



From Models to Markets: Building a Smarter Market Maker on Kalshi

Introduction

Kalshi is a federally regulated prediction market where users can trade real-money contracts on real-world events — everything from inflation rates to presidential elections. With market structure similar to traditional exchanges, Kalshi presents a unique opportunity for quantitative strategies.

We set out to build an automated **market-making bot** for Kalshi that could:

- Quote buy/sell prices in real time
- Adapt to volatility and liquidity
- Manage risk autonomously
- Scale across many markets

This post outlines our journey — from infrastructure and modeling to backtesting and live optimization.

Laying the Foundation: Real-Time Infrastructure

Our first challenge was integrating with Kalshi's API, which uses RSA-PSS signatures for each request. We built a robust client that:

- Authenticates via public/private key pairs
- Supports both REST and WebSocket connections
- Implements retry logic, pagination, and rate-limiting

With this foundation, we could:

- Pull **market and trade data**
 - Stream **live orderbooks**
 - Place and cancel **orders programmatically**
-

Modeling Execution: Smarter Quoting with Avellaneda–Stoikov

We based our quoting strategy on a modified version of the **Avellaneda–Stoikov model**, a well-known framework for optimal market making. This model sets bid and ask prices based on:

- **Mid-price** from the current orderbook
- **Market volatility** (σ), computed from recent trades
- **Inventory levels** (q), to reduce skew
- **Risk aversion** (γ) and **order flow sensitivity** (κ), as tunable parameters

The result: quotes that were dynamic, probabilistically profitable, and inventory-aware.

Managing Risk: When Not to Trade Matters More

We implemented a flexible risk manager that enforced:

- **Volatility-aware position limits**: reduced exposure in turbulent markets
- **Dynamic sizing**: smaller orders during spikes, larger when calm
- **Trading halts**: paused quoting when near position limits

This enabled the bot to maintain neutral inventories and avoid runaway exposure while still being responsive.

Simulation & Backtesting: Testing Before Trusting

To validate and optimize the model, we developed a full **event-driven backtester** that replayed historical orderbooks and trades.

Key features:

- Simulated fills based on quote prices and market trades
- Estimated volatility from a rolling window
- Evaluated **PnL**, **inventory drift**, and **quote performance**

This simulation framework became our lab — where we ran grid searches over 1,000+ parameter pairs and gathered actionable insights.

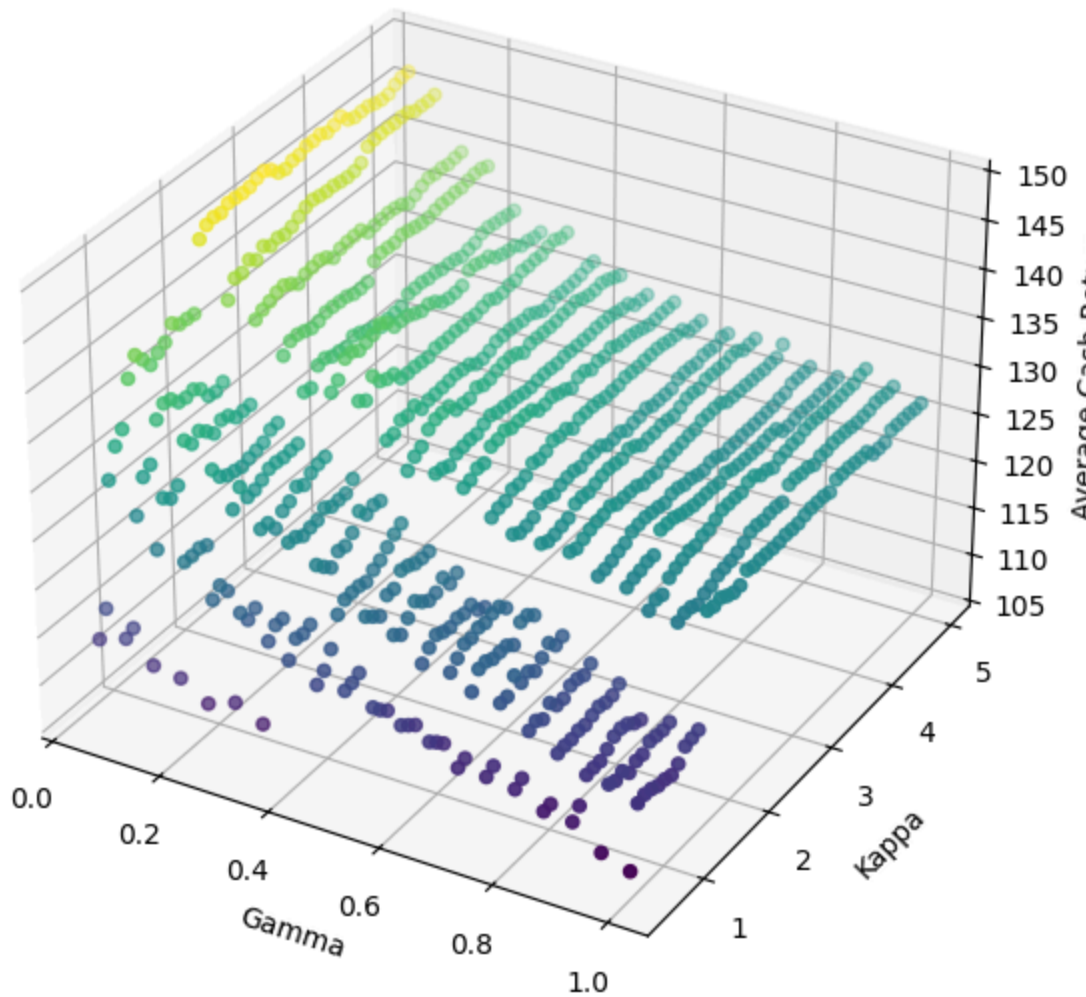
Parameter Optimization: What the Data Told Us

Using real Kalshi data across many markets, we ran a grid search over combinations of:




- **Gamma (γ)**: risk aversion, from 0.05 to 1.0
- **Kappa (κ)**: order flow responsiveness, from 0.5 to 5.0


Here's what we found:

Overall Returns vs Gamma and Kappa



Key Takeaways:

-  **Best results came from low γ (0.05–0.25) and high κ (3.0–5.0).** These settings generated tighter quotes and higher fill rates without letting inventory spiral.
-  **High gamma (> 0.5)** severely reduced profits. These conservative strategies posted wide spreads and often sat idle.
-  **Low kappa (< 1.0)** consistently underperformed, likely underestimating fill probabilities and posting timid quotes.

-  The optimal zone clustered around $\gamma \approx 0.15$, $\kappa \approx 4.0$, where return was maximized while keeping inventory neutral.
-

Market Selection: Picking the Right Battles

Not all markets are worth quoting. We designed a scoring system that filtered candidates based on:

- 24-hour **volume**
- Estimated **volatility**
- Current **spread width**







We used a scoring formula similar to:

$$\text{score} = (\text{volume} / \text{volatility}) \times \exp(-((\text{spread} - \text{target})^2) / 2\sigma^2)$$

This helped us focus the bot on markets where conditions were ideal — liquid enough to trade, volatile enough to profit, and calm enough to control risk.

From Research to Deployment

The final version of the bot includes:

-  Real-time WebSocket pricing
-  Order placement + cancellation
-  Volatility-aware quoting logic
-  Inventory-based rebalancing
-  Live market filtering
-  Simulation mode

Final Thoughts

This project taught us how to bridge the gap between **theory and execution**. While prediction markets like Kalshi are relatively new, they behave much like traditional financial instruments — and reward the same kinds of discipline: rigorous modeling, data-driven tuning, and conservative risk management.

There's plenty left to explore — reinforcement learning, inter-market arbitrage, latency-aware execution — but we're confident this foundation can scale and adapt with the market.