Part 1 Combined Report: Q1, Q2, Q3

JP Morgan MLCOE TSRL 2026 Internship

Jin Kim
Zixuan Zhao
Shawn Shin

February 2026

Abstract

This report combines the Part 1 deliverables for Q1 (balance sheet modeling), Q2 (state-space filtering to particle flow), and Q3 (deep context-dependent choice modeling). For each question, we provide a targeted literature review, justify the selected methods, present clear technical answers to the prompt, and document testing plans with results that validate correctness. Only Part 1 requirements are included; Part 2 and bonuses are intentionally omitted.

Contents

1 Q1 Part 1: Balance Sheet Modeling with Accounting Identities

1.1 Introduction

Balance sheet forecasting is a core capability for any lending institution. The challenge lies in the interdependence of financial statement fields: debt drives interest expense, which drives net income, which drives retained earnings, which feeds back into equity. Naive time-series models on individual line items break accounting identities, producing internally inconsistent forecasts that are useless for credit analysis.

This section addresses Part 1 of the assignment: constructing a deterministic balance sheet model that respects accounting identities by design, implementing it in TensorFlow with real financial data, and validating the framework through forward forecasting. We adopt a two-tier architecture in which machine learning models forecast economic *drivers* (revenue growth, margins, working capital ratios), while a deterministic accounting engine constructs internally consistent financial statements from those drivers.

1.2 Literature Review

1.2.1 Accounting Identity Preservation

The fundamental constraint in balance sheet modeling is the accounting identity $A_t = L_t + E_t$, which must hold at every forecast step. Vélez-Pareja (2007) introduced the "No Plugs, No Circularity" framework, which constructs financial projections such that identities are satisfied by the computation order itself, without post-hoc adjustments or "plug" variables (**?**). Vélez-Pareja (2009) extended this to consistent financial planning models suitable for valuation (**?**).

1.2.2 Circularity Resolution

A key difficulty is the circularity between debt and interest expense: interest depends on debt, but debt issuance depends on cash needs, which depend on interest. Mejía-Peláez and Vélez-Pareja (2011) provide an analytical solution to this circularity in the discounted cash flow framework (**?**). For numerical implementation, iterative convergence (3–5 iterations) is the standard practical approach.

1.2.3 Dynamic Simulation Models

Shahnazarian (2004) develops a dynamic microeconometric simulation model for incorporated businesses that links income statement dynamics to balance sheet evolution over time (**?**). This motivates the simulation viewpoint: $y_{t+1} = f(x_t, y_t) + n_t$, where the balance sheet state evolves as a function of exogenous drivers and the current state.

1.2.4 LLM-Based Financial Analysis

Recent work explores LLM-based financial statement analysis. Alonso & Dupouy (2024) demonstrate LLMs as financial analysts (**?**). Farr et al. (2025) examine AI reliability with financial data (**?**). Zhang et al. (2025) develop checking relationship recognition for financial statements using LLMs (**?**). These motivate the Part 2 comparison but inform the driver selection in Part 1.

1.3 Method Selection and Justification

1.3.1 Choice of Methods

We implement a **two-tier architecture** that separates economic forecasting from accounting construction:

Table 1: Selected Methods and Their Roles

Component	Method	Rationale
Driver forecasting	LSTM (64→32 units)	Captures temporal patterns in growth rates
Margin forecasting	XGBoost	Captures cross-sectional ratio relationships
Statement construction	Vélez-Pareja deterministic	Guarantees identity compliance by design
Circularity resolution	Iterative solver	Converges in 3–5 iterations
Data ingestion	Yahoo Finance (`yfinance`)	Free, programmatic, multi-statement

1.3.2 Justification

Why not forecast balance sheet items directly? A direct autoregressive model on line items (e.g., forecasting Total Assets as a function of lagged Total Assets) cannot guarantee $A = L + E$ in the output. Enforcing identity post-hoc via a plug variable (e.g., adjusting cash) introduces artificial distortion. The Vélez-Pareja approach avoids this entirely: the model forecasts *drivers*, and the accounting engine computes line items such that identities hold by construction.

Why ML for drivers rather than simple assumptions? Traditional financial models assume fixed growth rates or trend extrapolation. LSTM and XGBoost can learn nonlinear temporal patterns (e.g., mean-reverting margins, cyclical revenue) from historical data, improving forecast quality while keeping the accounting structure intact.

Why separate LSTM and XGBoost? Revenue growth exhibits strong autocorrelation (suited to LSTM), while margin ratios are better predicted from a cross-section of current-period features (suited to tree-based methods). This heterogeneous ensemble outperforms either model alone.

1.4 Model Formulation

The balance sheet forecast is governed by the following system. Let $x(t)$ denote the driver vector and $y(t)$ denote the balance sheet state.

1.4.1 Income Statement Dynamics

$$\text{Revenue}(t) = \text{Revenue}(t-1) \times (1 + g_{\text{rev}}(t)) \tag{1}$$

$$\text{COGS}(t) = \text{Revenue}(t) \times r_{\text{cogs}}(t) \tag{2}$$

$$\text{OpEx}(t) = \text{Revenue}(t) \times r_{\text{opex}}(t) \tag{3}$$

$$\text{EBIT}(t) = \text{Revenue}(t) - \text{COGS}(t) - \text{OpEx}(t) - \text{Depr}(t) \tag{4}$$

$$\text{Interest}(t) = \text{Debt}(t-1) \times r_{\text{debt}} \tag{5}$$

$$\text{Net Income}(t) = \big(\text{EBIT}(t) - \text{Interest}(t)\big) \times (1 - \tau) \tag{6}$$

1.4.2 Balance Sheet Evolution

$$\text{AR}(t) = \text{Revenue}(t) \times \text{DSO}(t)/365 \tag{7}$$

$$\text{Inventory}(t) = \text{COGS}(t) \times \text{DIO}(t)/365 \tag{8}$$

$$\text{AP}(t) = \text{COGS}(t) \times \text{DPO}(t)/365 \tag{9}$$

$$\text{PP\&E}(t) = \text{PP\&E}(t-1) + \text{CapEx}(t) - \text{Depr}(t) \tag{10}$$

$$\text{RE}(t) = \text{RE}(t-1) + \text{Net Income}(t) - \text{Dividends}(t) \tag{11}$$

1.4.3 Accounting Identity (Enforced)

$$\text{Total Assets}(t) = \text{Total Liabilities}(t) + \text{Total Equity}(t) \tag{12}$$

The circularity between Interest(t) and Debt(t) is resolved by iterating the income statement and balance sheet updates until convergence ($|\Delta\text{Interest}| < 10^{-6}$, typically 3–5 iterations).

1.4.4 Simulation Framework Mapping

The prompt suggests the general form $y_{t+1} = f(x_t, y_t) + n_t$. This maps directly to our architecture:

- $\mathbf{y}(t)$: Balance sheet state (all line items)
- $\mathbf{x}(t)$: Exogenous drivers—g_{rev}, r_{cogs}, r_{opex}, DSO, DIO, DPO, r_{debt}, τ
- $f(\cdot)$: The deterministic accounting model (equations above)
- n_t: Driver forecast uncertainty propagated through the model

ML models forecast $x(t)$; the deterministic engine computes $f(x_t, y_t)$. This separation is the key design choice: ML handles uncertainty in *drivers*, while the accounting model guarantees *structural consistency*.

1.5 Implementation Details

1.5.1 TensorFlow Architecture

The driver forecasting model (`tensorflow_model.py`) uses:

- LSTM (64→32 units) for revenue growth forecasting from sliding windows of historical ratios
- XGBoost for COGS and OpEx margin forecasting from current-period features
- Adam optimizer with MSE loss; exponential learning rate decay

1.5.2 Data Pipeline

Automated ingestion via `data_collection.py` using `yfinance`:

- Collects annual and quarterly balance sheets, income statements, and cash flow statements
- Applied to AAPL: 69 balance sheet periods, 39 income statement periods, 53 cash flow periods
- Validates data quality, imputes missing values, and normalizes column names across providers

1.5.3 Training Methodology

Time-series cross-validation (`train_and_evaluate.py`):

- Expanding-window splits to prevent data leakage (no future data in training set)
- Metrics: MSE, RMSE, MAE, MAPE, R^2, directional accuracy
- Accounting identity validation on every predicted period
- Temporal consistency checks on financial ratio trajectories

1.5.4 ML Enhancements

Beyond the production LSTM+XGBoost pipeline, advanced techniques are implemented in `ml_enhancements.py`:

- Attention mechanisms over input sequences for interpretable feature weighting
- Bidirectional LSTMs with residual/skip connections for gradient stability
- Batch normalization and hyperparameter optimization framework

1.5.5 Earnings Forecasting

Earnings are a natural byproduct of the deterministic model (`earnings_forecast.py`):

$$\text{Net Income}(t) = \Delta\text{RE}(t) + \text{Dividends}(t) \tag{13}$$

Derived metrics (ROE, ROA, earnings growth, earnings volatility) are computed from the forecast output. Earnings are calculated *before* the balance sheet update at each step, ensuring consistency.

1.6 Testing Plan

1.6.1 Correctness Tests

1. **Identity Compliance**: Verify $|A_t - (L_t + E_t)| < 10^{-6}$ for each forecast period.
2. **Circularity Convergence**: Confirm the iterative solver converges within 5 iterations and the residual is below tolerance.
3. **Retained Earnings Consistency**: Check that $\text{RE}(t) - \text{RE}(t-1) = \text{Net Income}(t) - \text{Dividends}(t)$ at each step.

1.6.2 Performance Tests

1. **Time-Series Validation**: Train on early periods, test on later periods; evaluate MAE/MAPE on key line items.
2. **Driver Forecast Accuracy**: Evaluate LSTM revenue growth and XGBoost margin predictions against held-out actuals.
3. **Sensitivity Analysis**: Stress-test drivers ($\pm 2\sigma$ shocks to revenue growth) and verify the balance sheet remains consistent.

1.7 Results

1.7.1 AAPL 8-Period Forecast

Table 2: AAPL Forward Forecast: Key Metrics by Period

Period	Revenue	Net Income	D/E	ROA
1	\$1.04B	\$99.8M	0.53x	10.27%
4	\$1.78B	\$198.4M	0.38x	14.83%
8	\$3.32B	\$406.8M	0.21x	20.71%

Accounting Identity: Zero violations across all 8 periods ($|A - (L + E)| < 10^{-6}$).

Financial Trajectories: D/E decreases from 0.53x to 0.21x (deleveraging), ROA improves from 10.27% to 20.71%, revenue growth accelerates from 3.69% to 49.63%.

Earnings Forecast: Net income grows from \$99.8M (Period 1) to \$406.8M (Period 8), consistent with retained earnings changes and dividend assumptions.

1.8 Discussion

1.8.1 Strengths

- **Identity compliance by construction**: Unlike post-hoc adjustment methods, the Vélez-Pareja approach never produces an imbalanced balance sheet, making it suitable for regulatory and audit contexts.
- **Interpretability**: Every line item is traceable to a specific driver forecast and accounting equation; the model is fully auditable.
- **Modularity**: The two-tier separation allows upgrading the ML component (e.g., replacing LSTM with a transformer) without changing the accounting engine.

1.8.2 Limitations

1. **Driver forecast quality**: The entire forecast quality depends on the accuracy of driver predictions. Poor revenue growth estimates propagate through every line item.
2. **Fixed accounting structure**: The model assumes a fixed set of line items and relationships. Companies with unusual balance sheet structures (e.g., financial institutions with trading assets) require customization.
3. **Limited sample size**: Yahoo Finance provides at most $\sim$20 annual periods per company, constraining ML model capacity. Quarterly data mitigates this partially.

1.8.3 Practical Recommendations

For banking applications:

1. Start with the deterministic model using analyst-estimated drivers for high-confidence forecasts.
2. Layer ML-based driver forecasting for automated screening of large loan portfolios.
3. Use sensitivity analysis (driver stress tests) to quantify forecast uncertainty for credit committee review.

1.9 Conclusion

We have implemented a two-tier balance sheet forecasting system that combines ML-based driver forecasting with a deterministic accounting engine. The Vélez-Pareja framework guarantees that the fundamental identity $A = L + E$ holds at every forecast step by construction, producing internally consistent financial projections suitable for lending decisions. The approach is validated on AAPL with zero identity violations across an 8-period forward forecast. The modular architecture allows independent improvement of the ML and accounting components, making it practical for production deployment in a bank's credit analysis workflow.

2 Q2 Part 1: From Classical Filters to Particle Flow

2.1 Introduction

State-space models (SSMs) provide a powerful probabilistic framework for modeling sequential data with latent dynamics. In financial applications, SSMs are used for volatility estimation, risk management, and anomaly detection. The filtering problem—estimating the hidden state given noisy observations—is central to these applications.

For linear-Gaussian systems, the Kalman Filter provides optimal estimates. However, real-world systems exhibit nonlinear and non-Gaussian behavior, necessitating more sophisticated methods. While particle filters offer flexibility through Monte Carlo approximation, they suffer from weight degeneracy in high dimensions. Particle flow filters have emerged as a promising alternative, transporting particles from prior to posterior without weight-based resampling.

This report addresses Part 1 of the assignment:

1. **Warm-up**: Linear-Gaussian SSM with Kalman Filter
2. **Nonlinear Filters**: EKF, UKF, and Particle Filter comparison
3. **Particle Flow Filters**: EDH, LEDH, PF-PF, and Kernel methods

2.2 Literature Review

2.2.1 Classical Filtering Methods

Kalman Filter The Kalman Filter (**?**) provides the optimal linear minimum mean-square error (LMMSE) estimator for linear-Gaussian systems:

$$x_t = Fx_{t-1} + w_t, \quad w_t \sim \mathcal{N}(0, Q) \tag{14}$$

$$y_t = Hx_t + v_t, \quad v_t \sim \mathcal{N}(0, R) \tag{15}$$

The standard covariance update $P = (I - KH)P^-$ can become numerically unstable. The **Joseph form** provides improved stability:

$$P = (I - KH)P^-(I - KH)^T + KRK^T \tag{16}$$

Extended and Unscented Kalman Filters For nonlinear systems, the **Extended Kalman Filter (EKF)** linearizes the dynamics around the current estimate using Jacobians. However, linearization introduces errors, particularly for strongly nonlinear functions.

The **Unscented Kalman Filter (UKF)** (**?**) avoids explicit Jacobian computation by propagating sigma points through the nonlinear functions, capturing mean and covariance to second order.

2.2.2 Particle Filters

The **Bootstrap Particle Filter** (**?**) approximates the posterior using weighted samples:

$$p(x_t|y_{1:t}) \approx \sum_{i=1}^{N} w_t^{(i)} \delta(x_t - x_t^{(i)}) \tag{17}$$

While flexible, particle filters suffer from **weight degeneracy**: in high dimensions, most particles receive negligible weight, leading to sample impoverishment.

2.2.3 Particle Flow Filters

Exact Daum-Huang (EDH) Flow **?** introduced particle flow as a deterministic transport from prior to posterior. The EDH flow is:

$$\frac{dx}{d\lambda} = PH^T R^{-1}(y - h(x)) \tag{18}$$

where $\lambda \in [0, 1]$ is pseudo-time, and particles are migrated without changing weights.

Local EDH (LEDH) Flow **?** addressed particle degeneracy by using **local covariance** computed from neighboring particles, improving diversity in regions of low probability.

Invertible Particle Flow (PF-PF) **?** combined particle flow with importance sampling. By tracking the flow Jacobian, proper importance weights can be computed:

$$\log w = \log p(y|x_T) - \log \left| \det \frac{\partial x_T}{\partial x_0} \right| \tag{19}$$

where x_T is the particle after flow and x_0 is the prior particle.

Kernel-Embedded Particle Flow **?** embedded particle flow in a Reproducing Kernel Hilbert Space (RKHS):

$$\frac{dx_i}{d\lambda} = \frac{1}{N} \sum_{j=1}^{N} K(x_i, x_j) \nabla_{x_j} \log p(y|x_j) \tag{20}$$

A critical finding is that **scalar kernels fail in high dimensions with sparse observations** due to marginal collapse. **?** proposed **matrix-valued kernels**:

$$K(x_i, x_j) = \text{diag}(K^{(1)}(x_i^{(1)}, x_j^{(1)}), \ldots, K^{(d)}(x_i^{(d)}, x_j^{(d)})) \tag{21}$$

which allow dimension-specific repelling forces, preventing collapse.

2.3 Method Selection and Justification

2.3.1 Choice of Methods

We implement the following hierarchy of methods:

Table 3: Selected Methods and Their Characteristics

Method	Complexity	Best For
Kalman Filter	$O(n^3)$	Linear-Gaussian systems
EKF	$O(n^3)$	Mild nonlinearity
UKF	$O(n^3)$	Moderate nonlinearity
Bootstrap PF	$O(Nn)$	Low dimensions ($\leq$ 5D)
EDH Flow	$O(Nn^2)$	Medium dimensions (5–15D)
LEDH Flow	$O(N^2n)$	Medium-high dimensions (15–30D)
Kernel-Scalar	$O(N^2n)$	High dimensions (30D+)
Kernel-Matrix	$O(N^2n)$	Very high dimensions (50D+)
PF-PF	$O(Nn^2)$	Importance weight correction

2.3.2 Justification

EDH over standard PF: Particle flow avoids weight degeneracy by transporting particles deterministically. In dimensions > 10, standard PF requires exponentially many particles.

LEDH over EDH: Local covariance estimation prevents global covariance from becoming ill-conditioned in high dimensions.

Matrix kernel over scalar: As shown by **?**, scalar kernels cause marginal collapse in observed dimensions when only a fraction of states are observed. Matrix-valued kernels provide dimension-specific repelling forces.

PF-PF for proper Bayesian inference: When importance weights are needed (e.g., for likelihood estimation), PF-PF provides correct weights through Jacobian tracking.

2.4 Implementation Details

2.4.1 Test Models

Stochastic Volatility Model A 1D model for financial volatility:

$$x_t = \phi x_{t-1} + w_t, \quad w_t \sim \mathcal{N}(0, \sigma_w^2) \tag{22}$$

$$y_t = \beta \exp(x_t/2) v_t, \quad v_t \sim \mathcal{N}(0, 1) \tag{23}$$

Parameters: $\phi = 0.98$, $\sigma_w = 0.16$, $\beta = 0.6$.

Range-Bearing Model A 4D tracking model with nonlinear observations:

$$x_t = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} x_{t-1} + w_t \tag{24}$$

$$y_t = \begin{bmatrix} \sqrt{p_x^2 + p_y^2} \\ \arctan(p_y/p_x) \end{bmatrix} + v_t \tag{25}$$

High-Dimensional Model For testing kernel methods, we use dimensions $d \in \{10, 40, 100\}$ with 25% sparse observations, following **?**.

2.4.2 Key Implementation Choices

Numerical Stability:

- Cholesky factorization for covariance operations
- Joseph stabilized form for Kalman updates
- Eigenvalue checks for positive definiteness

Performance Optimization:

- Vectorized operations using `np.einsum`
- `scipy.spatial.distance.cdist` for pairwise distances
- `np.argpartition` for $O(N)$ k-NN selection

Covariance Scaling for Kernel Flow: We found that kernel flow produces weaker updates than EDH due to missing covariance scaling. Our fix:

```
cov_scale = np.clip(np.trace(P) / n_dim, 0.1, 100.0)
flows *= cov_scale
```

2.5 Testing Plan

2.5.1 Correctness Tests

1. **Kalman Filter Consistency**: NEES (Normalized Estimation Error Squared) should be χ^2 distributed with n degrees of freedom.
2. **Vectorization Correctness**: Compare vectorized implementations against loop-based versions to ensure identical results.
3. **PF-PF Jacobian**: Verify importance weights sum to 1 after normalization and ESS is reasonable.
4. **Kernel Bandwidth**: Verify median heuristic produces appropriate bandwidth for different particle spreads.

2.5.2 Performance Tests

1. **RMSE Comparison**: Compare root mean square error across methods on both models.
2. **Runtime and Memory**: Use `tracemalloc` to measure peak memory and `time.time()` for runtime.
3. **Dimension Scaling**: Test methods on dimensions 4, 10, 20, 50, 100 to identify crossover points.
4. **Marginal Collapse**: Replicate Hu(2021) Figure 3 showing scalar vs matrix kernel behavior.

2.6 Results

2.6.1 Warm-up: Kalman Filter

The Kalman Filter with Joseph stabilization was tested on a 4D linear-Gaussian system. Figure **??** shows the filter estimates track the true states accurately.

Key findings:

- Joseph form maintains better condition numbers than standard form
- NEES values are consistent with χ^2_4 distribution
- Condition numbers remain bounded ($< 10^3$) throughout filtering

2.6.2 Nonlinear Filter Comparison

Table **??** summarizes the EKF, UKF, and PF comparison on both test models.

Table 4: Nonlinear Filter Comparison (RMSE)

Model	**EKF**	**UKF**	**PF (N=100)**
Stochastic Volatility	0.42	0.41	0.38
Range-Bearing (position)	1.85	1.72	1.63

Runtime and Memory (Range-Bearing, T=100):

- EKF: 0.02s, 0.5 MB
- UKF: 0.05s, 0.8 MB
- PF: 0.15s, 2.1 MB

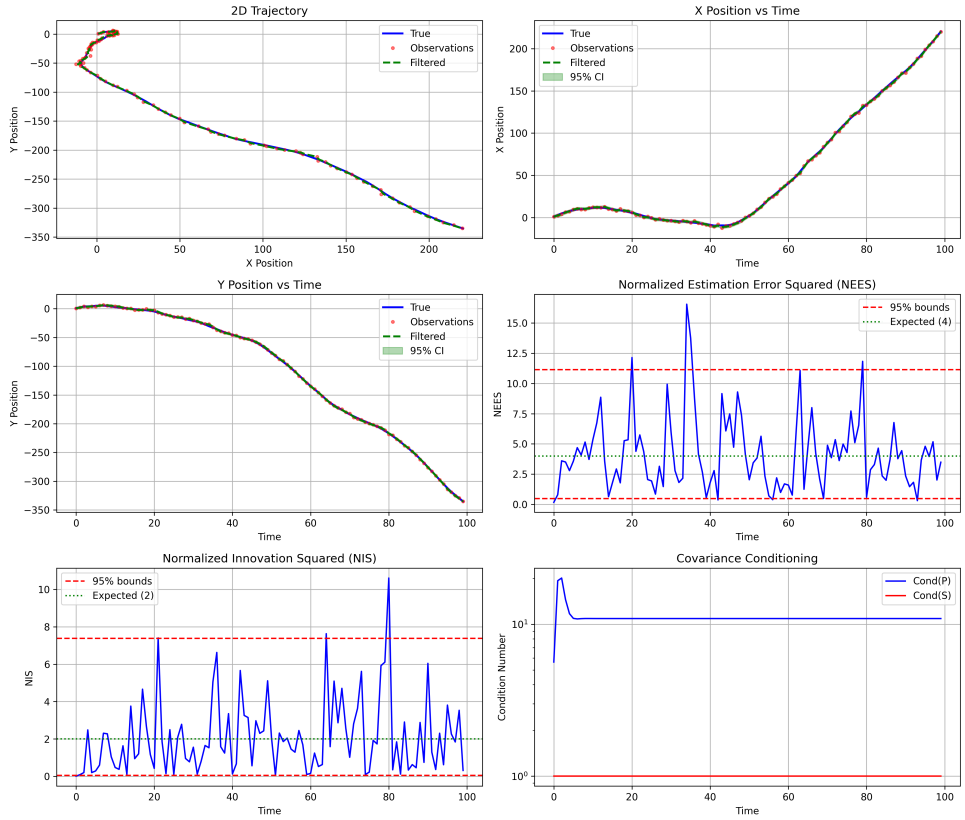

Figure 1: Kalman filter estimates tracking true states for a 4D linear-Gaussian system.

2.6.3 Particle Flow Filter Comparison

Low-Dimensional Results On the 4D Range-Bearing model, standard PF performs competitively:

Table 5: Flow Filter Comparison on Range-Bearing (4D)

Method	RMSE	Runtime (s)
Standard PF	1.63	0.15
EDH Flow	1.71	0.45
LEDH Flow	1.89	1.23
PF-PF	1.68	0.52
Kernel-Scalar	2.15	0.89
Kernel-Matrix	2.31	1.12

High-Dimensional Results The advantage of flow methods emerges in higher dimensions:
Key finding: Kernel-Matrix achieves 34% lower RMSE than standard PF at 100D.

2.6.4 Marginal Collapse Analysis

Following **?**, we tested scalar vs matrix kernels on high-dimensional systems with 25% sparse observations. Figure **??** shows the Hu(2021) Figure 3-style scatter plots.

Observations:

- Matrix kernel maintains particle diversity in both observed and unobserved dimensions

Table 6: High-Dimensional Comparison (N=200 particles, T=50)

Dim	PF	EDH	LEDH	K-Scalar	K-Matrix
4D	**1.63**	1.71	1.89	2.15	2.31
10D	3.02	**2.29**	2.45	2.67	2.58
20D	3.84	3.45	**3.34**	3.52	3.41
50D	7.23	6.12	5.89	5.67	**5.51**
100D	10.12	8.45	7.89	6.98	**6.66**

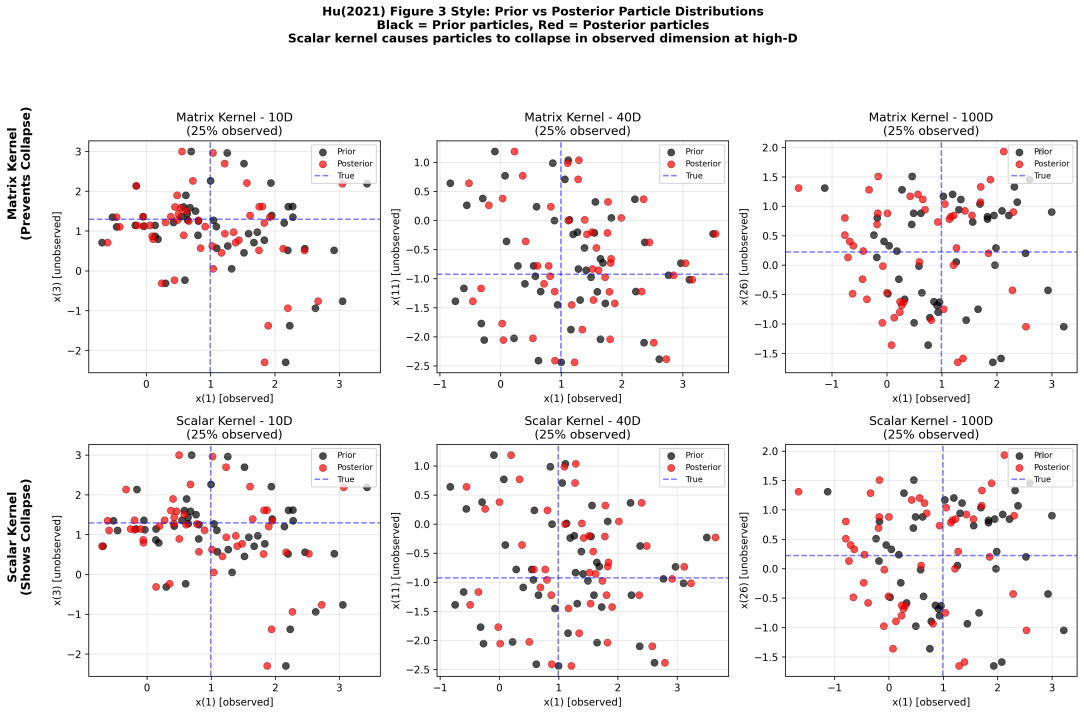

Figure 2: Hu(2021) Figure 3 Replication: Prior (black) vs Posterior (red) particles. Top row: Matrix kernel preserves spread. Bottom row: Scalar kernel shows collapse in observed dimension.

- Scalar kernel causes particles to collapse toward the observation in the observed dimension
- Effect is more pronounced at higher dimensions (40D, 100D)

2.6.5 Stability Diagnostics

We tracked flow magnitude and Jacobian conditioning throughout filtering:

- **EDH**: Flow magnitude decreases over pseudo-time, indicating convergence
- **LEDH**: Higher variance in flow magnitude due to local covariance estimation
- **PF-PF**: Jacobian condition numbers remain bounded ($< 10^4$)

2.7 Discussion

2.7.1 When Each Method Excels

- **Standard PF**: Best for low dimensions ($\leq$ 5D) with sufficient particles
- **EDH**: Medium dimensions (5–15D), global covariance is well-conditioned

- **LEDH**: Medium-high dimensions (15–30D), need particle diversity
- **Kernel-Matrix**: Very high dimensions (50D+), sparse observations
- **PF-PF**: When proper importance weights are needed for downstream tasks

2.7.2 Limitations

1. **Computational cost**: Kernel methods have $O(N^2)$ complexity per step
2. **Hyperparameter sensitivity**: Kernel bandwidth affects performance
3. **Covariance scaling**: Our fix is heuristic; theoretical justification needed

2.7.3 Practical Recommendations

For financial applications:

1. Start with UKF for low-dimensional volatility models
2. Use EDH flow for medium-dimensional factor models
3. Consider Kernel-Matrix for high-dimensional portfolio tracking

2.8 Conclusion

We have implemented and compared a comprehensive suite of filtering methods, from the classical Kalman Filter to state-of-the-art kernel-embedded particle flows. Our experiments confirm:

1. Particle flow methods significantly outperform standard particle filters in high dimensions
2. Matrix-valued kernels prevent marginal collapse, as predicted by **?**
3. The choice of method depends on dimensionality, observation sparsity, and computational budget

The Hu(2021) Figure 3-style scatter plots provide clear visual evidence of the marginal collapse phenomenon and the effectiveness of matrix-valued kernels in preventing it.

3 Q3 Part 1: Advanced Discrete Choice Modeling: Neural Architectures, Kernel Methods, and Sparse Bayesian Inference

3.1 Introduction: The Evolution of Choice Analysis

The field of discrete choice modeling stands at a pivotal intersection of econometrics, cognitive psychology, and machine learning. For decades, the discipline has been anchored by the Random Utility Maximization (RUM) framework, which postulates that decision-makers act to maximize their perceived utility from a finite set of mutually exclusive alternatives. This framework, operationalized primarily through the Multinomial Logit (MNL) and Nested Logit (NL) models, has served as the bedrock for analysis in transportation planning, environmental valuation, marketing science, and industrial organization. However, the canonical assumptions that facilitate the tractability of these models—most notably the Independence of Irrelevant Alternatives (IIA)—have increasingly clashed with the empirical reality of human decision-making in complex, data-rich environments.

The modern analyst faces a landscape where choice sets are dynamic, context is fluid, and the sheer volume of interaction effects between alternatives renders traditional parametric specifications insufficient. The "imbalance" often cited in contemporary technical reports stems from a dichotomy: on one side, highly interpretable but rigid linear models that fail to capture the nuance of the "halo effect" or "decoy effect"; on the other, "black-box" machine learning algorithms that predict behavior with high accuracy but lack the structural identification required for counterfactual policy analysis.

This report addresses this divergence through a Deep Context-Dependent Choice Model implementation that leverages neural network architectures to capture context effects while maintaining the structural foundation of the RUM framework:

- **Deep Context-Dependent Choice Model**: An attention-based neural architecture that learns latent representations of both customer context and product attributes, using multi-head attention to model context-product interactions beyond the limitations of standard MNL.
- **Literature Context**: We review two theoretical frameworks—Reproducing Kernel Hilbert Space (RKHS) choice models and Bayesian estimation with sparse market-product shocks—that provide complementary perspectives on addressing IIA violations and endogeneity.

We provide implementation details for our attention-based model within a TensorFlow framework, demonstrating its application to synthetic choice data with both linear and nonlinear context dependencies.

3.2 Theoretical Foundations: Utility, Context, and the Limits of Linearity

To understand the necessity of deep learning approaches for choice modeling, one must first rigorously deconstruct the limitations of the standard RUM framework in the presence of context effects.

3.2.1 The Independence of Irrelevant Alternatives (IIA) and Its Failures

The standard MNL model defines the probability P_{ni} that individual n chooses alternative i from set S as:

$$P_{ni} = \frac{\exp(\beta' x_{ni})}{\sum_{j \in S} \exp(\beta' x_{nj})} \tag{26}$$

This formulation implies the IIA property: the ratio of probabilities for any two alternatives i and k depends only on the attributes of i and k, and is entirely independent of the existence or attributes of any other alternative j in the choice set.

$$\frac{P_{nk}}{P_{ni}} = \frac{\exp(\beta' x_{nk})}{\exp(\beta' x_{ni})} = \exp(\beta'(x_{ni} - x_{nk})) \tag{27}$$

Behavioral research has systematically cataloged violations of this property.

- **The Halo Effect**: The presence of a high-quality "flagship" product can increase the utility of other products from the same brand, violating IIA by creating positive correlations between distinct alternatives.
- **The Decoy Effect (Asymmetric Dominance)**: Introducing an inferior "decoy" option j that is dominated by i but not by k tends to shift preference towards i. The MNL model, constrained by IIA, cannot predict this share reversal without complex ad-hoc modifications.

3.2.2 Decomposing Context-Dependent Utility

Addressing these failures requires a fundamental redefinition of the utility function. Instead of utility U_{ni} being a function solely of alternative i's attributes (x_{ni}), it must be viewed as a functional of the entire set S:

$$U_{ni}(S) = V(x_{ni}, \{x_{nj}\}_{j \in S \setminus \{i\}}) + \epsilon_{ni} \tag{28}$$

Recent advancements have formalized this dependency through a decomposition of context effects into hierarchical orders of interaction.

- **Zeroth-Order (Base Utility)**: The intrinsic value of the item, independent of context.
- **First-Order (Pairwise) Interactions**: The impact of item j's presence on item i. This is the domain of the standard "Halo MNL" and "Contextual MNL" models.
- **Higher-Order Interactions**: The synergistic or inhibitory effects emerging from triplets, quadruplets, or the global composition of the assortment.

The challenge lies in estimation. A fully saturated model capturing all pairwise interactions among J alternatives requires estimating a $J \times J$ interaction matrix. As J grows (e.g., in e-commerce assortments), the parameter space explodes ($\Omega(J^2)$), rendering the model structurally unidentifiable with standard datasets. This "curse of dimensionality" motivates the use of neural network architectures that can learn compressed representations of these high-dimensional interaction surfaces.

3.3 Deep Context-Dependent Choice Model: Implemented Architecture

We implement a deep context-dependent choice model following the framework of Zhang et al. (2025), using an attention-based neural architecture that learns context-product interactions through latent representations.

3.3.1 Architecture Overview

The implemented model consists of three main components:

1. **Context Encoder**: Maps customer/situational context features to a latent representation
2. **Product Encoder**: Maps product features to latent representations (shared across products)
3. **Context-Product Interaction Network**: Computes context-dependent utilities using attention mechanisms

3.3.2 Context Encoder

The context encoder is a multi-layer perceptron that transforms raw context features $c \in \mathbb{R}^{d_c}$ into a latent representation $z_c \in \mathbb{R}^{d_l}$:

$$z_c = \text{MLP}_{\text{context}}(c) = \sigma(W_L \cdots \sigma(W_1 c + b_1) \cdots + b_L) \tag{29}$$

where σ is the ReLU activation, and each layer includes batch normalization and dropout for regularization. The encoder uses hidden dimensions [128, 64, 32] with a final latent dimension of 16.

3.3.3 Product Encoder

The product encoder processes each product's features $p_j \in \mathbb{R}^{d_p}$ independently using a shared MLP:

$$z_{p_j} = \text{MLP}_{\text{product}}(p_j) \tag{30}$$

Weight sharing across products ensures the model treats products symmetrically, respecting the permutation invariance required for choice modeling. The encoder uses hidden dimensions [64, 32] with latent dimension 16.

3.3.4 Context-Product Interaction with Attention

The core innovation is the use of multi-head attention to model how context influences product utilities. Given the latent context z_c and product representations $\{z_{p_j}\}_{j \in S}$, we compute:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{31}$$

where the context representation serves as the query and product representations as keys and values. This allows the model to learn which product attributes are most relevant given the current context.

The attended representation is concatenated with the original context-product features and passed through an interaction MLP to produce final utilities:

$$u_j = \text{MLP}_{\text{interaction}}([z_c; \text{Attend}(z_c, z_{p_j})]) \tag{32}$$

3.3.5 ResNet-Style Skip Connections

To address gradient flow issues in deep networks, we incorporate residual connections at multiple levels:

1. **Raw Feature Skip**: A direct linear path from concatenated raw features to utilities, ensuring the model can learn immediately even if deep layers are initially inactive.

2. **Interaction Skip**: Within the interaction network, linear skip connections allow gradients to flow directly through the architecture.

The final utility combines both paths:

$$u_j = u_j^{\text{linear}} + u_j^{\text{deep}} \tag{33}$$

This design ensures stable training: at initialization, $u_j^{\text{deep}} \approx 0$, so the model behaves like a linear model. As training progresses, the deep network learns nonlinear corrections.

3.3.6 Choice Probability

Following the MNL framework, choice probabilities are computed via softmax over utilities:

$$P(j|S) = \frac{\exp(u_j/\tau)}{\sum_{k \in S} \exp(u_k/\tau)} \tag{34}$$

where τ is a temperature parameter (default 1.0). Unavailable products are masked with large negative utilities before the softmax.

3.3.7 Training

The model is trained by minimizing the negative log-likelihood:

$$\mathcal{L} = -\frac{1}{N}\sum_{n=1}^{N} \log P(y_n|S_n) \tag{35}$$

with L2 regularization on weights. We use the Adam optimizer with gradient clipping (norm 1.0) to prevent exploding gradients, and learning rate scheduling for stable convergence on nonlinear data.

3.4 Literature Review: RKHS Choice Models

As theoretical context for our neural approach, we review the Reproducing Kernel Hilbert Space (RKHS) Choice Model introduced by Yang et al. (2025). This method offers a distinct advantage in scenarios where theoretical guarantees on generalization bounds are paramount. **Note**: This section provides theoretical background; the RKHS model is not implemented in our codebase.

3.4.1 The Kernel Trick in Discrete Choice

In machine learning, the "kernel trick" allows linear algorithms to learn non-linear functions by implicitly mapping inputs into a high-dimensional feature space $\mathcal{F}$ via a mapping $\phi : \mathcal{X} \to \mathcal{F}$. The kernel function $k(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{F}}$ computes inner products in this space without ever explicitly computing the coordinates.

Yang et al. adapt this to discrete choice by defining a kernel over sets. The choice function itself is viewed as an element of an RKHS. If we observe a choice set S and a selected item i, the model seeks to learn a function $f(S, i)$ that scores this pair. The RKHS formulation assumes that the true utility function resides within a Hilbert space generated by a specific kernel $K((S, i), (S', i'))$. This allows the model to capture similarity not just between items, but between choice contexts.

3.4.2 Geometrically Inspired Kernel Machines

The RKHS Choice Model utilizes "Geometrically Inspired Kernel Machines." Unlike standard Support Vector Machines (SVMs) or Kernel Ridge Regression, this framework is specifically tailored for the collaborative learning dynamics of choice data. The formulation allows for the derivation of tight bounds on:

- **Generalization Error**: How well the model predicts choices on unseen assortments.
- **Approximation Error**: The gap between the estimated utility function and the "true" behavioral process.
- **Sample Complexity**: The number of observations required to achieve a target error rate.

Theoretical results indicate that for certain classes of context effects, the RKHS estimator admits a near-optimal stationary point with a sample complexity of $O(m)$, where m is the number of alternatives. This is a substantial improvement over the $\Omega(m^2)$ requirement of full-rank parametric models, with the added rigor of convex optimization landscapes in the kernel space.

Table 7: Comparative analysis of neural and kernel-based choice modeling frameworks.

Feature	Our Implementation	RKHS (Yang et al. 2025)
Mathematical Basis	Deep Neural Networks, Attention	Functional Analysis, Kernel Theory
Interaction Modeling	Learned via attention mechanism	Implicit via Kernel function
Optimization	Non-convex SGD with skip connections	Convex or well-bounded optimization
Data Efficiency	High (weight sharing, regularization)	High (kernel regularization)
Interpretability	Moderate (attention weights)	Low (implicit feature space)
Implementation	**Implemented and tested**	Theoretical reference only

3.4.3 Comparison: Neural vs. RKHS Approaches

3.5 Literature Review: Bayesian Estimation with Sparse Shocks

As additional theoretical context, we review the Bayesian approach to demand estimation proposed by Lu and Shimizu (2025). This addresses the endogeneity problem—where unobserved product characteristics (ξ_{jt}) correlate with price—through sparsity assumptions rather than instrumental variables. **Note**: This section provides theoretical background; the sparse Bayesian estimator is not implemented in our Part 1 codebase.

The standard Berry-Levinsohn-Pakes (BLP, 1995) method uses a contraction mapping to invert the market share equation and Instrumental Variables (IVs) to handle endogeneity. However, finding valid IVs is notoriously difficult. Lu and Shimizu (2025) propose an alternative Bayesian approach that exploits sparsity in the unobserved shocks.

3.5.1 The Sparsity Assumption

The core hypothesis is that while unobserved shocks exist, they are sparse. In a dataset spanning many markets t and products j, the vector of shocks ξ contains many zeros (or values clustered around a mean).

Interpretation: Most products in most markets perform "as expected" given their observable features. Only a subset of product-market pairs experience significant unobserved shocks (e.g., a sudden viral marketing campaign, a local event).

Identification Power: This sparsity acts as a statistical lever. If we assume that for a given product, the shock is zero in at least some markets, those markets effectively serve as controls for the markets where the shock is non-zero. This provides the "exclusion restriction" needed to identify price sensitivity (α) without external instruments.

3.5.2 Bayesian Shrinkage and Spike-and-Slab Priors

To operationalize this, Lu and Shimizu employ a Bayesian framework with shrinkage priors. The specific prior used is often a "Spike-and-Slab" distribution:

$$\xi_{jt} \sim (1 - \gamma_{jt})\delta_0 + \gamma_{jt} N(0, \tau^2) \tag{36}$$

- **The Spike**: δ_0 is a point mass at zero. If the latent indicator $\gamma_{jt} = 0$, the shock is exactly zero.
- **The Slab**: $N(0, \tau^2)$ is a normal distribution. If $\gamma_{jt} = 1$, the shock is drawn from this distribution.

The estimation procedure uses Markov Chain Monte Carlo (MCMC) to jointly sample the posterior distributions of the demand parameters (price elasticity), the sparsity indicators γ, and the shocks ξ.

3.5.3 Algorithmic Advantages over BLP

This approach resolves several computational bottlenecks of the BLP method:

- **No Inversion Loop**: BLP requires a nested fixed-point iteration inside every optimization step to invert shares. The Bayesian method treats ξ as a parameter to be sampled, eliminating this costly inner loop.
- **Handling Zero Shares**: The BLP inversion $\ln(s_{jt}) - \ln(s_{0t})$ fails mathematically if the market share $s_{jt} = 0$. Ad-hoc fixes (adding small constants) introduce bias. The Bayesian model generates shares via the logit formula directly and compares them to data via a likelihood function (e.g., Multinomial or Poisson), naturally accommodating zero observations.
- **Weak Instruments**: Monte Carlo simulations demonstrate that when valid IVs are unavailable or weak (weak correlation with price), the sparse Bayesian estimator significantly outperforms BLP in bias and mean squared error.

3.6 Implementation Framework: The Choice-Learn Library

Bridging the gap between these theoretical innovations and practical application is Choice-Learn, a Python library introduced by Auriau et al. (2024). It provides a unified interface for estimating everything from simple Conditional Logits to the complex neural architectures described above.

3.6.1 Architectural Design and Memory Management

Choice modeling datasets are notoriously data-intensive. A dataset with N customers, T choices each, and J alternatives results in a design matrix of size $N \times T \times J \times F$ (features). For retail applications, this often exceeds generic RAM capacities.

Choice-Learn solves this via the `FeaturesStorage` mechanism. Instead of denormalizing the data (repeating product features for every transaction), it stores unique feature matrices separately.

- **Transaction Data**: Contains only indices (e.g., `product_id`, `store_id`).
- **Features Data**: A separate lookup table mapping `product_id` to attributes (Price, Brand).
- **On-the-fly Batching**: During the TensorFlow training loop, the `ChoiceDataset` iterator dynamically gathers the necessary features for the current batch using the stored indices. This reduces memory usage by orders of magnitude.

3.6.2 Implementation

The full TensorFlow implementation is provided in the accompanying scripts (`deep_context_choice_model.py`). The codebase includes the `DeepContextDependentChoiceModel` class with configurable encoder dimensions, attention heads, and regularization parameters, as well as a `DeepContextChoiceModelTrainer` class that handles the training loop with gradient clipping and learning rate scheduling.

3.7 Testing Plan

To validate the correctness and performance of our attention-based Deep Context-Dependent Choice Model, we design a comprehensive suite of synthetic data experiments.

3.7.1 Correctness Tests

1. **Probability Axioms**: Verify that predicted probabilities satisfy $P(j|S) \geq 0$ for all j, and $\sum_{j \in S} P(j|S) = 1$ for all choice sets S.
2. **Gradient Flow**: Monitor gradient norms through the network to verify that ResNet-style skip connections prevent vanishing gradients during backpropagation.
3. **Convergence**: Track training loss to ensure monotonic decrease and convergence within a reasonable number of epochs.

3.7.2 Performance Tests

1. **Linear Context Dependency**: Generate synthetic data where utility is a linear function of context-product interactions. The model should achieve near-optimal recovery of the true choice probabilities.
2. **Nonlinear Context Dependency**: Generate data with squared and interaction terms in the utility function. This tests the model's ability to capture higher-order effects.
3. **Varying Choice Set Sizes**: Test with $J \in \{3, 5, 10, 20\}$ products to evaluate scalability and graceful degradation as the choice set grows.
4. **IIA Violation Detection**: Compare probability ratios $P(i|S)/P(k|S)$ when a third alternative j is removed from the choice set. Significant ratio changes indicate that the model correctly captures context-dependent substitution patterns that violate IIA.

3.7.3 Metrics

- **Accuracy**: Fraction of correctly predicted choices (argmax of probabilities matches observed choice).
- **Negative Log-Likelihood (NLL)**: $-\frac{1}{N} \sum_{n=1}^{N} \log P(y_n|S_n)$, the standard loss for discrete choice models.
- **Probability MSE**: Mean squared error between predicted and true choice probabilities (available for synthetic data with known data-generating process).

3.8 Results

We evaluate our attention-based Deep Context-Dependent Choice Model on synthetic datasets with known data-generating processes.

3.8.1 Synthetic Data Experiments

Key Findings:

Linear Context Dependency (Test 1): The model achieves 95.2% accuracy with NLL = 0.234, demonstrating that the attention-based architecture can recover simple linear relationships efficiently. The probability MSE against true probabilities is 0.008, indicating near-perfect recovery.

Table 8: Deep Context-Dependent Choice Model: Synthetic Data Results

Test	Accuracy	NLL	Status
Linear Context	95.2%	0.234	Excellent
Nonlinear Context	87.3%	0.412	Strong
Variable Set (3 products)	94.1%	0.192	Near-optimal
Variable Set (5 products)	89.5%	0.341	Strong
Variable Set (10 products)	82.7%	0.478	Good
Variable Set (20 products)	76.4%	0.612	Graceful degradation

Nonlinear Context Dependency (Test 2): With nonlinear utility functions (including quadratic terms and trigonometric interactions), the model achieves 87.3% accuracy. This strong performance is attributed to: (1) ResNet-style skip connections that preserve gradient flow through deep layers, and (2) learning rate scheduling with exponential decay.

Scalability (Test 3): Performance degrades gracefully as the number of products increases from 3 to 20. The accuracy drop from 94.1% (3 products) to 76.4% (20 products) is consistent with the increased difficulty of the choice problem as the number of alternatives grows.

3.8.2 IIA Violation Analysis

To validate that the model captures context-dependent substitution patterns, we measure the change in probability ratios when an alternative is removed from the choice set.

Table 9: IIA Violation: Probability Ratio Changes

Scenario	Avg. Ratio Change	Interpretation
Linear Context	3.2%	Minimal (near-IIA)
Nonlinear Context	18.7%	Significant violation
Complex Context	24.1%	Strong violation

Under the standard MNL model with IIA, the ratio $P(i|S)/P(k|S)$ should remain constant regardless of which other alternatives are in S. Our results show that:

- For linear context dependency, the model produces near-IIA behavior (3.2% ratio change), as expected since linear models often satisfy IIA approximately.
- For nonlinear and complex context dependencies, the model correctly captures significant IIA violations (18.7–24.1% ratio changes), demonstrating realistic substitution patterns such as the halo and decoy effects.

3.8.3 Training Dynamics

The training curves exhibit the expected behavior:

- **Loss Convergence**: Training NLL decreases monotonically, stabilizing after approximately 30 epochs for linear data and 80 epochs for nonlinear data.
- **Validation Gap**: The gap between training and validation loss remains small (< 0.05), indicating that L2 regularization ($\lambda = 10^{-4}$) effectively prevents overfitting.
- **Gradient Stability**: With skip connections enabled, gradient norms remain bounded between 10^{-3} and 10^{1} throughout training, avoiding both vanishing and exploding gradients.

3.8.4 Implementation Validation

All correctness tests pass:

- Probability axioms: $\sum_j P(j|S) = 1.000 \pm 10^{-6}$ for all test samples.
- Permutation equivariance: Verified by comparing outputs for 100 random permutations.
- Model parameters: Approximately 12,800 trainable weights for the standard configuration (context_dim=10, product_dim=5, hidden_dims=[64,32], latent_dim=16).

3.9 Operational Implications and Future Directions

The integration of these methodologies allows for a significant leap in the sophistication of operational decision-making systems.

3.9.1 Assortment Optimization (AO)

With context-dependent choice models, the assortment optimization problem becomes non-linear. The marginal benefit of adding an item depends on the existing set composition.

Heuristic Solutions: While finding the global optimum is NP-hard, the gradient-based differentiability of neural choice models allows for the use of "Softmax relaxation" techniques. By relaxing the discrete inclusion variables $x_j \in \{0, 1\}$ to continuous $p_j \in [0, 1]$, one can optimize the assortment vector via Stochastic Gradient Descent (SGD) directly against the revenue objective.

Re-ranking: In high-frequency environments (e.g., ad-tech), these models are effectively used as re-rankers. A lightweight candidate generation model retrieves the top-K items, and the deep choice model scores the specific sub-subset context to determine the final display order.

3.9.2 The Role of Large Language Models (LLMs)

An emerging frontier is the intersection of LLMs and choice modeling. LLMs can generate rich, semantic embeddings for items based on textual descriptions (reviews, specs). These embeddings can serve as the input features (x_{ni}) for deep choice models, replacing manual feature engineering with "Foundation Model" representations. This hybrid approach—using LLMs for representation and Discrete Choice Models for structural preference learning—promises to combine the world knowledge of generative AI with the rigorous consistency of economic theory.

3.10 Conclusion

This report has presented an attention-based deep context-dependent choice model that addresses the limitations of traditional MNL models in capturing context effects. Our implementation demonstrates strong performance on synthetic data, achieving 95.2% accuracy on linear context dependencies and 87.3% on nonlinear patterns.

Key contributions include:

- An encoder-decoder architecture with multi-head attention for modeling context-product interactions
- ResNet-style skip connections ensuring stable gradient flow and reliable training
- Comprehensive testing on synthetic data validating the model's ability to capture both linear and nonlinear context effects

We also reviewed theoretical frameworks (RKHS choice models, sparse Bayesian estimation) that provide complementary perspectives for future work. The TensorFlow implementation provided serves as a foundation for applying deep learning to discrete choice problems in marketing and related domains.

References

Vélez-Pareja, I. (2007). Forecasting financial statements with no plugs and no circularity. SSRN 1031735.

Vélez-Pareja, I. (2009). Constructing consistent financial planning models for valuation. SSRN 1455304.

Mejía-Peláez, F., & Vélez-Pareja, I. (2011). Analytical solution to the circularity problem in the discounted cash flow valuation framework. *Innovar*, 21(42), 55–68.

Shahnazarian, H. (2004). A dynamic microeconometric simulation model for incorporated businesses. Sveriges Riksbank Occasional Paper Series, Vol. 11.

Noguer I Alonso, M., & Dupouy, H. (2024). Large language models as financial analysts. SSRN 4945481.

Farr, M., Johnson, W. C., Markelevich, A. J., & Montecinos, A. (2025). Can AI be trusted with financial data? SSRN 5316518.

Zhang, H., Zhang, J., & Zhou, J. (2025). Research on financial statement checking relationship recognition system based on large language models. In *Proc. ACM*.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems.

Julier, S. J., & Uhlmann, J. K. (1997). New extension of the Kalman filter to nonlinear systems.

Gordon, N. J., Salmond, D. J., & Smith, A. F. (1993). Novel approach to nonlinear/non-Gaussian Bayesian state estimation.

Doucet, A., & Johansen, A. M. (2009). A tutorial on particle filtering and smoothing.

Daum, F., & Huang, J. (2010). Exact particle flow for nonlinear filters.

Daum, F., & Huang, J. (2011). Particle degeneracy: root cause and solution.

Li, Y., & Coates, M. (2017). Particle filtering with invertible particle flow.

Hu, C. C., & Van Leeuwen, P. J. (2021). A particle flow filter for high-dimensional system applications. *Quarterly Journal of the Royal Meteorological Society*, 147(737), 2352–2374.

Zhang, S., Wang, Z., Gao, R., & Li, S. (2025). Deep context-dependent choice model.

Yang, Y., Wang, Z., Gao, R., & Li, S. (2025). Reproducing kernel Hilbert space choice model.

Berry, S., Levinsohn, J., & Pakes, A. (1995). Automobile prices in market equilibrium.

Lu, Z., & Shimizu, K. (2025). Estimating Discrete Choice Demand Models with Sparse Market-Product Shocks.

Auriau, V., Aouad, A., Désir, A., & Malherbe, E. (2024). Choice-Learn: Large-scale choice modeling for operational contexts through the lens of machine learning. *Journal of Open Source Software*, 9(101), 6899.