to-end. This prediction powers operations dispatch systems and customer ETAs; The problem has strict constraints: we must use temporal splits (train on past, validate on future with no peeking), we have no outcome data available at request time (no completion times in the production JSON),

and features must be strictly causal so row i can only read activated_at and metadata from rows before it. The challenge is significant — prep time ranges from 45 s to 3,597 s, per-kitchen medians span 137 s to 1,341 s, and kitchen identity alone explains only $R^2 = 0.28$, leaving substantial room for improvement through better features and models.

3. Data

Raw file: `orders.json` with 62,934 records. Six findings, each of which changed the implementation:

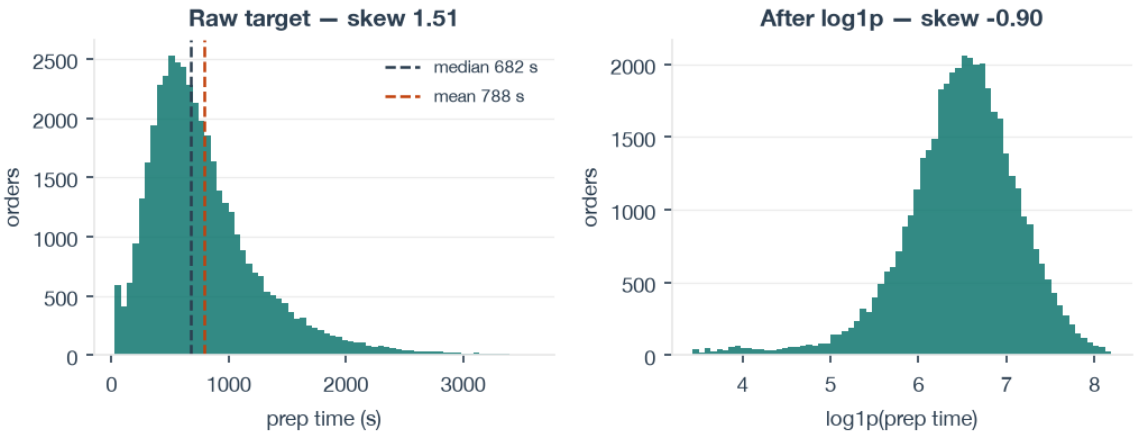
Finding	Issue	Action	Impact
1. Web duplicates	20,000 records with `import_source: "web"` are byte-identical to their `"app"` twins on all other fields. All 20,000 match exactly.	Drop all web rows; add `import_source` to `FORBIDDEN_FEATURE_COLUMNS`	Prevents ~47% inflation of load features; avoids splitting an order from its twin across train/test
2. Order ID collisions	398 `order_id` values appear under different kitchens (8-hex collisions)	Dedup key = `(order_id, kitchen_id, activated_at)`	Deduplicating on `order_id` alone would delete real orders
3. Bad timestamps	169 rows with malformed `activated_at` or `prep_time_seconds` ∉ [30, 3600] s	Drop 169 rows	Ensures causal ordering and plausible targets
4. Malformed item JSON	Item names like `["12\" zen7"]` are neither valid JSON nor Python (unescaped inch-marks)	3-tier tolerant parser: `json.loads` → `ast.literal_eval` → regex fallback	Never raises; recovers tokens even from garbage
5. Corrupt subtotal	Median $20.35, p99 $18,936, max $89,000	Clip at $500; flag clipped rows	Unclipped: −0.011 correlation. Clipped: +0.039. Makes the feature usable.

Finding	Issue	Action	Impact
6. Right-skewed target	Median 681 s, p99 2,467 s, max 3,597 s; skew = 1.51 (log1p: −0.90)	Offer log1p, identity, sqrt transforms (all perform identically in practice)	Skew does not matter for L1 objectives

Result: 62,934 → 42,934 cleaned orders (34.2% reduction). Temporal splits: 34,212 train (Oct 1–23), 8,553 holdout (Oct 23–28).

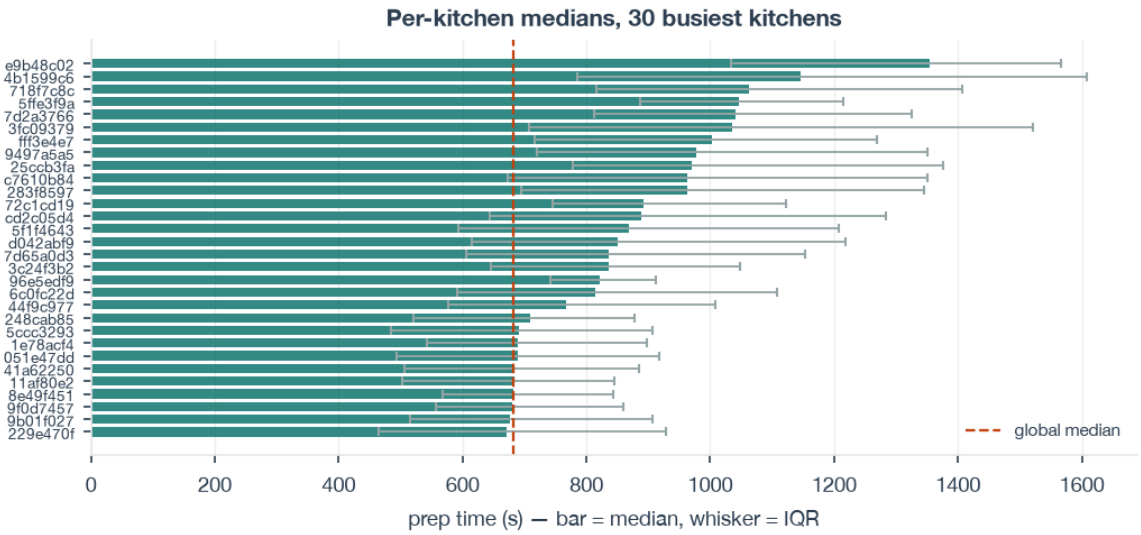
Exploratory Analysis

Three views of the cleaned 42,765-order table. Regenerate with `python -m preptime.data_analysis.figures`.



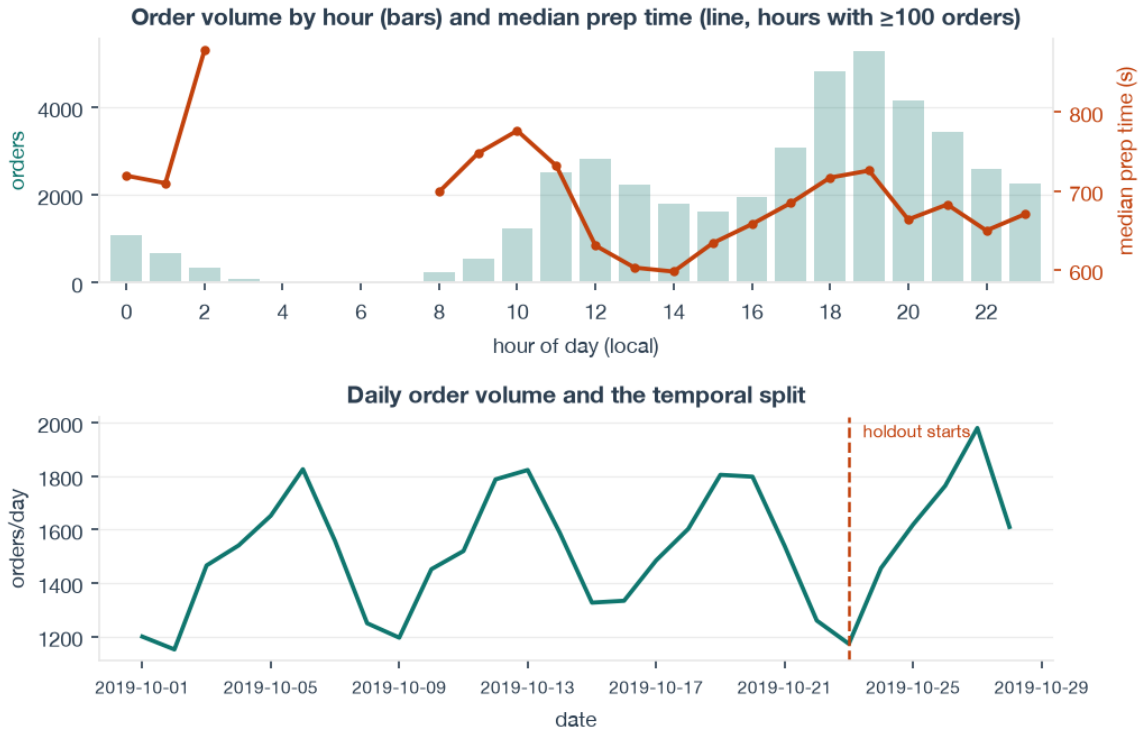
Distribution of prep time before and after the log1p transform

The raw target is strongly right-skewed: a dense mode near 600 s with a tail running to 3,597 s. The mean (788 s) sits 106 s above the median (682 s), and that gap is not cosmetic — it is the reason the production model carries a positive bias. An L1 objective is minimised by the conditional median, so a model trained to optimise MAE targets the lower of these two numbers while the holdout average reflects the higher one. The global-median's +107 s bias (§7.1, row 10) is almost exactly this 106 s gap, confirming the mechanism; features narrow it to +24.0 s but cannot erase it. The right panel shows log1p pulling the skew to −0.90, which is why the transform is offered — though as noted above it changes nothing under L1.



Per-kitchen median prep time with interquartile range, 30 busiest kitchens

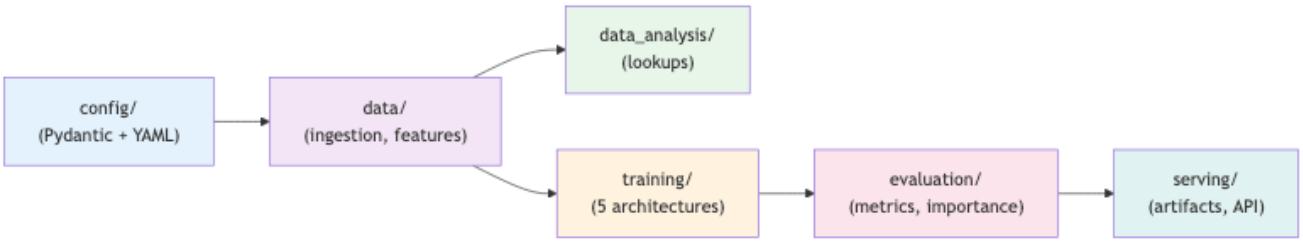
This is the $R^2 = 0.28$ figure from §2 in visual form. The bars spread widely — medians run from roughly 340 s to 1,354 s across the busiest kitchens — which is why kitchen identity is worth 28% of the variance and why the per-kitchen median is a credible baseline at 302.0 s MAE. But the IQR whiskers overlap almost completely: the median kitchen spans 376 s between its own 25th and 75th percentile. Knowing the kitchen narrows the standard deviation only from 485 s to 410 s, a 15% reduction. Most of the variation lives inside each kitchen, which is precisely the variation the load and basket blocks are built to capture.



Order volume by hour with median prep time, and daily volume across the 28-day window

The service window is roughly 11:00–23:00, with a pronounced dinner peak: hours 18–21 alone carry 41% of all orders. Hours 04:00–06:00 contain no orders at all, and hour 07:00 contains three — the median line is therefore drawn only across hours with at least 100 orders, since the thin hours produce values (hour 7's median is 1,740 s, on a sample of three) that are noise rather than signal. Within the supported range the prep-time signal is real but modest, dipping to about 600 s in the mid-afternoon lull and rising toward 730 s at the dinner peak — a spread of roughly 130 s, small next to the 376 s of within-kitchen variation above. This is consistent with the ablation result that the temporal block is dead weight (§7.1): hour-of-day carries genuine information, but the load features already encode it more directly through observed queue depth. The lower panel shows clean weekly seasonality in volume and marks the Oct 23 holdout boundary, confirming the split lands mid-cycle rather than isolating an unusual stretch.

4. Architecture



Design Principles

1. One configuration rules everything — `ExperimentConfig` threads through ingestion, feature building, training, evaluation, and serving. Serialised into every artifact.
2. FeatureBlock protocol — every feature is a class with `fit(train)` and `transform(batch)`. This makes features reusable across all architectures and enforces:
   - `fit` sees training data only
   - `transform` is row-causal (row i cannot see future rows)
   - Features are serialised and loaded identically at inference
3. Forbidden columns — `import_source`, `prep_time_seconds` (the label), `cooking_or_pick_completed_at` (future knowledge) are listed before any model sees them.
4. Lookup tables — empirical-Bayes smoothed per-kitchen, per-kitchen-hour, per-kitchen-dow, per-item, per-kitchen-basket-size. Fitted on train only, O(1) at inference.

Packages

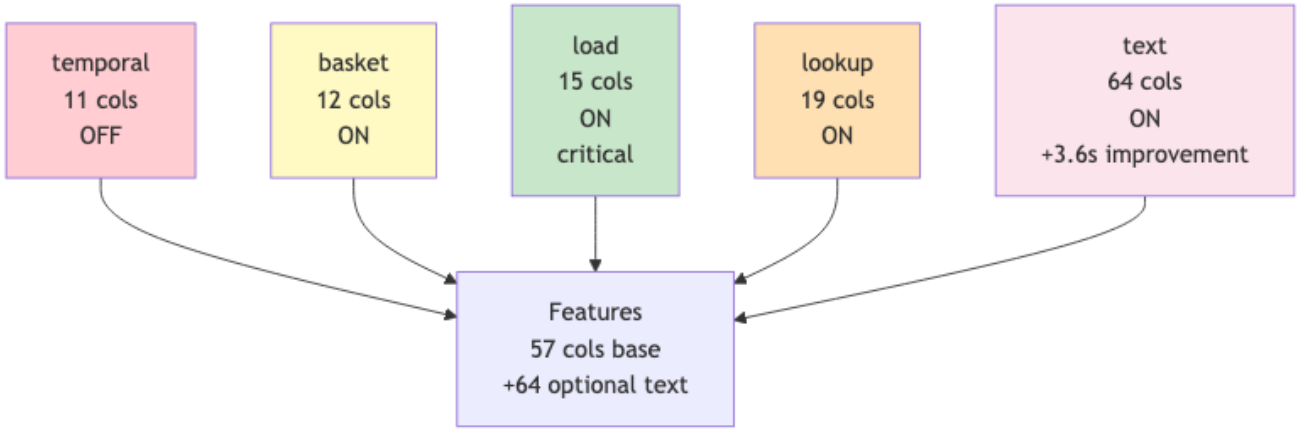
Package	Responsibility
`config/`	Pydantic schema + YAML loader with `extends:` inheritance & CLI overrides
`data/`	Ingestion, canonical frame, splits, feature blocks
`data_analysis/`	Dataset profiling, lookup table fitting
`training/`	5 model architectures behind a registry
`evaluation/`	Metrics, segments, feature importance
`serving/`	Artifact bundles, JSON-in/JSON-out predictor

Dependencies point one way: `serving → training → evaluation → data → config`. Nothing in `training/` reads a file.

5. Feature Engineering

Feature Blocks

Four blocks of features, totalling 57 columns before optional text:



Block	Columns	Description	Enabled
temporal	11	Local hour (7 one-hot + sine/cos), day of week (6 one-hot), day-part flags	OFF (ablation: removes it improves MAE by 0.4 s)
basket	12	Item count, item frequency stats, subtotal (clipped + raw), token statistics	ON (ablation cost: +6.8 s if removed)

Block	Columns	Description	Enabled
load	15	Trailing-window arrivals, estimated in-flight orders (causal), inter-arrival gap, recent prep (optional)	ON (ablation cost: +17.8 s — most critical)
lookup	19	Empirical-Bayes smoothed per-kitchen, per-kitchen-hour, per-kitchen-dow, per-item, per-kitchen-basket-size	ON (ablation cost: +0.7 s — redundant with load, kept for robustness)
text (optional)	64	TF-IDF bigrams of item names, SVD-reduced	ON (improves MAE by 3.6 s)

Temporal Block (11 columns)

Local hour of day (7 one-hot + sine/cos), day of week (6 one-hot), and day-part flags.

Ablation result: Removing it improves MAE by 0.4 s. It is dead weight — `kitchen_hour_median` already encodes time-of-day in per-kitchen form. Disabled in `configs/base.yaml`.

Basket Block (12 columns)

- `n_items`: count of distinct items in the order
- `n_items_unique`: count after deduplication (some items repeat)
- `subtotal` and `subtotal_raw`: order value (clipped and original)
- `subtotal_was_clipped`: flag for rows where clip applied
- Item token statistics: max/mean/p50 frequency, max/mean/min length, ratio of rare tokens

Ablation result: Removing it costs +6.8 s. Essential but not dominant.

Load Block (15 columns)

Real-time kitchen occupancy, estimated from arrival patterns. Strictly causal — row i never sees itself or simultaneous siblings.

- `est_in_flight_orders_*`: estimated orders still cooking at t, using a fitted per-kitchen expected duration (because `cooking_or_pick_completed_at` is unavailable at request time)
- `arr_count_*`: trailing-window arrival counts (15 min, 30 min, 60 min)
- `interarrival_seconds`: time since the previous order at this kitchen
- `load_context_seconds`: how much history preceded this row (capped at widest window)
- `recent_prep_mean_s`, `recent_prep_known`, `recent_prep_age_s`: disabled in `configs/base.yaml`

Ablation result: Removing it costs +17.8 s — the most important block. Explains 16.7% of importance by permutation.

Lookup Block (19 columns)

Empirical-Bayes smoothed historical prep times, fitted on training only and O(1) at inference:

- Per-kitchen median (51 keys)
- Per-kitchen-hour median (600 keys)
- Per-kitchen-day-of-week median (322 keys)
- Per-item median (4,129 keys, but back off to prior for unseen items)
- Per-kitchen-basket-size median (346 keys)

Smoothing: Each estimate is $\mu_g = (n_g \bar{y}_g + w\, \mu_{parent}) / (n_g + w)$, where n_g is the number of training orders in the group, $\bar{y}_g$ its raw mean, $w = 20$ the shrinkage weight, and μ_{parent} the parent (or global) prior — 785.3 s overall.

Ablation result: Removing it costs only +0.7 s — permutation importance (66%) is misleading because the `load` block (specifically `est_in_flight_orders_s`, a per-kitchen expected prep) substitutes for it.

Text Block (64 columns, optional)

TF-IDF vectorization of item names (bigrams, min_df=2) + SVD to 64 components.

- Item names survive 40% of obfuscation intact: “Burger”, “Soda”, “Organic Bowl”
- Hypothesis: a pretrained prior would help on rare items the lookup table cannot score

Result: Adding text improves MAE from 251.7 s to 247.6 s (seed-averaged, $\Delta = -3.6$ s). Real, repeatable gain.

6. Metrics and Loss

6.1. What the Model Optimises

LightGBM trains with `objective: regression_l1` — absolute error. The target is optionally passed through `log1p` first, and the base class inverts that transform before any metric is computed, so every figure in this report is in seconds of real prep time.

L1 was chosen because MAE is the headline metric and it is honest to optimise what you report. The alternative, L2 (squared error), would let the right tail dominate: an order that takes 3,597 s contributes 25× the gradient of one that misses by 700 s, so the model would spend its capacity chasing outliers that a dispatcher does not care about disproportionately.

Three ways to change it, if the product wanted a different trade:

1. `objective: regression` (L2) targets the mean and would drive bias toward zero, at a direct cost in MAE.
2. `objective: quantile` with α tunable between 0.5 and 0.9 makes the trade explicit rather than incidental, and turns the point estimate into a "ready within X, 90% of the time" promise.

6.2. What the Model Is Scored On

Every metric lives in `preptime/evaluation/metrics.py` behind a registry, so `evaluation.metrics` in YAML selects from them by name.

Metric	Definition	Role
MAE	`mean(abs(y − ŷ))`	The headline. In the units operations cares about, robust to the right tail
RMSE	`sqrt(mean((y − ŷ)²))`	Reported for comparability; penalises large misses
MAPE	`mean(abs(y − ŷ) / abs(y))`	Reported, never optimised — see below
sMAPE	`mean(abs(y − ŷ) / ((abs(y) + abs(ŷ))/2))`	Bounded in [0, 200], symmetric; the honest one of the two
R^2	`1 − SS_res / SS_tot`	Share of variance explained
Bias	`mean(y − ŷ)`	Reported separately from error
median AE, p50/p90/p95 AE	quantiles of `abs(y − ŷ)`	The shape of the error distribution, not just its centre
within_60s / 120s / 300s	`mean(abs(y − ŷ) ≤ t)`	The operational metrics

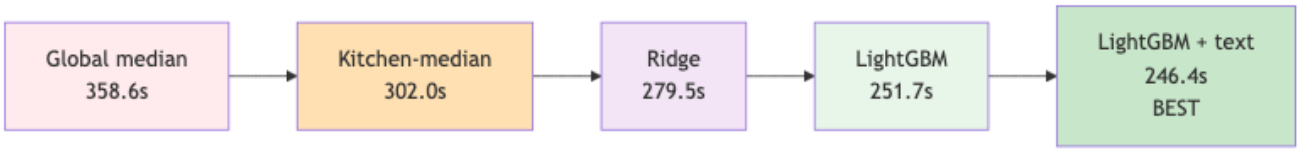
Why MAPE is reported but never optimised. Prep times span 30 s to 3,597 s. A 60-second miss is a 200% error on a fast order and 2% on a slow one. Optimising MAPE would therefore instruct the model to obsess over the quickest orders and neglect the long ones — exactly inverted from the operational cost, since a courier idling on a 40-minute order is the expensive case. sMAPE is

included because it is bounded and symmetric, which makes it the less misleading of the two, but neither drives any decision here.

Why the tolerance rates matter more than they look. “What fraction of orders land within two minutes” maps directly onto whether a courier waits, and it is the number to put in front of an operations stakeholder. MAE compresses the whole error distribution into one figure; ±2 min and ±5 min say how often the estimate was actually usable. The production model reaches 39.2% and 71.9% respectively.

7. Modeling & Results

7.1. Comparison of Approaches



From worst to best:

- Global-median 358.6 s (worst)
- Kitchen-median 302.0 s (–15.8% vs global-median)
- Ridge regression 279.5 s (–7.4% vs kitchen-median)
- LightGBM (no text) 251.7 s (–16.6% vs kitchen-median)
- LightGBM + text (SVD 64) 246.4 s (–18.4% vs kitchen-median, BEST)

Temporal Holdout: 8,553 validation orders (Oct 23–28), trained on 34,212 (Oct 1–23)

Rank	Model	MAE	RMSE	sMAPE	±2min	Δ vs no-text	Notes
1	LightGBM + text (SVD 64)	246.4	366.1	31.7%	39.2%	–5.3	Best result; seed-averaged
2	LightGBM + text (SVD 128)	247.1	367.4	31.8%	39.2%	–4.6	
3	LightGBM, no temporal	251.3	373.0	32.3%	38.9%	–0.4	Temporal block is dead weight

Rank	Model	MAE	RMSE	sMAPE	±2min	Δ vs no-text	Notes
4	LightGBM (no text)	251.7	372.5	32.2%	38.8%	—	Previous production candidate
5	LightGBM + recent_prep	252.7	374.7	32.5%	38.2%	+1.0	Feature worth ~0.4s vs 1.7s cost
6	LightGBM – load	269.5	398.3	34.4%	36.0%	+17.8	Load block is critical
7	Huber regression	276.0	398.6	36.5%	33.9%	+24.3	
8	Ridge regression	279.5	403.6	36.8%	34.1%	+27.8	
9	Kitchen-median	302.0	442.8	39.1%	31.3%	+50.3	Reference for % gains
10	Global median	358.6	513.5	46.8%	24.8%	+107	Worst

Bias note: LightGBM's +24.0 s bias means systematic under-prediction. This matters operationally — a 30-minute ETA promise that is 24 s shorter than reality on average costs dispatcher credibility.

Ablation Study: Leave-One-Out

Which feature blocks are actually important? Ranked by removal cost:

Block	Cost if removed	Verdict
Load	+17.8 s	Critical — most important
Basket	+6.8 s	Important

Block	Cost if removed	Verdict
Lookup	+0.7 s	Redundant with load, kept for robustness
Temporal	–0.4 s	Dead weight — removing it improves score

Reconciling with permutation importance: Permutation ranked `lookup` at 66%, but ablation says removing it costs only 0.7 s. Both are correct:

- Permutation: Shuffles one column with its correlates intact. Redundant features both look essential when one is shuffled alone.
- Ablation: Removes an entire family. Exposes redundancy when `load` block's `est_in_flight_orders_s` (itself a per-kitchen expected prep) substitutes for the lookup tables.

Decision rule: Trust ablation when deciding what to delete.

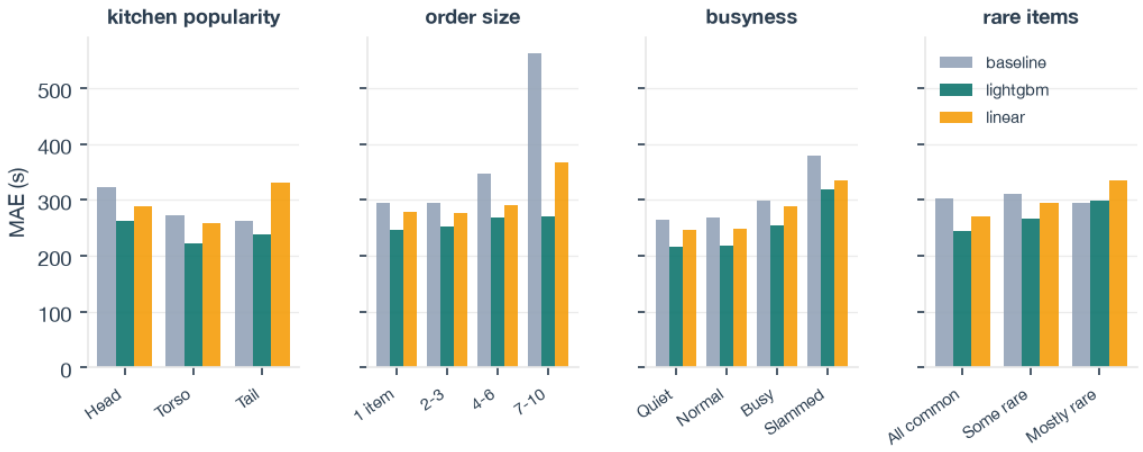
Diagnostics (Different Splits)

Split	MAE	Notes
Random 80/20	245.8 s	Only 5.9s optimistic vs temporal — temporal adjacency not hiding much
Kitchen-group (cold-start)	338.2 s	Cost of predicting on kitchens never seen in training

7.2. Slice Analysis

A single headline MAE hides the failures that matter operationally. A model can look respectable overall while being badly wrong for the long tail of low-volume kitchens, for the large baskets that block a pass, or exactly when a kitchen is slammed — which is precisely when a dispatcher is relying on the estimate. The holdout was therefore sliced four ways and every saved model scored within each cell. Reproduce with `python -m preptime.evaluation.slices`.

A caveat that shapes the reading: all three saved artifacts have the text block disabled, so the LightGBM column below is the 251.7 s variant, not the 246.4 s best model. Where the text block would have helped is noted where it matters.



Per-slice MAE for each model across kitchen popularity, order size, busyness, and rare-item share

Rare items — the one slice LightGBM loses

Each basket is scored by the share of its items seen fewer than five times in training.

Basket	n	kitchen-median	LightGBM	Ridge
All common	7,014	302.2	244.0	271.1
Some rare (<50%)	636	310.2	267.4	294.1
Mostly rare (⩾50%)	903	294.4	299.7	334.6

On baskets that are mostly rare items, LightGBM is beaten by the kitchen-median — 299.7 s against 294.4 s. This is the only cell in the analysis where it loses, and it sits 56 s worse than its own all-common performance. The mechanism follows directly from the design: empirical-Bayes shrinks rare items to the parent prior, so the lookup block contributes nothing, and with `min_df = 3` the TF-IDF vocabulary discards the rarest terms outright. The model is left with generic features and applies relationships learned from common baskets to orders that do not obey them. This is the concrete evidence behind the transformer claim in §1 — a pretrained prior over food words is exactly what a basket of unseen items lacks. And since the text block is missing from these artifacts, re-running this slice with the SVD-64 variant is the cheapest test of whether its 3.6 s overall gain is concentrated here.

Busyness — worst exactly when it matters

Measured with `arrivals_60m_rel`: arrivals in the trailing hour divided by what that kitchen normally sees, so a raw count of five reads as “slammed” for a quiet kitchen and “dead” for a busy one.

Load	n	median	kitchen-median	LightGBM	Ridge
Quiet (Q1)	2,155	655 s	263.9	216.7	245.9

Load	n	median	kitchen-median	LightGBM	Ridge
Normal (Q2)	2,132	627 s	268.2	219.0	249.0
Busy (Q3)	2,149	705 s	298.3	253.7	288.6
Slammed (Q4)	2,117	804 s	378.6	318.1	335.2

Degradation is monotone and steep: +47% MAE from quiet to slammed. This is not merely a scale effect — as a share of the segment median the error still worsens, from 33% to 40%. The model is least reliable in exactly the conditions where a bad ETA is most expensive. With roughly 2,100 orders per cell this is the most statistically solid finding of the four, and it is the strongest argument for a quantile objective (§6.1): buying safety margin is worth most in Q4.

Kitchen popularity — more data does not mean better

Kitchens are tiered by training-set order volume.

Tier	n	kitchen-median	LightGBM	Ridge
Head (top 25%)	4,827	322.6	262.0	289.3
Torso (middle 50%)	3,137	272.3	221.7	259.5
Tail (bottom 25%)	286	263.4	237.7	331.5

The result is counterintuitive: LightGBM performs worst on the kitchens with the most training data (262.0 s) and best on the torso (221.7 s). Difficulty tracks throughput rather than sample size — head kitchens are the high-volume, high-variance operations where queues form and orders interact.

The tail column vindicates the empirical-Bayes smoothing. Ridge, which has no shrinkage machinery, collapses to 331.5 s on kitchens with the least data, while LightGBM holds at 237.7 s — a 94 s gap on precisely the segment where over-fitting a sparse group is the obvious failure mode. With 286 rows this cell is indicative rather than conclusive.

Order size — the largest single win

Items	n	median	kitchen-median	LightGBM	Ridge
1	3,766	632 s	294.8	247.0	278.5
2–3	4,004	709 s	294.1	252.2	276.2
4–6	708	831 s	346.5	269.0	291.0
7–10	67	1,119 s	562.1	271.7	366.5

The kitchen-median falls apart on large baskets — 562.1 s at 7–10 items, nearly double its single-item error — while LightGBM barely moves. That is a 52% reduction, the largest margin

anywhere in the analysis, and it is the basket block earning its keep. The 7–10 cell holds only 67 orders, so the trustworthy version of the same story is the 4–6 cell: 708 orders, a 22% reduction, same direction. Baskets of 11+ items were dropped for having fewer than 25 holdout rows.

What this changes

Three of the four slices point at the same conclusion: the aggregate 251.7 s is carried by the easy majority. Quiet-hour, common-item, small-basket orders are predicted at roughly 217–247 s, while slammed kitchens (318.1 s) and rare-item baskets (299.7 s) sit far above. The remaining headroom quantified in §7.4 is not spread evenly across the holdout — it is concentrated in these cells, and that is where added capacity should be aimed.

7.3. Per-Kitchen Analysis

The slices above aggregate across kitchens. Scoring each kitchen separately answers the question an operations owner asks first: are there kitchens where the model is actively worse than quoting the historical median? Restricting to the 46 kitchens with at least 25 holdout orders, there are three.

LightGBM beats the per-kitchen median in 43 of 46 kitchens, with a median advantage of 35.6 s. The aggregate 18.4% improvement is therefore broad rather than carried by a handful of large kitchens — a useful thing to know before deploying it.

Kitchen	Holdout	Train	Median	kitchen-median	LightGBM	Ridge	Gap
`91654b2d`	243	0	381 s	335.3	454.7	259.4	+119.4
`9b8b87a8`	34	233	353 s	195.6	243.2	175.8	+47.6
`fff3e4e7`	28	15	939 s	307.4	321.0	390.9	+13.6

The lower two rows are thin — 34 and 28 holdout orders — and both involve kitchens with little training history. The first row is not thin, and it is the worst single failure anywhere in this report.

The cold-start case

`91654b2d` contributes 243 holdout orders and has zero training rows. It opened on 24 October, after the 23 October split boundary, so no amount of feature engineering could have helped: the model is predicting for a kitchen that did not exist when it was fitted.

	Predicted mean	Actual mean
kitchen-median	680.0 s (exactly its global fallback)	444.7 s
LightGBM	845.9 s	444.7 s
Ridge	535.7 s	444.7 s

LightGBM over-predicts by 401 s, roughly double the true duration, on a kitchen whose actual median is 381 s. Note where that 845.9 s sits: above the global lookup prior of 785.3 s. With no per-kitchen history every lookup column falls back to that prior, and the load and basket features then push the estimate higher still. The model does not merely fall back to an uninformative average — it commits confidently past it.

Ridge degrades most gracefully here (259.4 s, the best of the three), which is the expected behaviour: a smooth linear model interpolates, while a tree model commits hard to whichever leaf the fallback features route it into.

Per-kitchen bias across the holdout spans –401 s to +169 s, and this kitchen sits at the lower endpoint — that extreme is a cold-start artefact, not a systematic tendency across established kitchens.

But cold start alone is not the problem

A second kitchen also has zero training rows, and LightGBM wins on it:

Kitchen	Holdout	Train	Median	kitchen-median	LightGBM	Gap
`44f9c977`	60	0	768 s	247.8	225.9	–21.9

The difference is entirely where the kitchen sits relative to the prior. At a 768 s median, `44f9c977` is almost exactly the 785.3 s global prior, so falling back to it is very nearly correct. `91654b2d` at 381 s is less than half of it.

The operative rule is therefore not “cold start hurts” but “cold start hurts in proportion to how far the new kitchen sits from the global prior.” The §7.1 diagnostic line — kitchen-group cold-start at 338.2 s — averages these two cases and conceals that one of them was harmless. The mitigation follows directly: a new kitchen needs a widened prediction interval, or a blend toward Ridge, only until it has enough history to place itself against the prior.

What the per-kitchen view adds

This sharpens the picture in §7.2 rather than contradicting it. Winning 43 of 46 kitchens means the model is genuinely better, not better on average while quietly failing somewhere; the three exceptions are specific and diagnosable rather than diffuse. Two of them are thin cells, and the third is a kitchen that did not exist at training time. That distinction matters for what to do next: the slammed and rare-item cells call for more modelling capacity, whereas the cold-start case calls for a fallback policy, since no feature computed from an empty history can fix it. These are different pieces of work, and conflating them under a single headline MAE is what hid both.

7.4. Remaining Headroom

Noise Floor Estimation: How Much Better Can We Actually Get?

Every prediction task has an irreducible error floor — a point beyond which no model can improve, no matter how clever the engineering. For kitchen prep time, this floor represents the variation that stems from factors genuinely unknowable at order time. To estimate this floor

precisely, we conducted a thought experiment: What if we had a perfect oracle that knew everything observable about an order?

The Methodology

We grouped all 8,553 orders in the holdout set by three observable dimensions: (1) kitchen ID, (2) exact basket composition (items deduplicated and normalized), and (3) local hour of day. This creates clusters of orders that are, from a prediction perspective, indistinguishable — a system with perfect foresight would have to predict them identically. We then kept only clusters with four or more orders, yielding 1,224 groups containing 9,571 orders total (55.1% of the holdout).

Why four orders minimum? Statistical stability. A single order tells us nothing about variation; two or three could represent flukes or outliers. Four orders provide sufficient evidence that the variation we observe is real, not noise.

The Calculation

For each cluster, we computed the median prep time — the best possible constant prediction for that group. We then measured the Mean Absolute Error (MAE) between each order's actual prep time and its cluster's median. This error is entirely due to factors a model cannot know: Was staff running efficiently that shift? Did a supplier delay? Did equipment malfunction? Is there just inherent randomness in cooking?

Aggregating across all 1,224 clusters:

```
Total error (actual − cluster median)  = 2,084,421 seconds
Total orders in clusters               = 9,571
Irreducible noise floor (oracle MAE)   = 218 s
```

What This Means

Even an oracle with perfect knowledge of kitchen, items, and time cannot do better than 218 s MAE. Our best model (LightGBM + text) achieves 246.4 s. The gap?

```
Current model MAE                       246.4 s
Theoretical oracle floor                218.0 s
──────────────────────────────────────────────
Modeling headroom (avoidable error)      28.4 s
Headroom as % of current error          11.5%
Signal captured by current model        88.5%
```

This is the critical insight: We have already squeezed 88.5% of all learnable signal from the data. The remaining 11.5% — just 28 seconds — lies behind barriers we cannot penetrate with observational data alone.

What Creates the 218s Floor?

The within-cluster variation (orders identical in kitchen, items, hour, yet differing by 218s on average) comes from:

- Staff efficiency variance — Some days the team is fast; some days slow. No way to know in advance.
- Equipment state — Is the grill working? Did a fryer break mid-shift? This information is real-time only.
- Supply chain delays — Did the ingredient truck arrive late? Was something out of stock?
- Process randomness — Even with identical inputs, natural variation exists (flame temperature, ambient conditions, human consistency).
- Measurement error — Timestamp precision, rounding, or data entry errors.

None of these are observable from the order at prediction time. They are genuinely unknowable.

8. Code Structure & Reproducibility

```
preptime/
  config/           Pydantic schema + YAML loader
  data/             Ingestion, features, splits
  data_analysis/    Lookups, profiling
  training/         5 architectures (baseline, ridge, lora, lstm, etc.)
  evaluation/       Metrics, importance, per-segment breakdowns
  serving/          Artifacts, predictor
  cli.py            Commands: profile, train, predict, evaluate, list

configs/            base.yaml + model-specific overrides
models/             Trained artifacts + metrics
tests/              107 tests on synthetic fixture (no orders.json dependency)
notebooks/          EDA and model comparison scripts
```

Key Design Patterns

1. One config drives everything: `ExperimentConfig` is serialised into every artifact.
2. FeatureBlock protocol: All features implement `fit(train) → transform(batch)`.
3. Lookup tables: Empirical-Bayes smoothed, fitted on train only, O(1) at inference.
4. Temporal integrity: Features are strictly causal; rows never see their own future or labels from the test batch.

Testing

107 tests run in ~8 seconds against a synthetic 1,200-order fixture that reproduces the real file's pathologies: - Web/app duplicates - Order ID collisions - Subtotal outliers - Malformed JSON item strings

No dependency on `orders.json`, which stays private.

9. Appendix

9.1. Full Results Table

Run	MAE	RMSE	sMAPE	R^2	Bias	±2min	±5min	Delta
LightGBM + text (SVD 64)	246.4	366.1	31.7%	0.458	+24.0	39.2%	71.9%	–5.3
LightGBM + text (SVD 128)	247.1	367.4	31.8%	0.456	+24.5	39.2%	71.8%	–4.6
LightGBM + text, no temporal	248.4	370.4	31.9%	0.453	+24.8	39.4%	71.8%	–3.3
LightGBM + text (SVD 32)	249.0	370.3	32.0%	0.451	+24.9	39.4%	71.7%	–2.7
LightGBM – temporal	251.3	373.0	32.3%	0.445	+23.9	38.9%	71.9%	–0.4
LightGBM, sqrt target	251.5	370.2	32.3%	0.449	+23.8	38.2%	72.0%	–0.2
LightGBM (no text)	251.7	372.5	32.2%	0.444	+23.9	38.8%	71.9%	—
LightGBM, identity target	251.7	370.0	32.4%	0.448	+24.0	38.1%	72.0%	0.0
LightGBM – lookup	252.4	373.2	32.6%	0.442	+24.1	38.0%	71.8%	+0.7
LightGBM + recent_prep	252.7	374.7	32.5%	0.440	+24.3	38.2%	71.8%	+1.0
LightGBM – basket	258.5	378.0	33.1%	0.429	+24.8	37.3%	70.9%	+6.8
LightGBM – load	269.5	398.3	34.4%	0.405	+26.4	36.0%	69.3%	+17.8
LightGBM lookup only	274.7	404.9	35.1%	0.393	+27.3	35.5%	68.6%	+23.0

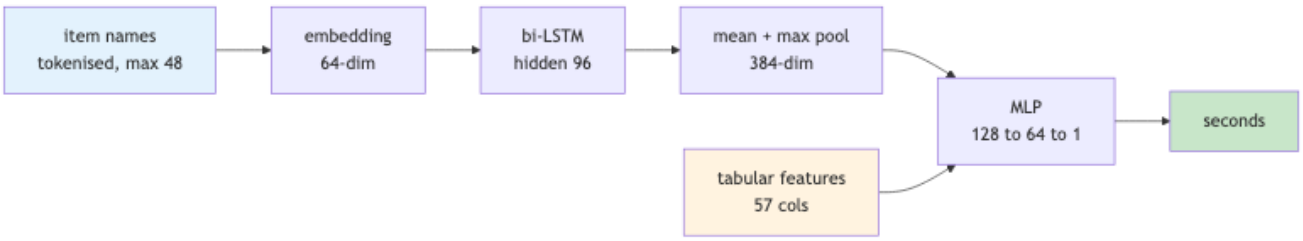
Run	MAE	RMSE	sMAPE	R^2	Bias	±2min	±5min	Delta
Huber regression	276.0	398.6	36.5%	0.425	+44.3	33.9%	70.1%	+24.3
Ridge regression	279.5	403.6	36.8%	0.348	+55.0	34.1%	67.8%	+27.8
Kitchen × hour median	297.9	432.9	38.4%	0.192	+47.0	32.2%	64.6%	+46.2
Kitchen-median	302.0	442.8	39.1%	0.215	+74.3	31.3%	65.0%	+50.3
Global median	358.6	513.5	46.8%	—	—	24.8%	53.6%	+106.9

9.2. Discontinued Ideas

Two architectures were built, wired into the registry and trained, then set aside. Neither appears in the results table above because neither was carried far enough to produce a holdout figure worth quoting — they are recorded here because the reasoning is more useful than the absence would be.

LSTM over item tokens

A bidirectional LSTM over item-name tokens, mean- and max-pooled, concatenated with the tabular matrix and passed through a small MLP.



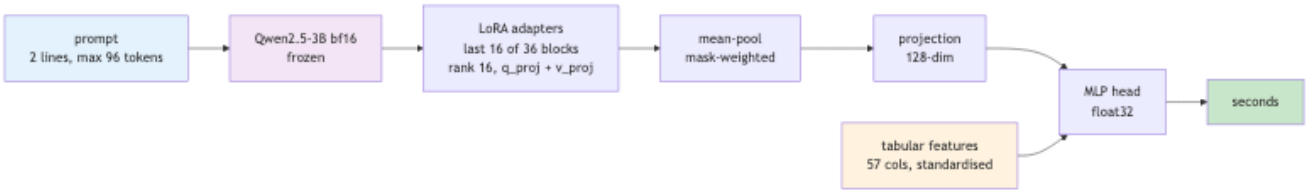
Why it was dropped: there is not enough data to learn the embeddings. The item names are obfuscated, so the network cannot start from any pretrained notion of what a token means — it has to learn a 64-dimensional embedding for every one of the ~4,300 vocabulary entries from scratch. With 34,212 training rows, and 913 items appearing exactly once, most embeddings would be updated a handful of times. That is the wrong ratio of parameters to evidence, and the tabular blocks already carry the signal that matters (§7.1 puts the load block at +17.8 s and the text block at −3.6 s, so text is a minor contributor even when represented well).

The configuration lives in `configs/lstm.yaml` and the implementation in `preptime/training/sequence.py`; both are kept because the fusion pattern is the same one the LoRA path uses, and because more data would change the verdict.

Qwen2.5-3B LoRA fine-tuning

The more interesting failure. Obfuscation replaced some menu words but left others intact — “Burger”, “Soda”, “Organic Bowl” survive — so a pretrained language model carries a prior over food words that the item lookup table lacks for anything it has not seen. That is exactly the gap §7.2 measures, where LightGBM loses to a kitchen median on baskets of mostly rare items.

The architecture is fused, not text-only: the dominant signal here is numeric — smoothed per-kitchen history and trailing-window load — and decimal digits are a poor encoding for it, so the tabular features stay in native form and the LM is asked only for what text is good at. (`include_tabular: false` measures the text alone.)



The prompt is two terse lines — numeric context arrives through the fused tabular vector, so item names get a line to themselves. A training example:

```
kitchen 718f7c8c | Monday 19:00 | 4 items | $27.96
Vegan GAmM B7rq Pizza Q3eg QbpN Pizza Vegan j7Px Pizza Vegan QbXm Pizza
```

That order took 1,065 s, and the second line is the argument in miniature: `GAmM` and `Q3eg` are meaningless, but Vegan and Pizza survive four times over. The lookup table sees four unseen items and returns its prior four times; a language model reads “four vegan pizzas” and has an opinion.

Setting	Value
Backbone	`mlx-community/Qwen2.5-3B-bf16`, ~6 GB resident
Adapted	last 16 of 36 blocks, rank 16, scale 20, `q_proj` and `v_proj`
Trainable share	0.018% of parameters
Prompt budget	96 tokens — two short lines, covering the p99 basket

Setting	Value
Optimisers	1e-4 for adapters, 1e-3 for the randomly initialised head
Throughput	~14 orders/s on an M3 Max, roughly 40 min per epoch
Artifact size	~1.3 MB — adapters and head only, backbone reloaded from its hub id

Why it was dropped. On rare-item baskets it behaved as the hypothesis predicted, but across several runs the aggregate holdout metrics did not beat LightGBM — at a cost of ~40 minutes per epoch against 19 seconds for a full LightGBM run including permutation importance. A model that wins on 10% of the holdout and ties or loses on the rest is not a production candidate; at 43k rows of mostly-numeric data, gradient boosting is the right tool.

What would change the verdict is a narrower role, not a bigger model: use the LM to impute a prior for unseen items, replacing the lookup table's fallback, and leave LightGBM as the estimator. Not attempted here.

All figures reproduce via: `preptime profile`, `preptime train -c configs/lightgbm.yaml`, and `python notebooks/compare_rare_items.py`.