Certified T0 Feature Stores for the Slow21 AFT Survival Model

Leakage-proof data products, certification guardrails, and feature-block architecture

Technical Thesis (Internal)
Generated on 2026-01-14
Scope: feature-store design, T0 leak-proof certification, and how stores power AFT training and evaluation

Abstract. This document describes a production-style, leak-proof feature engineering system for predicting listing time-to-sale and classifying “Slow21” outcomes (sale duration ≥ 21 days / 504 hours) for iPhone marketplace listings. The system decomposes feature engineering into multiple *T0-certified feature stores*, each defined as a one-row-per-(generation, finn_id, t0) contract, validated by an automated certification suite, and protected by a hard-failing “certification guard” at training and scoring time. We formalize the leakage boundary, document the invariants enforced by certification, and explain how the stores are assembled into a single training matrix (~320–350 features) feeding an Accelerated Failure Time (AFT) survival model with boundary-aware calibration. Finally, we present empirical evidence of predictive power and interpretability, including feature-block SHAP attributions and the incremental signal of “live inventory (stock)” features.

Contents

1. Problem framing and leakage boundary

We model the sale process of marketplace listings as a *time-to-event* problem. Each listing (identified by (generation, finn_id)) has a sequence of observed states (edits, first_seen / last_seen snapshots, and potentially a sold_date). We designate a reference time *t0* (typically the listing's edited_date used for feature extraction) and train a model to predict whether the listing will be "Slow21": the realized sale duration is at least 21 days (504 hours). To preserve causal validity and avoid target leakage, every feature value must be computable using information available at or before t0.

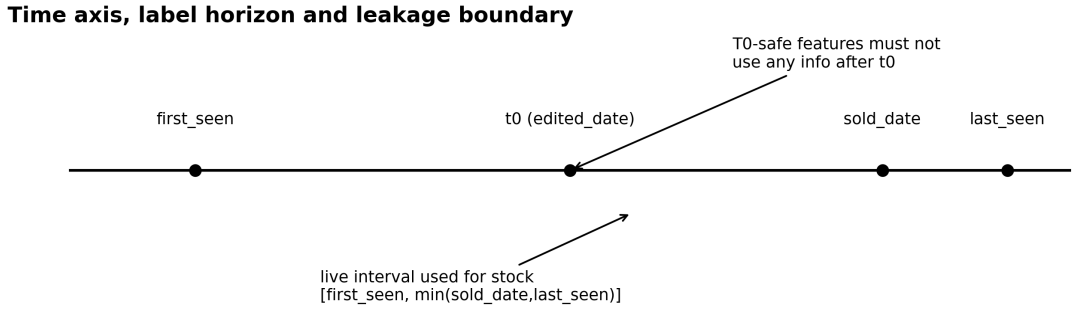

Figure 1. A leakage boundary is defined at t0 (edited_date). Features must not depend on information after t0; labels may depend on events after t0.

In practice, leakage risks arise from (i) using timestamp functions evaluated at query time (e.g., NOW()), (ii) joining to tables where the "as-of" constraint is not enforced (e.g., comp_max_t0 ≥ t0), and (iii) deriving features from post-sale activity (e.g., sold_date, post-sale price changes) when the feature is intended to be known at t0. The system described here mitigates these risks by encoding an explicit T0 contract and enforcing it via automated certification and access control.

$$\log T_i = \mu(x_i) + \sigma\varepsilon_i, \quad \varepsilon \sim \mathcal{D} \in \{\text{Normal}, \text{Logistic}, \text{Extreme}\}$$

Equation 1. AFT modeling assumption for log time-to-sale.

2. T0 data contract for feature stores

A **T0 feature store** is defined as a database view (or materialized view) that produces **exactly one row** per key (generation, finn_id, t0), where t0 is the reference timestamp for leakage control. The store is expected to satisfy a strict contract:

• **Uniqueness:** (generation, finn_id, t0) is unique, with no duplicate rows.
• **No post-t0 lookahead:** any join to time-indexed data must constrain source_time ≤ t0.
• **No query-time functions:** the store definition must not contain NOW(), CURRENT_DATE, CLOCK_TIMESTAMP, etc.
• **Timestamp hygiene:** the store must not emit arbitrary timestamp columns other than t0 (or explicitly permitted snapshot dates).
• **Determinism:** two evaluations of the store at the same database state must return the same results.

$$\forall t_0 : \ \phi(t_0) \in \mathcal{F}_{t_0} \ \text{ and } \ \phi(t_0) \text{ uses no future info } (\mathcal{F}_{t > t_0})$$

Equation 2. Formal leakage constraint: feature values at t0 must be measurable with respect to information available up to t0.

The contract enables a compositional design: each store can be certified independently, and training datasets are built by joining multiple certified stores on the shared key. This decomposition is critical for scalability (multiple teams contributing features), for safety (each store has a narrow blast radius), and for interpretability (features are grouped into meaningful blocks).

3. Certified feature stores and feature blocks

The pipeline is decomposed into multiple stores, each responsible for a coherent family of features and certified against the T0 contract. At training time, stores are merged into a single wide table keyed by (generation, finn_id, t0). The design supports: (i) independent refresh and certification, (ii) strict leak-proofness guarantees, and (iii) block-level interpretability and governance.

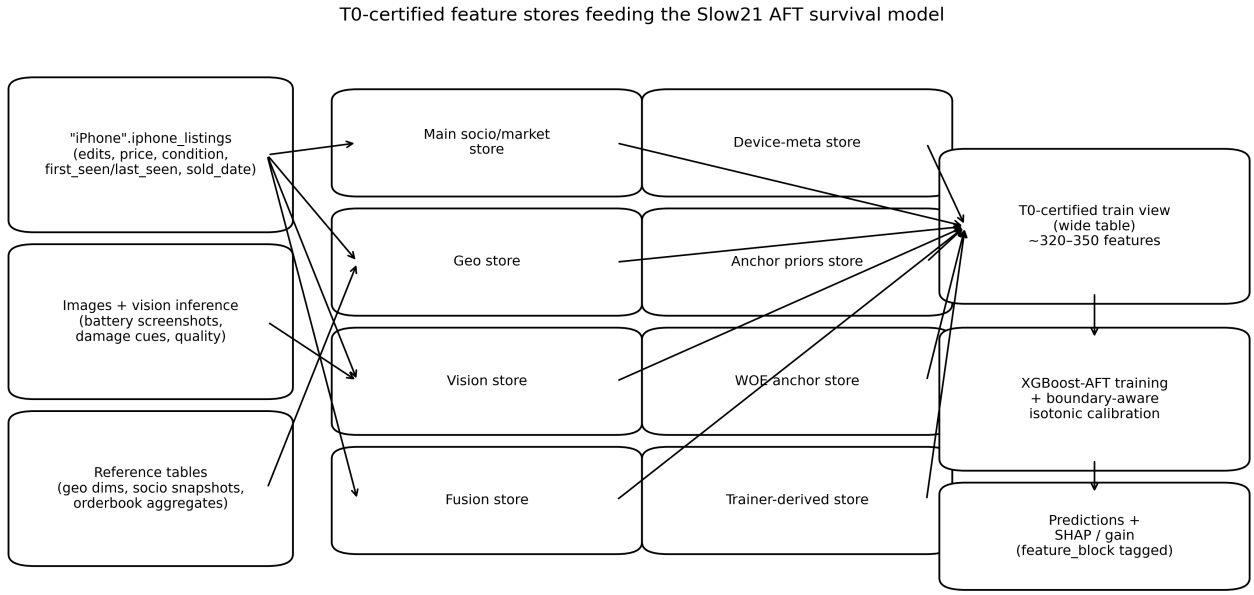

Figure 2. Feature-store DAG. Each store is certified independently; the training matrix is formed by joining on the shared key.

3.1 Store inventory

Table 1 summarizes the primary feature blocks, their entrypoints, and approximate feature counts as observed in a representative model run.

Feature block	Entrypoint / view	Keyed by	Approx. n
main_feature_store	ml.socio_market_feature_store_train_v	listing t0 (edited_date)	7
socio_economic_store	ml.socio_market_feature_store_train_v (socio cols)	kommune/postal snapshots ≤ t0	8
geo_store	ml.geo_dim_super_metro_v4_t0_train_v	finn_id → region/super_metro @ t0	71
vision_store	ml.iphone_image_features_unified_v1_train_v	image-derived features @ t0	41
fusion_store	ml.v_damage_fusion_features_v2_scored_train_v	text×image fused signals @ t0	55
device_meta_store	ml.iphone_device_meta_encoded_v2_t0_train_v	device priors + GMC stats @ t0	56
orderbook_store	ml.tom_ob_features_v1_mv	orderbook aggregates @ t0	9
ai_enrichment_store	ml.tom_features_v*_enriched_ai_*	LLM-extracted fields @ t0	25
anchor_priors_store	ml.anchor_priors_store_t0_v1_v	price-to-value anchors @ t0	12
anchor_woe_store	ml.woe_anchor_store_t0_v1_v	WOE encoding of anchor signals	2
trainer_derived_store	ml.trainer_derived_feature_store_t0_v1_v	derived counts / missing flags @ t0	8
stock_store (optional)	Python-only (no DB store)	live stock counts at t0	3

Table 1. Store inventory (approximate feature counts).

3.2 Block semantics

Feature blocks are not only an organizational tool; they become first-class metadata used for interpretability, drift monitoring, and change management. During training, every feature is labeled with a *feature_block* tag (e.g., “vision_store”, “anchor_priors_store”), and downstream artifacts (feature importance tables, SHAP summaries) preserve the mapping.

4. Certification program and drift tolerance

The certification system converts “we think this is leak-proof” into a testable, enforceable guarantee. It consists of: (1) a static analysis of view definitions (closure), (2) runtime invariants validated on data, (3) baseline recording (viewdefs + dataset hashes), (4) drift detection against baselines, and (5) a hard-failing guard that prevents training/scoring on stale or drifting views.

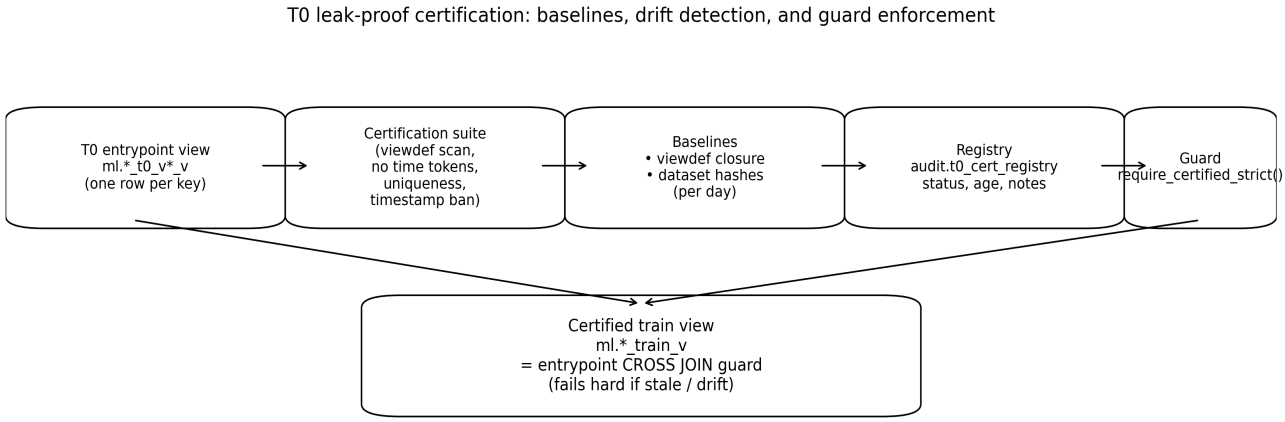

Figure 3. Certification workflow: baselines are recorded; drift is detected; the guard enforces freshness at query time.

4.1 Core invariants enforced by certification

• **View definition closure:** resolve referenced views/matviews recursively; certify the entire dependency graph.
• **Time-token scan:** reject NOW(), CURRENT_DATE, CLOCK_TIMESTAMP(), random(), and other non-deterministic tokens.
• **Key uniqueness:** reject duplicates on (generation, finn_id, t0).
• **Timestamp ban:** reject output columns that are timestamp-like (other than t0 and explicitly whitelisted snapshot dates).
• **As-of constraints:** enforce that any competitor/aggregate “max_t0” used for features is strictly less than t0 (no lookahead).

4.2 Drift-tolerant certification in development

In development environments, reference tables and feature logic evolve rapidly. The drift-tolerant process addresses this by re-baselining hashes for a recent window (e.g., last N days) while keeping the same structural invariants. This makes certification robust to benign changes (e.g., refreshed reference snapshots) while still detecting dangerous changes (e.g., introduction of NOW()).

```
# Pseudocode: drift-tolerant certification (dev)
for view in entrypoint_views:
    closure = resolve_dependency_graph(view)
    assert_no_time_tokens(closure.viewdefs)
    assert_unique_key(view, key=(generation,finn_id,t0))
    assert_no_disallowed_timestamps(view.columns)
    baseline_viewdefs_if_missing(closure)
    baseline_hashes_last_N_days(view, N=7)
    compare_hashes_to_baseline(view)   # drift detection
    write_registry_row(view, status="pass", certified_at=now())
```

5. Anchor systems: priors and WOE encoding

A defining characteristic of the Slow21 model is its heavy reliance on **price-to-value (PTV) anchors**. Anchors provide a robust “market reference” for a listing at t0: they summarize what similar items have sold for and how quickly they tend to sell, allowing the model to reason about overpricing and liquidity. Empirically, anchor features are the strongest predictors (often >50% of total gain importance).

5.1 Anchor priors store

The anchor priors store produces multiple anchor variants with different bias-variance tradeoffs, including a strict anchor (high precision, strict similarity constraints) and a smart anchor (broader coverage with principled backoff). The strict anchor is designed to avoid mixing storage tiers and to preserve generation boundaries; the smart anchor provides fallback support when strict anchors are sparse.

5.2 WOE anchor store

The WOE anchor store converts anchor-derived continuous variables into supervised, monotonic encodings using Weight-of-Evidence (WOE). WOE encoding can stabilize downstream learning, especially under distribution shift, by mapping raw anchor deviations into a log-odds scale relative to the observed Slow21 base rate. To prevent leakage, WOE encodings are computed using out-of-fold (OOF) scoring on the training SOLD subset.

$$\mathrm{WOE}_b = \log \frac{p_b + \varepsilon}{1 - p_b + \varepsilon} - \log \frac{p_0}{1 - p_0}$$

Equation 3. A canonical WOE transform for a bin b relative to base rate p0 (ε is a small smoothing constant).

In the overall training script, the WOE anchor artifacts are persisted (versioned) and then re-applied to both the training and evaluation slices, ensuring that the encoding is stable and reproducible across retrains.

6. Model training: XGBoost AFT with Slow21 gating

The core predictive model is an Accelerated Failure Time (AFT) survival model trained on listing durations. The AFT formulation models the log time-to-sale as a function of features, with support for right-censoring. The trained survival model is then converted into a Slow21 classifier by evaluating the probability that the sale time exceeds 504 hours, followed by boundary-aware calibration.

$$\mathcal{L}_i = \left(\frac{1}{\sigma t_i} f_{\mathcal{D}}(z_i)\right)^{\delta_i} (S_{\mathcal{D}}(z_i))^{1-\delta_i}, \quad z_i = \frac{\ln t_i - \mu(x_i)}{\sigma}$$

Equation 4. Likelihood contribution for a sold (δ=1) or censored (δ=0) observation under an AFT model.

6.1 Handling censoring and the Slow21 label

Listings that have not sold by the time of data extraction contribute as right-censored observations. The pipeline also enforces a minimum censoring horizon (e.g., ≥21 days) to ensure that “not sold yet” is informative for Slow21 rather than a short observation window artifact.

6.2 Boundary-aware calibration

Raw survival probabilities can be miscalibrated near operational decision boundaries (e.g., around the 21-day threshold). The training pipeline therefore applies a boundary-aware isotonic calibration on a recent SOLD slice, increasing resolution and robustness near the threshold.

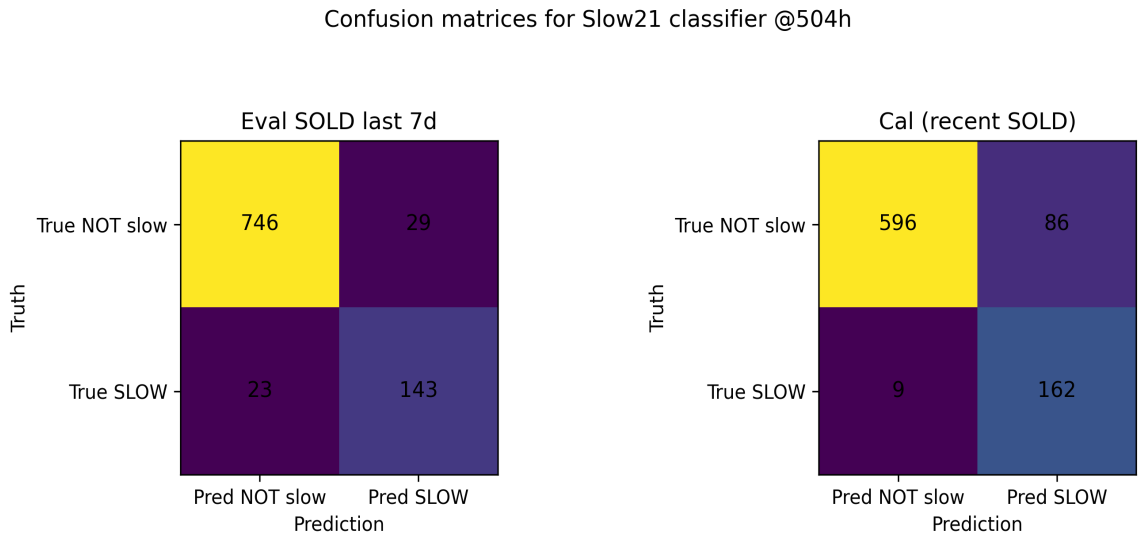

Figure 4. Confusion matrices for the Slow21 classifier on evaluation and calibration slices (example run).

6.3 Example performance snapshot

Slice	Precision	Recall	F1	TP	FP	FN	TN
Eval SOLD last 7d	0.8314	0.8614	0.8462	143	29	23	746
Cal (recent SOLD)	0.6532	0.9474	0.7733	162	86	9	596

Table 2. Representative evaluation metrics.

7. Interpretability: SHAP and feature-block governance

Because the feature system is block-structured, interpretability can be performed at multiple levels: individual features, groups of features within a store, and store-to-store comparisons. A practical workflow is: (1) examine feature-block SHAP to understand which data products dominate predictive power; then (2) drill down into the top blocks; finally (3) validate that the dominant blocks remain stable under retraining and under drift-tolerant certification.

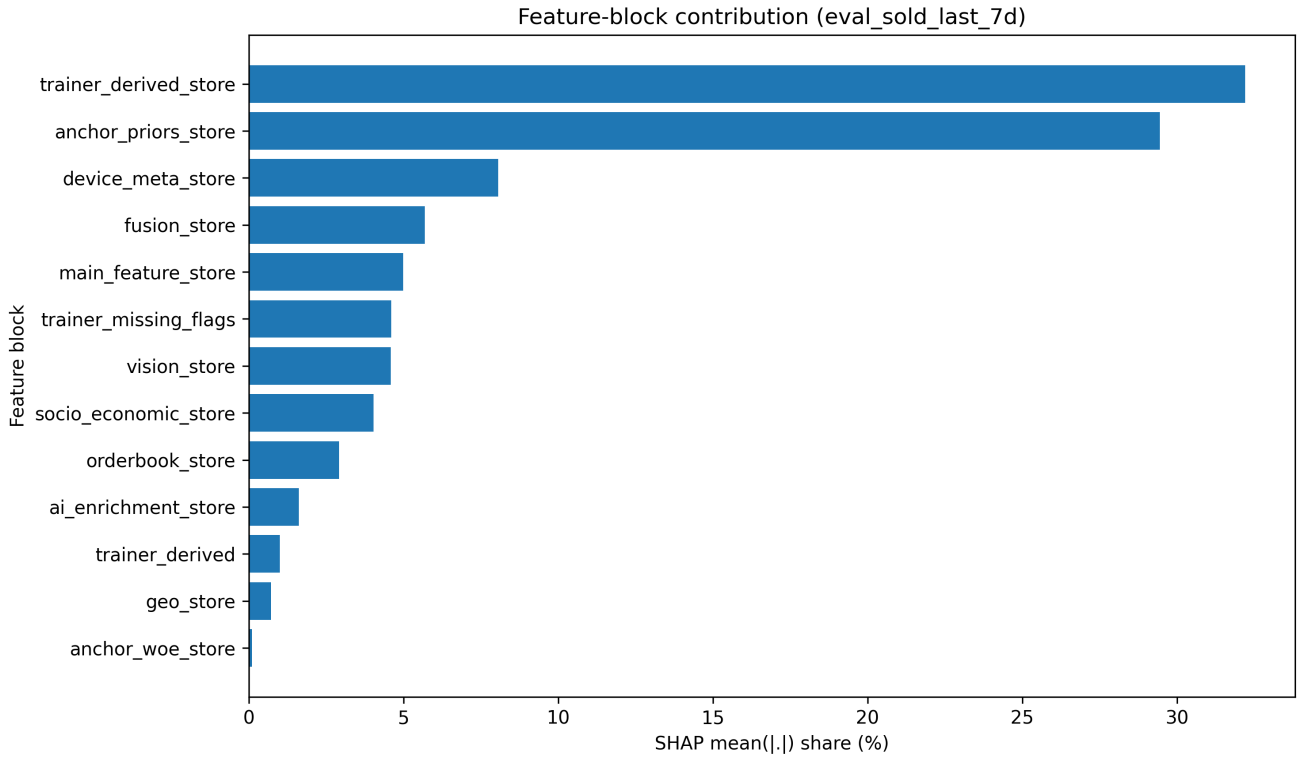

Figure 5. Example feature-block SHAP share (mean absolute). Anchors and trainer-derived features dominate in this run.

7.1 Feature-block summary table

feature_block	n_features	shap_sum	pct_total(%)
trainer_derived_store	8	1.1304	32.2047
anchor_priors_store	12	1.0337	29.4489
device_meta_store	56	0.2828	8.0574
fusion_store	55	0.1999	5.6936
main_feature_store	7	0.1753	4.9940
trainer_missing_flags	22	0.1618	4.6103
vision_store	41	0.1611	4.5898
socio_economic_store	8	0.1416	4.0334
orderbook_store	9	0.1025	2.9193
ai_enrichment_store	25	0.0569	1.6199
trainer_derived	4	0.0352	1.0019

feature_block	n_features	shap_sum	pct_total(%)
geo_store	71	0.0254	0.7241
anchor_woe_store	2	0.0036	0.1027

7.2 Top features by gain (example run)

feature	gain_pct
ptv_anchor_strict_t0	28.153
ptv_anchor_smart	25.452
ptv_sold_30d	4.146
member_since_year__missing	2.349
ai_lqs_textonly_f	0.902
review_count__missing	0.837
aff_decile_in_seg	0.715
allgen_30d_post_count	0.686
image_count__missing	0.671
anchor_n60_t0	0.657
speed_median_hours_ptv	0.622
battery_pct_effective	0.605
seller_rating__missing	0.576
anchor_n30_t0	0.573
generation	0.544

Anchors (ptv_anchor_strict_t0 and ptv_anchor_smart) alone account for more than half of gain importance in this example. This is consistent with a marketplace economics interpretation: pricing relative to comparable sales is the primary driver of sale speed, with secondary contributions from seller quality signals, socio-economic affordability, and content quality (images/text).

8. Stock features: design, leakage analysis, and measured signal

“Stock” is defined as the number of substitute listings a buyer can see at the same time as a given listing. Intuitively, higher stock implies more competition and could increase the probability that a listing becomes Slow21 (takes ≥21 days to sell), especially for overpriced listings. The research workflow constructs three stock features:

• **stock_n_sm4_gen_sbucket**: count of distinct listings live at t0 in the same (super_metro_v4_geo, generation, sbucket).
• **stock_n_sm4_gen**: count of distinct listings live at t0 in the same (super_metro_v4_geo, generation).
• **stock_share_sbucket**: stock_n_sm4_gen_sbucket / NULLIF(stock_n_sm4_gen, 0), a normalized “within-generation” inventory share.

8.1 Leak-proof intent. In a real-time system, these counts can be computed at t0 by querying the marketplace index as-of t0. In offline training, we reconstruct a proxy for “live at t0” using listing life intervals derived from first_seen and an inferred live_end. This reconstruction is a potential source of subtle leakage if last_seen is used from after t0; therefore, stock features are treated as *experimental* until a fully T0-certified implementation is established (e.g., by maintaining an as-of listing snapshot table keyed by scrape time).

8.2 Empirical signal: unconditional deciles

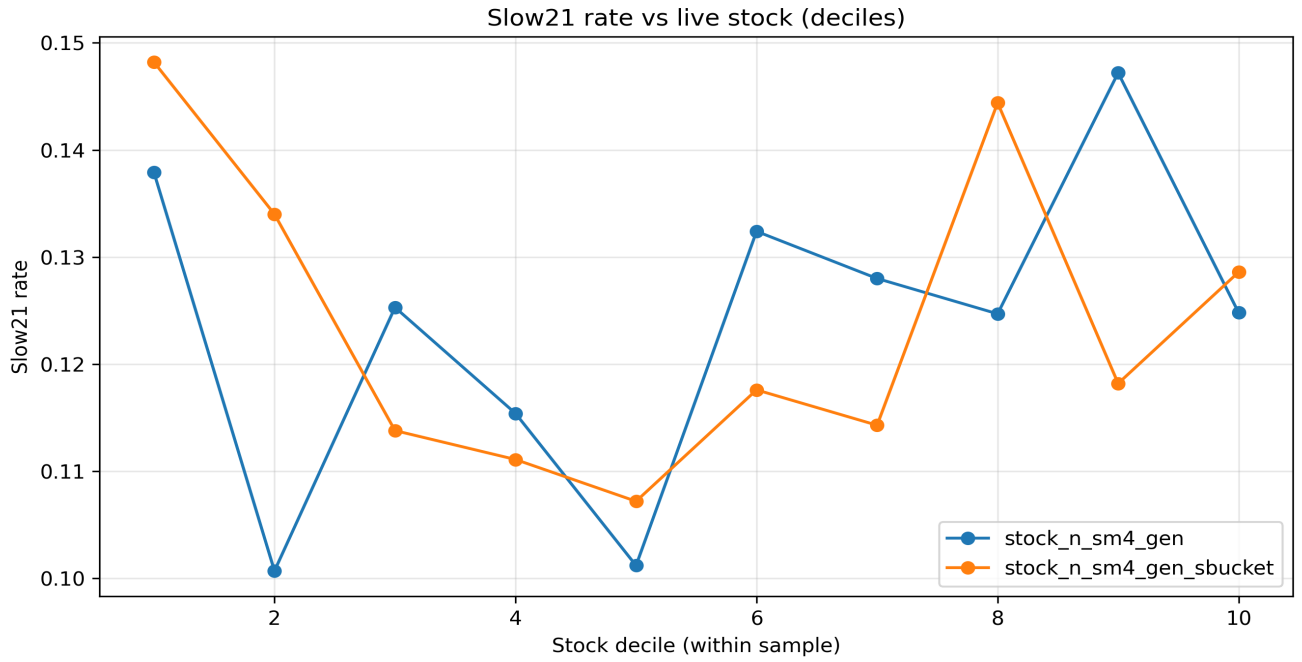

Figure 6. Slow21 rate vs stock deciles. Unconditional trends are present but not strictly monotonic.

8.3 Conditional signal: incremental lift within price-relative strata

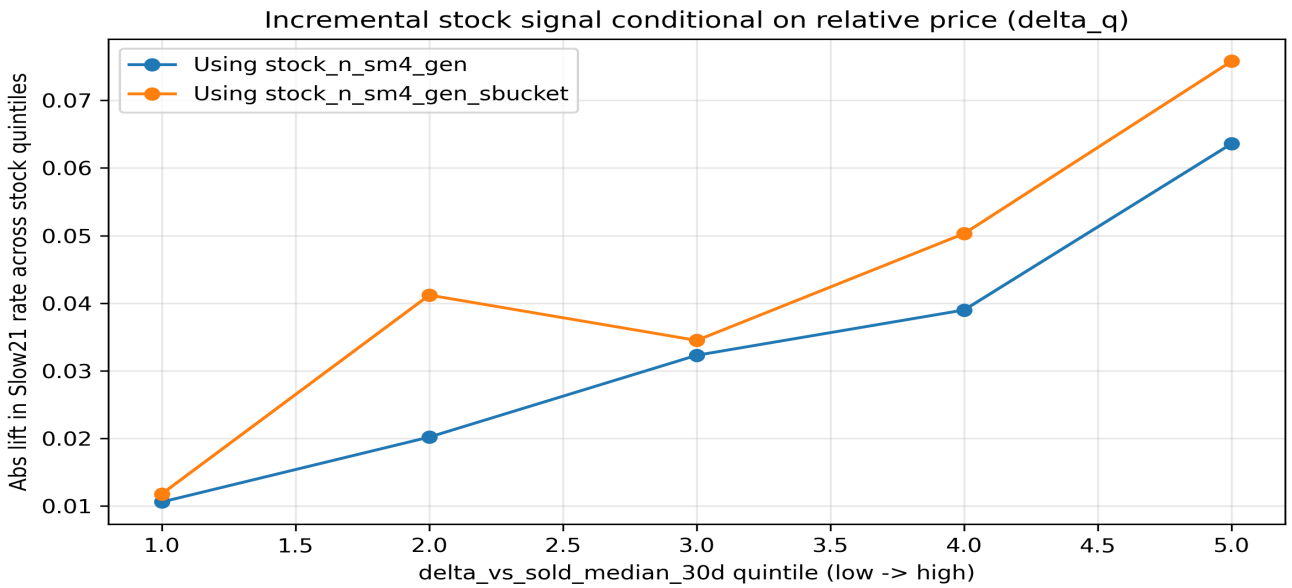

Figure 7. Absolute Slow21 lift (max-min across stock quintiles) increases with relative overpricing (delta_q).

The conditional analysis indicates that stock is most informative when the listing is already in a "hard-to-sell" pricing regime (high delta_vs_sold_median_30d quintiles). In those strata, the absolute Slow21 rate can shift by ~0.05–0.08 (and in some segments >0.15) across stock levels, suggesting non-linear interactions among price positioning, market inflow, and live inventory.

8.4 Information gain (entropy reduction) within (delta_q, flow_q) segments

delta_q	flow_q	n_seg	entropy_before	entropy_after	info_gain_bits
4	1	666	0.52971	0.50455	0.02515
2	4	713	0.33420	0.31685	0.01735
2	1	806	0.22940	0.21773	0.01167
5	1	656	0.78360	0.77269	0.01091
5	2	793	0.86714	0.85761	0.00952
4	4	819	0.65144	0.64281	0.00863
1	2	665	0.25186	0.24421	0.00765
4	2	666	0.60265	0.59529	0.00737
5	3	747	0.89229	0.88619	0.00610
2	5	727	0.32952	0.32462	0.00490

Table 3. Information gain (bits) from stock stratification within segments.

9. Operational runbook: refresh, certification, and debugging

9.1 Refresh and certify. Each T0 store has a refresh procedure (often a materialized view refresh) and a certification step that records baselines and updates the certification registry. Training and scoring must read from *train_v* views that cross join the guard, so that staleness or drift causes a hard error rather than silent leakage.

```
# Typical operational sequence (conceptual)
REFRESH MATERIALIZED VIEW CONCURRENTLY ml.<store>_mv;
CALL audit.certify_t0_view('ml.<entrypoint_t0_view>');

# Training uses *_train_v which includes the guard:
SELECT ... FROM ml.<feature_store>_train_v;
```

9.2 Debugging certification failures

Common failure modes include duplicates on the key, time-token violations introduced during refactors, and implicit lookahead joins (e.g., using a “max_t0” without enforcing max_t0 < t0). The certification suite is intentionally conservative; the recommended remediation pattern is to add an explicit as-of constraint, introduce a snapshot_date key, or split the view into an as-of safe component and a post-hoc analysis component.

9.3 Debugging experimental stock features

Stock features are computed in Python (no new DB objects). When enabled, the training script should emit a line similar to: **[stock] attached: +3 cols (stock_n_sm4_gen, stock_n_sm4_gen_sbucket, stock_share_sbucket)**. If the line does not appear, verify that the CLI flag **--use_stock_features** is passed and that feature_cols.json contains the three names.

10. Conclusion and next experiments

This thesis documented a modular, certified, leak-proof feature engineering system powering a high-performing Slow21 AFT survival model. The key engineering contribution is the **T0 contract + certification guard**, which turns leakage avoidance into an enforceable property rather than a best-effort convention. The key modeling contribution is a calibrated survival-to-classification pipeline that achieves strong Slow21 discrimination on recent SOLD slices.

Recommended next experiments (in priority order):

• **Certified stock store:** introduce an as-of listing snapshot table (scrape_time keyed) to compute stock without using future last_seen, then certify it as a proper T0 store.
• **Interaction modeling:** explicitly model interactions between (delta_vs_sold_median_30d, flow, stock) via monotonic constraints or spline-like transformations to capture non-linear inventory effects.
• **Stability under drift:** track feature-block SHAP shares across weekly retrains and alert on block-level shifts (e.g., anchors dropping, vision dominating unexpectedly).
• **Calibration audits:** evaluate calibration curves across subsegments (generation, super_metro, sbucket) and adjust boundary-aware calibration if bias emerges.

Appendix note. Store-level details, certification SQL, and operational scripts are maintained in the certified packages and documentation artifacts.